МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2019 г.

Разработка веб-приложения для управления командами, проектами и задачами

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Е.С. Ярош
«___»_____2019 г.

Автор работы,
студент группы КЭ-452
_____Д.И. Ураканов
«___»_____2019 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2019 г.

Челябинск-2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2019 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-452
Ураканову Денису Ивановичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка веб-приложения для управления командами, проектами и задачами» утверждена приказом по университету от 25 апреля 2019 г. № 899

2. **Срок сдачи студентом законченной работы:** 1 июня 2019 г.

3. **Исходные данные к работе:**

   Обеспечить основной функционал приложения:

   1. Организация проектов, списков задач и отдельных задач.
   2. Создание команд с собственными проектами или без таковых.
   3. Представление списков задач и проектов в виде графов.
   4. Создание и отображение оптимального выполнения списка задач.

5. Создание сообщества, в котором пользователи могут подавать заявки на участие в проекте или быть частью команды.
6. Взаимодействие проектов.

Среда и средства реализации – по выбору разработчика.

4. **Перечень подлежащих разработке вопросов:**

– анализ современного подхода к взаимодействию между членами командами, изучение способы гибкой работы небольших групп;
– изучение взаимодействия коммерческих организаций или инвесторов и команд-разработчиков какого-либо продукта;
– анализ-обзор родственных разработок;
– выбор методологии решения задачи;
– выбор средств реализации задачи;
– проектирование архитектуры приложения;
– организация базы данных;
– программная реализация;
– тестирование разработанного приложения.

5. **Дата выдачи задания:** 1 декабря 2018 г.

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*Д.И. Ураканов*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	05.02.2019	
Разработка модели, проектирование	15.03.2019	
Реализация системы	20.04.2019	
Тестирование, отладка	15.05.2019	
Компоновка текста работы и сдача на нормоконтроль	24.05.2019	
Подготовка презентации и доклада	30.05.2019	

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*Д.И. Ураканов*/

Аннотация

Д.И.Ураканов. Разработка веб-приложения для управления командами, проектами и задачами. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2019, 216 с., 101 ил., библиогр. список –23 наим.

В рамках выпускной квалификационной работы выполнены все этапы, связанные с созданием программного комплекса по управлению командами, проектами и задачами. Описаны процессы коммуникации новых команд, приходящих на рынок, состоявшихся организаций и обычных разработчиков, выявлены недостатки имеющихся на рынке программных продуктов, обеспечивающих коммуникативные функции проектных команд, сформулированы требования к разработке. Произведен детальный анализ современных технологий создания веб-приложений, выбраны технологии на основе языка программирования Java - Java Spring Framework 5, оболочки Java Spring Boot и языков программирования и разметки JavaScript, HTML и CSS. За основу модели взаимодействия команд взята философия Agile-разработки, в частности, фреймворки Scrum и Kanban. Заложены возможности оптимизации проекта по критериям времени, сложности, материальных затрат. В качестве математического аппарата для реализации связей и иерархий веб-приложения были выбраны графы. Выполнено модульное тестирование и альфа-тестирование разработанного программного комплекса.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день отсутствует удобная возможность взаимодействия на рынке состоявшихся организаций, начинающих стартап-команд и приходящих в определенную сферу новых людей, ищущих определенный проект или команду по их интересам. Актуальность темы, связанной с данным взаимодействием, заключается в том, что в современном мире Интернета и облачных технологий разные слои людей на рынке могут достаточно легко коммуницировать друг с другом, создавая новые инновационные продукты. За последнее десятилетие большие организации нередко выкупали команды разработчиков того или иного инновационного приложения или программы. На более низком уровне часто стартап-командам требуются единомышленники, еще не нашедшие постоянное место работы.

На основании вышеизложенного можно утверждать, что инструмент, предоставляющий возможности нахождения методов гибкой и наиболее быстрой работы команды и её членов, будет весьма полезен и востребован. Гибкость предполагается также в способах взаимодействий разных проектов.

Целью представленной выпускной квалификационной работы является разработка веб-приложения, обеспечивающего систему взаимодействий команд, проектов и задач с использованием технологии Java Spring, базы данных PostgreSQL и JavaScript фреймворков(Vue JS).

Для достижения поставленной цели, необходимо решить следующие задачи:

1. Рассмотреть существующие на рынке проекты или научные исследования по теме работы.
2. Провести детальный анализ найденных проектов с точки зрения их достоинств и недостатков, учесть полученные результаты при разработке.

3. Определить основные цели и назначение проекта. Определить основной функционал.
4. Сформулировать основные сущности проекта.
5. Выбрать средства реализации проекта.
6. Разработать схему данных.
7. Разработать серверную часть приложения, опираясь на ранее созданную схему данных. Разработать фронт-энд сторону приложения.
8. Протестировать итоговую версию продукта.

На протяжении всей работы должны проводиться тесты после каждого из практических пунктов (начиная со схемы данных).

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

Софтверный рынок, связанный с управлениями задачами и командами, является насыщенным различными категориями продуктов. Их общая особенность состоит в том, что они предоставляются какую-либо возможность автоматизировать работу на уровне представления задач и их параметров.

Хотя количество приложений на рынке действительно большое для разных платформ (Web, Android, iOS, Windows) их можно свести к следующей классификации.

1) CRM – системы
   Например: Salesforce.

Ссылка на страницу с контактными данными разработчика ресурса и описанием веб-приложения: https://www.salesforce.com/.

Ниже представлена таблица 1.1 с преимуществами и недостатками веб-приложения:

Таблица 1.1 – Преимущества и недостатки CRM – систем

Преимущества	**Недостатки**
Все «облака» и продукты Salesforce взаимно дополняют друг друга, что позволяет создавать целостную бизнес-систему.	Данные CRM системы ориентируются на работу непосредственно с клиентом, а не на проект как таковой.
Анализ потока и оттока клиентов определенного продукта.	Такие системы платны.

Продолжение таблицы 1.1

Преимущества	Недостатки
Опираясь на статистику и анализ, можно прогнозировать выручку и планировать развитие предприятия.	Ориентируются на уже состоявшийся бизнес.
	Ориентация на работу отдела продаж, учет и анализа статистики движения заказов и планирование развития предприятия, а не на развитие технологически непосредственно самого продукта.

2) Программы для управления проектами
   Пример: Trello, Notion

Ссылки на страницы с контактными данными разработчика ресурса и описанием веб-приложения Trello и Notion соответственно: https://trello.com/, https://www.notion.so.

Ниже представлена таблица 1.2 с преимуществами и недостатками веб-приложения:

Таблица 1.2 – Преимущества и недостатки программ для управления проектами

Преимущества	Недостатки
Возможность декларативно управлять проектами.	Отсутствие развитой системы интеграции и коммуникации с проектами и задачами других команд и организаций.

Продолжение таблицы 1.2

Преимущества	Недостатки
Возможность декларативно управлять командами.	
Бесплатны в своей базовой версии (полная версия платная).	

3) Системы отслеживания ошибок и управления проектами
   Пример: Jira

Ссылка на страницу с контактными данными разработчика ресурса и описанием веб-приложения Jira: https://ru.atlassian.com/software/jira.

Ниже представлена таблица 1.3 с преимуществами и недостатками веб-приложения:

Таблица 1.3 – Преимущества и недостатки систем управления проектами отслеживания ошибок

Преимущества	Недостатки
Ориентация на Agile философию.	Основная ориентация на работу с задачами и исправлениями ошибок в самом проекте, приложение не создано для коммуникации разных проектов.
Управление проектами.	В основном подходит лишь для ИТ-сферы.
Управление командами.	Системы предоставляет обширную и детальную работу внутри определенного проекта, но без возможности интеграции с другими.

Продолжение таблицы 1.3

Возможность интеграции с системами контроля версий, такие как GitLab, GitHub и др..	
Возможность установки приоритетов в выполнении задач.	

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

1.2.2. Базы данных

Работа веб-приложения требует для хранения информации базу данных.

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных. [1, с.138]

Система управления базами данных – это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д. В общем СУБД – это система, позволяющая создавать базы данных и манипулировать сведениями из них. А осуществляет этот доступ к данным СУБД посредством специального языка-SQL. [2, с.138]

SQL – язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

При выборе базы данных на сегодняшний день сначала выбирают будет ли она реляционной или нереляционной.

Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая стока таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. В таблице 1.4 указанной ниже представлены достоинства и недостатки реляционных баз данных. [3, с.138]

Таблица 1.4 – Достоинства и недостатки реляционных баз данных

Достоинства	**Недостатки**
Эта модель данных отображает информацию в наиболее простой для пользователя форме.	Относительно медленный доступ к данным.
Основана на развитом математическом аппарате, который позволяет достаточно лаконично описать основные операции над данными.	Трудность в переводе в таблицу сложных отношений.
Соответствие требованиям ACID(атомарность, единообразие, изолированность и надежность).	Требуется относительно большой объем памяти.

В базах данных NoSQL (т.е. нереляционные базы данных) для доступа к данным и управления ими применяются различные модели данных, в том числе

документная, графовая, поисковая, с использованием пар «ключ-значение» и хранением данных в памяти. Базы данных таких типов оптимизированы для приложений, которые работают с большим объемом данных, нуждаются в низкой задержке и гибких моделях данных. Все это достигается путем смягчения жестких требований к непротиворечивости данных, характерных для других типов БД. Ниже представлена таблица 1.5, отображающая преимущества и недостатки нереляционных баз данных. [4, с. 138]

Таблица 1.5 – Преимущества и недостатки нереляционных баз данных

Преимущества	**Недостатки**
В базах данных NoSQL применяются различные модели данных, в том числе документные, графовые, поисковые, с использованием пар «ключ-значение» и хранением данных в памяти.	Приложение сильно привязывается к конкретной СУБД. Язык SQL универсален для всех реляционных хранилищ и поэтому в случае смены СУБД не придётся переписывать весь код.
Базы данных NoSQL обычно поддерживают высокую разделяемость благодаря шаблонам доступа на основе пар «ключ-значение» с возможностью масштабирования на основе распределенной архитектуры. Это повышает пропускную способность и обеспечивает устойчивую производительность почти в неограниченных масштабах.	Ограниченная емкость встроенного языка запросов. SQL имеет очень богатую историю и множество стандартов. Это очень мощный и сложный инструмент для операций с данными и составления отчетов. Практически все языки запросов и методы API хранилищ NoSQL были созданы на основе тех или иных функций SQL, но они имеют куда меньшую функциональность.

Продолжение таблицы 1.5

Преимущества	Недостатки
	NoSQL базы данных частично или полностью не следуют ACID в угоду масштабируемости и производительности, из-за этого могут быть проблемы с согласованностью данных в критических для определенных мест структур.

Для решения поставленной задачи не требуется действительно большая масштабируемость, во всяком случае на данном этапе разработки и поддержки приложения. А также предметная область поставленного задания удобно ложится на табличное представление. На текущей стадии работы над проектом работа с NoSQL базами данных лишь потребует большого количества времени на изучение определенной СУБД, в то время как SQL диалекты различных реляционных баз во многом даже идентичны. Необходимо выбрать реляционную базу данных для проекта.

В таблице 1.6 представим сравнительную характеристику самых популярных баз данных.

Таблица 1.6 – Сравнительная характеристика баз данных

РСУБД/Критерий	**Расширяемость, масштабируемость**	**Производительность**	**Поддержка ACID(уровень изолированности по умолчанию)**	**SQL-совместимость**	**Лицензия**
MySQL		Хотя MySQL относительно быстро справляется с операциями чтения, одновременные операции чтения-записи могут вызвать проблемы.	REPEATABLE_READ	В зависимости от выбора движка MySQL может недоставать некоторых функций.	GNU GPLv2
Microsoft SQL Server	РСУБД очень хорошо взаимодействует с другими продуктами Microsoft.	Движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов.	READ_COMMITTED	Поддерживает полный функционал SQL.	Microsoft EULA
PostgreSQL	PostgreSQL можно программно расширить за счёт хранимых процедур.	В простых операциях чтения PostgreSQL может уступать другим РСУБД, есть возможности оптимизации запросов.	READ_COMMITTED	Полная SQL-совместимость	PostgreSQL

Продолжение таблицы 1.6

РСУБД/Критерий	**Расширяемость, масштабируемость**	**Производительность**	**Поддержка ACID(уровень изолированности по умолчанию)**	**SQL-совместимость**	**Лицензия**
Oracle	Возможна горизонтальная и вертикальная масштабируемость.	есть возможности оптимизации запросов.	READ_COMMITTED	Полная SQL совместимость.	Коммерческая

В ходе выбора базы данных сначала был отклонён вариант Microsoft SQL Server, так как хоть данная РСУБД и поддерживается ОС Ubuntu, но одно из преимуществ продуктов от Microsoft – совместимость друг с другом – пропадает. А также проблемы, связанные с поддержкой данной РСУБД на операционных системах Linux, тяжелее решить, нежели для ОС Windows, так как сообщество Microsoft SQL Server в большинстве случаев работает на продуктах от Microsoft.

Далее необходимо отказаться от Oracle, так как продукт исключительно коммерческий, а его оплата нерезонно большая для данного проекта. Oracle предлагает бесплатную версию своего продукта (XE, Express Edition), которую можно использовать свободно и совершенно бесплатно. Разумеется, Oracle наложила ограничения на свой бесплатный продукт, поэтому версия XE, может работать только с 1 ядром, 1 GB оперативной памяти и хранить данные не более 11 GB. Данные ограничения заставляют задуматься либо о продуктах с менее

жесткими ограничениями, либо обратиться к open-source(свободному программному обеспечению) продуктам.

При выборе между MySQL и PostgreSQL следует обратиться к их лицензиям, так как особо сильной разницы в производительности и масштабируемости в особенности в рамках данного проекта – нет. MySQL имеет двойную лицензию. Данный продукт можно использовать абсолютно бесплатно, но и сам распространяемый продукт должен иметь лицензию GPL, то есть открывать исходные файлы программы. Либо приобретать коммерческую версию без данного ограничения. Лицензия Postgres позволяет бесплатно использовать данную РСУБД без дальнейших обязанностей по лицензированию продукта. Плюс ко всему PostgreSQL является более надежной с точки зрения ACID и поддерживает хранимые процедуры для масштабирования.

1.2.3. Серверная часть приложения

1.2.3.1. Выбор языка программирования

При выборе языка программирования сначала создадим пул языков наиболее популярных среди пользователей крупнейшего веб-сервисы для хостинга IT-проектов и их совместной разработки. [19]

Список наиболее популярных серверных языков на 2019 год сервиса GitHub:

1)Python

Преимущества:

- относительно прост в изучении;
- более высокий уровень абстракции позволяет писать многие операции за меньшее число строк кода нежели языки среднего уровня;
- присутствует функциональное программирование и ООП;

- в стандартных библиотеках Python есть средства для работы с электронной почтой, протоколами Интернета, FTP, HTTP, базами данных, и пр.;
- скрипты, написанные при помощи Python выполняются на большинстве современных ОС. Такая переносимость обеспечивает Python применение в самых различных областях;
- python подходит для любых решений в области программирования, будь то офисные программы, web-приложения, GUI-приложения и т. д.

Недостатки:

- динамическая типизация;
- значительно медленнее Си и ASM, как и большинство скриптовых языков. [5, с.138]

2)Java

Преимущества:

- является эталонным ООП языком;
- строго типизированный язык программирования, что гарантирует стабильную работу в большом проекте;
- java наследует подход языка Си, что позволяет человеку знающему последний быстро изучить Java;
- jVM. JVM определяет сразу несколько преимуществ: независимость от платформы, безопасность (так как программа исполняется не на прямую в среде операционной системы, а в виртуальной машине, соответственно все ошибки остается внутри её, а внешние инъекции кода просто невозможны);
- как компилируемый язык обладает сборщиком мусора;

- в стандартной библиотеке Java есть возможность работы с потоками;
- огромное сообщество, которое имеет большую базу знаний по Java и её фреймворкам.

Недостатки:

- низкая производительность. Но данный недостаток относится лишь к небольшим программам, запускаемым очень часто. Так как JVM – это большая абстрактная машина, то соответственно, чтобы ей подняться, требуется время. Но при разработке сервера, который планируется держать большое количество времени с возможностью масштабирования данный недостаток является пренебрежимо малым;
- многословный код.

3)JavaScript

Преимущества:

- javaScript предоставляет большое количество возможностей для решения самых разнообразных задач. Гибкость языка позволяет использовать множество шаблонов программирования применительно к конкретным условиям;
- популярность JavaScript открывает перед программистом немалое количество готовых библиотек, которые позволяют значительно упростить написание кода и нивелировать несовершенства синтаксиса;
- применение во многих областях. Широкие возможности JavaScript дают программистам шанс попробовать себя в качестве разработчика самых разнообразных приложений.

Недостатки:

- необходимость обеспечивать кроссбраузерность. Код должен корректно выполняться во всех, или хотя бы самых популярных, браузерах;
- система наследования в языке вызывает трудности в понимании происходящего. В JavaScript реализовано наследование, основанное на прототипах. Люди, изучавшие другие объектно-ориентированные языки программирования, привыкли к привычному «класс потомок наследует родительский класс». Но в JavaScript такими вещами занимаются непосредственно объекты, а это не укладывается в голове;
- отсутствует стандартная библиотека. JavaScript не предоставляет никаких возможностей для работы с файлами, потоками ввода-вывода и прочими полезными вещами;
- синтаксис в целом затрудняет понимание. Красота и логичность кода – не преимущество JavaScript. [6, с.138]

4)PHP

Преимущества:

- разработка с помощью PHP дает много возможностей. При должном уровне владения, с помощью шаблонизатора можно создавать не только сценарии для веб-приложений, но и полноценные программы;
- кроссплатформенность. PHP может быть запущен в любой операционной системе, включая юниксоиды;
- поддержка почти всех веб-серверов;
- невысокие расходы на разработку.

Недостатки:

- узкопрофильность. Если вы выучили разработку с помощью PHP, то у вас одна дорога — в веб. И хотя возможности расширены различными реализациями, все же он «заточен» под программирование для Интернета;
- безопасность. У PHP есть средства безопасности уровня системы и уровня web-приложения. Но, опять же, широкая используемость сыграла злую шутку: дыры в PHP находят быстрее, чем разработчики успевают их закрывать. В PHP 7 множество проблем решено, но злоумышленник всегда впереди. В силу того, что массы знают «препроцессор», трудно предугадать всё;
- противоречия в коде. Когда шаблонизатор был только создан, все программное обеспечение разрабатывались с помощью C. Потому в языке было применено множество синтаксиса из него. В то же время, современная аудитория больше сконцентрирована на Java. В итоге, код переполнен различными остатками из разных языков. И все они могут даже быть сконцентрированы в одном выражении кода.

5)C#

Преимущества:

- поддержка подавляющего большинства продуктов Microsoft;
- для небольших компаний и некоторых индивидуальных разработчиков бесплатными являются такие инструменты, Visual Studio, облако Azure, Windows Server, Parallels Desktop для Mac Pro и многие другие;

- большое количество синтаксического сахара, представляющего собой специальные конструкции, разработанные для понимания и написания кода. Они не имеют значения при компиляции;
- порог вхождения у языка C# низкий. Его синтаксис имеет много схожего с другими языками программирования, благодаря чему облегчается переход для программистов. Язык C# считается наиболее понятным и подходящим для новичков;
- после покупки Xamarin на C# можно писать программы и приложения для таких операционных систем, как iOS, Android, MacOS и Linux.

Недостатки:

- приоритетная ориентированность на Windows платформу;
- язык бесплатен только для небольших фирм, индивидуальных программистов, стартапов и учащихся . Крупной компании покупка лицензионной версии этого языка обойдется в круглую сумму;
- в языке осталась возможность использования оператора безусловного перехода. [7, с.138]

C# не был выбран из-за ориентированности на Windows платформу, а платформа .NET ещё развивается в области Web.

JavaScript, PHP и Python не являются строго-типизированными языками. При разработки большого алгоритма данного проекта, возможны Runtime Exception из-за приведения типов и тому подобное. Но все данные языки в купе с фреймворками дают относительно быстрый результат. Но незнание python потребует время на его изучение, так как он не Си подобный язык. Javascript часто используется в роли бекэнд языка, но в роли инструмента для описания сложной алгоритмической логики данный язык будет не лучшим вариантом.

Java – строго-типизированный ООП язык. PHP с 5 версии также стал ООП направленным. При выборе между данными языками следует обратиться к критериям скорости разработки, возможность работы в веб, безопасность, масштабируемость. Скорость работы на PHP при работе с проектами простой логики, например интернет-магазин или новостной блог, естественно будет ощутимой по сравнению с Java. Но при увеличении количества строк в проекте собственного кода PHP код становится более тяжело читаемым, в то время как Java в силу многолетней ООП платформы более удобна в средних и больших проектах. И PHP, и Java имеют инструменты для работы в вебе. Но безопасность и масштабируемость из-за строго-типизированности и работы программ под слоями абстракций JMV значительно лучше у языка Java.

Немаловажным фактом будет являться то, что разработчик проекта является Java-программистом с опытом работы. В этом случае один из основных недостатков Java можно считать решенным.

Для работы был выбран язык программирования Java.

1.2.3.2. Выбор фреймворка

В сфере веб-разработки языка Java предлагается несколько вариантов инструментов.

1) Java Spring Framework

Преимущества:

- огромное семейство фреймворков для создание информационной системы;
- представляет IoC контейнер для работы с зависимостями объектов;
- поставляет MVC фреймворк для работы с вебом.

Недостатки:

- очень тяжелый инструмент в изучении для новичка;
- слишком нагружен ненужными функциями для малого проекта;
- тяжело настраиваемый.

2) Java Spring Boot

Преимущества:

- фактически представляет собой Java Spring Framework с автоконфигурацией, что ускоряет разработку и уменьшает количество ошибок у новичков.

Недостатки:

- ограничивается тонкая настройка зависимостей и инициализации взамен простоте.

3) Java Spark

Преимущества:

- микрофреймворк для создания API для веб-приложения;
- не тянет за собой множество ненужных зависимостей для веба.

Недостатки:

- создание многих вещей отдается самому пользователю, что увеличивает время разработки.

4) JavaEE

Преимущества:

- представляет собой набор библиотек, следующих спецификациям JSR;
- не имеет под собой определенного фреймворка, основа – Java Servlet из стандартной библиотеки Java;
- самая детальная разработка под веб и сервера.

Недостатки:

- низкая скорость разработки;
- малое количество доступной и понятной литературы;
- сложность для новичков.

Данный проект не требует детальной работы под веб и настройку окружения операционных систем и виртуальных машин, а также нет необходимости многие вещи писать с нуля. Таким образом, следует отказаться от JavaEE и Java Spark.

Java Spring Framework зачастую используется для создания корпоративных систем. В этих случаях требуется детальная инициализация всех компонентов и структур. В остальных случаях есть возможность использовать версию для старт-апов и быстрых проектов – Java Spring Boot, позволяющий автоматически генерировать связи между компонентами и инициализацию системы.

1.2.4. Выбор фреймворка для frontend

Для отображения всех результатов работы будет использоваться веб-интерфейс. Раньше требовалось использовать чистые языки JavaScript и языки разметки HTML/CSS для создания веб-интерфейса. Сейчас на рынке фреймворков фронтэнда существует множество продуктов для создания веб-приложений под разные задачи. Практически все они завязаны под язык JavaScript. Перечислим их и их особенности и недостатки каждого в рамках данного проекта.

1) Vue.js

Vue.js (также называемый Vue) представляет собой среду JavaScript с открытым исходным кодом, предназначенную для упрощения и упорядочения разработки пользовательского интерфейса. Фреймворк, также называемый идеальной смесью Angular и React, зарекомендовал себя как идеальный выбор для

разработки легкого приложения с двухсторонней привязкой данных к структуре углов и серверному рендерингу React JS framework. Необходимо отметить, что Vue.js был номинирован как самый популярный JavaScript интерфейс на GitHub с 118 тысячами звёзд в прошлом году. Vue.js может показаться идеальной JavaScript структурой для разработки программного обеспечения (ПО), однако у нее также есть свои плюсы и минусы.

Преимущества

- быстрый набор популярности: всего за несколько лет с момента своего создания большое количество предприятий добавили Vue.js в свой технический стек;
- быстрая настройка: Vue имеет встроенную привязку данных и модель MVC (модель, вид, контроллер), что значительно упрощает настройку по сравнению с Angular.js и React.js;
- более простая интеграция: платформа поддерживает более легкую интеграцию с элементами HTML;
- маленькая кривая обучения: по сравнению с Angular JS Framework, Vue намного легче изучать, понимать и использовать.

Недостатки

- мало ресурсов: структура по-прежнему слишком молода для поиска полезных решений в Интернете и самообучения.
- маленькое вовлечение сообщества. Как уже упоминалось в последнем пункте, Vue.js является новичком на рынке и, следовательно, имеет меньшую поддержку сообщества по сравнению с Angular и React технологиями.

2) React

Поддерживаемый Facebook, Instagram и другими известными организациями, React является одним из лучших JavaScript фреймворков на протяжении последних 5 лет. Также называемая React.js или React JS, инфраструктуру frontend использует более чем 38% разработчиков по всему миру. Netflix, Flipboard, PayPal и BBC являются первыми организациями, которые использовали React.

Преимущества

- множество документации и онлайн-ресурсов. Благодаря поддержке Facebook существует множество возможностей использовать тонну документации и онлайн-ресурсов для обучения и использования фреймворка Javascript для React;
- быстрая, гибкая, эффективная и лёгкая технология: система JS широко рекомендуется благодаря своей эффективности, небольшому размеру блоков, гибкости и более быстрому подходу к работе благодаря простой компонентной модели и функциональности рендеринга на стороне сервера;
- переход между версиями. Миграция между версиями, как правило, очень проста, Facebook предоставляет “codemods” для автоматизации огромной части процессов;
- отлично себя чувствует при работе с ES6/7 ReactJS, может брать на себя любую нагрузку;
- структура имеет компонентную архитектуру, которая революционизировала веб-разработку приложений и повлияла на другие технологии;
- DOM (Document Object Model – «объектная модель документа») позволяет объединять HTML, XHTML или XML документы по определенным

критериям, чаще всего в дерево, поэтому React отлично подходит для веб-браузеров при анализе разнообразных элементов веб-приложений.

Недостатки

- необходимы средства сборки: данная инфраструктура JavaScript может работать некорректно без адекватных инструментов сборки или может отображать несовместимость с другими библиотеками и кодами из-за высокой DOM;
- большая кривая обучения: в отличие от Vue, React требует больше времени для изучения концепций и реализации. React JS требует огромное количество знаний в том как интегрировать пользовательский интерфейс в структуру MVC;
- отсутствие упорядоченной документации - сверхбыстрый обмен решениями в ReactJS не оставляет места для упорядочения документации, документы размещены немного хаотично, поскольку многие разработчики индивидуально вносят их в базу данных без какого-либо систематического подхода.

3) Angular.js

Angular.js – это полнофункциональная интерфейсная JavaScript среда, поддерживаемая Google и другими популярными корпорациями. Эта структура известна своими потенциальными возможностями, например, быстрым созданием кода, двусторонней привязкой данных, тестированием частей приложения. Рассмотрим плюсы и минусы этой технологии:

Преимущества:

- двусторонняя привязка данных;
- мобильный подход к веб-разработке;

- поддержка PWA (Progressive Web App – «прогрессивное веб-приложение»);
- стабильная и долгосрочная поддержка Google;
- универсальный MVVM(Model-View-ViewModel) модуль(есть возможность использовать одни и те же данные в одной части приложения);
- взаимозависимость функций, в виду их связанности с компонентами и модулями;
- RXJS, молниеносная компиляция (менее 2,9 секунд), изменённый пуск HttpClient.

Недостатки:

- плохая оптимизация: приложения на основе Angular.js требуют большей оптимизации для решения проблем с низкой производительностью;
- большая кривая обучения: Angular показывает высокую шкалу обучения, вам нужно больше времени, чтобы освоить эту структуру;
- интеграционные ошибки, которые могут возникать при переходе от старой версии к новой;
- сложный язык программирования, несмотря на то, что Angular использует TypeScript 2.4. [8, с.138]

Для реализации проекта требуется достаточен фреймворк с достаточно низким порогом вхождения, так как работа не требует новшеств в интерфейсе пользователя. Таковым является Vue.js, а также у данной технологии большая документация на русском языке.

1.3. ВЫВОД

Поставленная задача является автономной и решается реализацией в виде веб-приложения. Для её решения используется готовые инструменты как программного обеспечения – базы данных, сервер и веб-фреймворки, так и математического аппарата.

Предметная область задач, проектов, команд и их составляющих должна быть описана в табличном виде и обработана алгоритмически в соответствии с поставленными задачами в на серверной части приложения, а отображения результатов выводится в браузере веб-интерфейсом.

2. Определение требований

2.1. Функциональные требования

2.1.1. Основные требования к функционалу системы

Система выполняет роль экосистемы для проектов, задач и команд. Необходимо реализовать следующие основные функции:

– управление задачами

а) добавление задач;

б) редактирование задач;

в) прикрепление аудио-, видео- материалов, а также различных текстовых файлов к задачам;

г) установление временных штампов для задачи;

д) напоминание пользователю о невыполненных задачах и задачах, близких к концу сроку выполнения (срок выдачи оповещения должен задаваться в настройках задачи).

– управление проектами

а) создание проектов;

б) редактирование проектов;

в) управление межзадачными процессами в рамках проекта, создание зависимостей между задачами;

г) создание миграций задач;

д) установление пользователей/групп, работающих над задачей;

е) настройка ролей и обязанностей пользователей в отношении задач определенного проекта;

ж) система поощрений за выполнение задачи.

– управление командами

а) создание пользовательских ролей с определенными правами;

б) создание групп в рамках определенного проекта;

в) возможность делегирования прав одних ролей пользователей другим ролям – создание иерархии команды;

г) внутрикомандная коммуникация;

д) обеспечить возможность указания наличия или отсутствия вакансии для проекта или команды.

– межкомандное взаимодействие

а) слияние команд и, соответственно, проектов;

б) возможность скрыть свою команду в сообществе и/или скрыть проект(ы) от сообщества, сделать приватной команду или проект(ы);

в) миграция проектов от одной команды к другой;

г) слияние проектов нескольких команд;

д) система поиска/объявлений о проекте между командами;

е) система коммуникаций с целью поиска проектов и поиска участников проекта.

Система сообщества должна включать в себя следующие функциональные возможности:

– взаимодействие с группами/командами

а) поиск групп или команд;

б) просмотр, если возможно, информации о группе;

в) система создания заявки на прием в группу;

г) система принятия заявок о принятии в группу;

д) создание кооперации работ для групп по совместным проектам (общие задачи или список задач, делегирование задач и их обобщение).

– система взаимодействия с проектами

а) поиск проектов;

б) подача заявки на прием в определенный проект;

в) подача заявки на миграцию собственных проектов, списка задач или задачи на другой проект;

г) просмотр, если возможно, проектов;

д) добавление в «Избранное» данного пользователя определенного проекта.

– система работа с другими пользователями

а) просмотр, если возможно, информации о пользователе;

б) подача заявки в «Знакомые» любому пользователю, если возможно;

в) просмотр уровня работы, завершенности проектов, рекомендаций от организаций и команд, на которые есть определенные ссылки;

– система роста пользователя

а) общее количество проектов (собственных или привязанных к определенной группе/организации);

б) уровень пользователя в определенной сфере (законченные проекты, количество добавленных в «Избранное» его проектов);

в) рекомендации от сообщества (от отдельных пользователей, групп или организаций).

2.1.2. Требования к авторизации пользователя

Каждый пользователь, работающий в системе должен:

– зарегистрироваться в системе, указав

а) имя пользователя (необязательно настоящее);

б) e-mail пользователя;

в) пароль для входа в систему;

г) секретное слово при ответе на определенный вопрос.

– верифицировать свою личность через электронную почту, чтобы подтвердить, что он не является бот-спамом.

Каждый пользователь должен иметь доступ к тому материалу (проектам, командам и задачам), к которому он привязан с учетом прав доступа.

2.1.3. Общие требования к системе безопасности

Система должна обладать следующими средствами безопасности:

– пользователь при входе в систему обязан указывать логин и пароль;

– пользователь при регистрации в системе обязать указывать пароль два раза (для подтверждения);

– каждый пользователь должен имеет возможность скрыть определенную информацию о себе или, наоборот, показать;

– пароль шифруется на стороне сервера-обработчика;

– важные конфиденциальные данные передаются на сервер POST запросом.

2.1.4. Требования безопасности контента

После регистрации пользователь должен иметь следующие права:

– просмотр открытых проектов;

– просмотр задач вне проектов;

– просмотр открытых команд;

– создание собственных проектов и(или) групп и(или) задач;

– вывешивание объявлений;

– выдача заявок на присоединение к проекту и(или) к команде;

– выдача заявки на выполнение задачи.

Требования к модерированию контента:

– заявки на присоединение к проекту или к команде или на выполнение задач рассматривает создатель ресурса, его решение должно личным сообщением системой уведомлений;

– создание проектов, групп, задач должно рассматриваться глобальным модератором приложения. При положительном решении материал публикуется, при отрицательном решении мотивированный отказ посылается личным сообщением через систему уведомлений.

2.1.5. Требования к системе уведомлений

Приложение должно обладать следующим минимальным набором уведомлений:

– окончание срока выполнения заданий;

– окончание срока выполнения проекта;

– невыполнение задания;

– какие-либо изменения отношений между задачами и(или) проектами, касающихся пользователей, непосредственно связанных с ними;

– какие-либо изменения самих задач и(или) проектов, касающихся пользователей, непосредственно связанных с ними;

– получение отзыва на объявление администратору/владельцу проекта или команды;

– получение ответа на любой отзыв, непосредственно адресованный определенному пользователю.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

2.2.1. Требования к построению интерфейса системы

Приложение должно графически разделять следующие функциональные структуры:

– главная страница системы

а) основная информация о пользователе;

б) контекстное отображение всех проектов, групп;

в) совокупность уведомлений для данного пользователя;

г) ссылки для быстрого и удобного перемещения к определенным проектам и задачам.

– проекты и задачи, относящиеся к пользователю;

– система сообщества;

– инструментарий для создания проектов, задач и групп/команд;

– настройки основного интерфейса.

Проекты и задачи, относящиеся к пользователю, должны быть логически построены в интерфейсе сервиса.

2.2.2. Требования к графическому оформлению системы

Приложение должно придерживаться на всей карте системы определенного стиля:

– хорошо читаемый печатный шрифт;

– отсутствие ярких кислотных или сильно выделяющихся цветов;

– использовать цветовую гамму светло-синего, бело-серого, темно-красного или бурого цветов.

Система должна адаптироваться к экранам от 200*200px до 3000*2000px.

Учесть особенности оформления приложений на мобильных устройствах в части:

– увеличения в масштабе блоков по сравнению с десктопной версией особенно мелких, но важных с точки зрения функционала частей;

– горизонтального и вертикального адаптирования;

Оформление приложения должно быть организовано в следующем виде:

- верхняя шапка неизменна и содержит название сервиса, наименование разработчика;

– верхнюю часть занимает меню-навигация. Данное меню при перемещении по его пунктам, остается неизменным. Необходимо реализовать синхронное взаимодействие с ним посредством AJAX или иных средств, базирующихся на основе AJAX;

– центральная и нижняя часть экрана отдается под информацию, соответствующему выбранному функционалу.

2.2.3. Требования к графическому оформлению системы задач и проектов

Приложение должно иметь структуру водопада. При нажатии на определенный проект пользователь должен попадать на списки задач. При нажатии на список задач пользователь должен иметь возможность отдельно рассмотреть каждую задачу.

Для каждого уровня возможен просмотр в виде графа/дерева списка задач, либо отдельных задач в зависимости от уровня погружения.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

3.1.1. Основная архитектура

Для реализации веб-приложения для управления проектами, задачами и командами будет использоваться следующая архитектура: создание двух приложений, где первое является REST API сервис, а второе - SPA (Single Page Application) приложение.

REST API определяет набор функций, к которым разработчики могут совершать запросы и получать ответы. Взаимодействие происходит по протоколу HTTP. Преимуществом такого подхода является широкое распространение протокола HTTP, поэтому REST API можно использовать практически из любого языка программирования.

SPA (одностраничное приложение) - это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX.

По сути, это два независимых друг от друга приложения, но если созданный SPA – это оболочка приложения, то созданный REST API – это источник выгружаемых данных, представленный в виде MC(Model-Controller).

Ниже представлен рисунок с архитектурой SPA+REST API.

Рисунок 3.1 – SPA и REST API

3.1.2. SPA

Главным преимуществом SPA приложений считается их универсальность. Они одинаково хорошо функционируют как на персональных компьютерах, так и на мобильных устройствах. Планшеты, смартфоны и даже простые мобильные телефоны, постепенно уходящие в историю, без проблем работают с проектами, разработанными по принципу SPA. Поэтому первым преимуществом одностраничных приложений можно считать широкий охват устройств. Следовательно, создание одностраничного приложения позволяет рассчитывать на значительно большую целевую аудиторию, чем при использовании стандартных методов веб-разработки.

Вторым преимуществом SPA считается насыщенный интерфейс. Этот плюс обусловлен тем, что на одной веб-странице гораздо проще создать богатый интерфейс. Такой подход существенно упрощает процессы хранения информации, а также управления состоянием представления.

Еще одно преимущество одностраничных портал заключается в упрощении процедуры загрузки контента. Если сайт работает с шаблонами, то вместе с

загрузкой любой страницы этого портала пользователь обязательно загружает и разметку шаблона. Конечно, сегодня кэширование уже достигло неимоверно высоких результатов. Но тем не менее, в SPA нечего кэшировать, а это означает, что происходит существенная экономия и времени, и ресурсов.

Фреймворки, реализующие парадигму SPA, работают по следующим принципам:

- SPA осуществляет поддержку пользовательской навигации. Все пользовательские «передвижения» должны фиксироваться в истории навигации. При этом навигация должна быть «глубокой». Другими словами, если пользователь откроет скопированную ссылку на внутренний модуль в другом окне или, например, браузере, он должен попасть на необходимую ему страницу;
- разработка SPA приложения предусматривает использование только одной страницы. Следовательно, все необходимое для функционирования этого приложения (скрипты, стили и т.д.) должно размещаться на единственной веб-странице;
- SPA сохраняет данные о работе пользователей в DOM-хранилище или кэше браузера;
- SPA приложение предусматривает загрузку всех необходимых скриптов в процессе инициализации веб-страницы;
- SPA загружает дополнительные модули «по требованию» пользователя.

Схема, описывающая работу SPA приложений представлена на рисунке 3.2. [9, с.139]

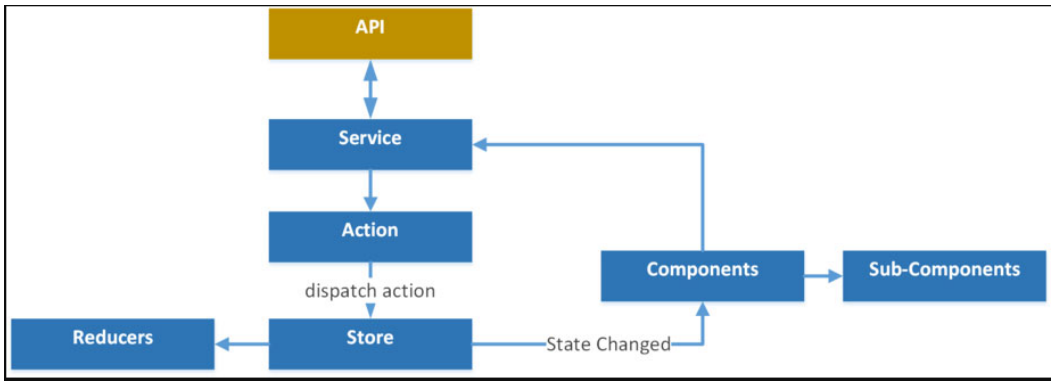

Рисунок 3.2 – SPA

Реализация SPA в веб-приложении будет представлена фреймворком Vue.js.

3.1.3. Сервер API

Серверная часть представлена фреймворком Java Spring Boot. Данная технология уже включает в себя по умолчанию фреймворк Java Spring MVC, имеющий в себе реализацию веб-контейнера для создания веб-приложения.

Исходя из названия фреймворка следует догадаться, что архитектура является MVC (Model-View-Controller), но так как компонент View представлен SPA приложением, описанным ранее, то архитектура сервера API будет включать в себя компонент Model и Controller.

Ниже представлена схема построения такой системы.

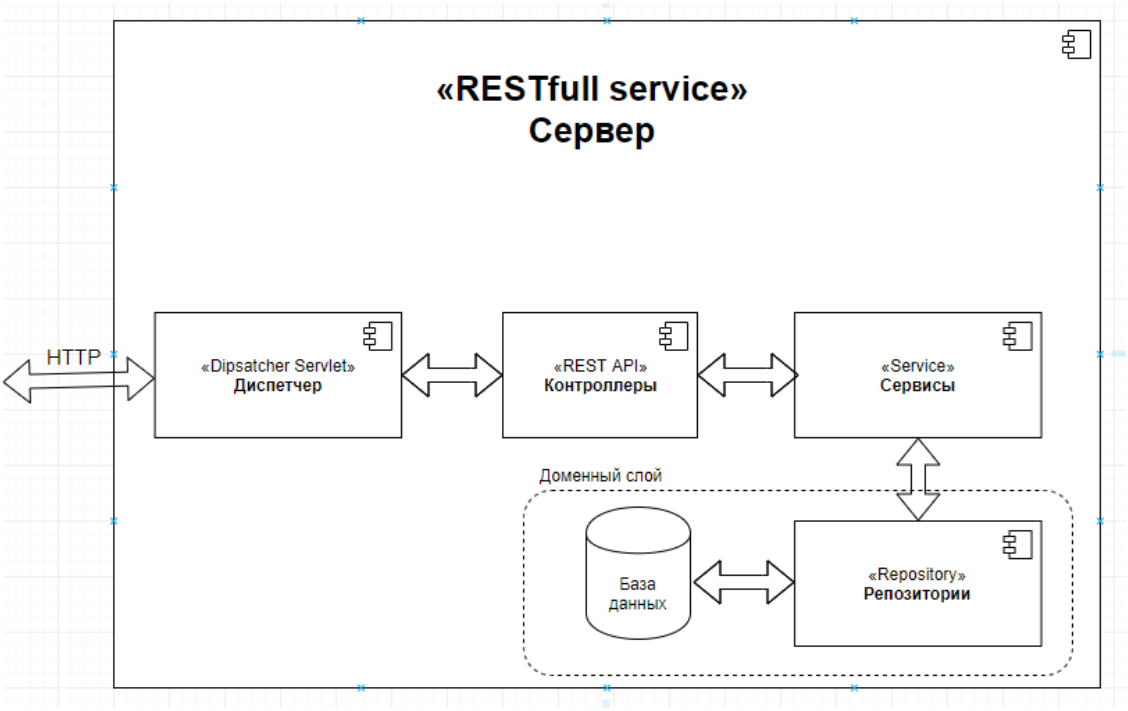

Рисунок 3.3 – Spring MVC RestFull Service

Шаги работы сервиса:

1) Фреймворк Spring MVC получает запрос от клиента и вызывается обработчик REST API.
2) JSON формат десериализуется в объект запроса.
3) Если присутствует реализация валидатора, то данные проверяются.
4) Диспетчер-сервлет передаёт объект запроса определенному контроллеру, которому изначально предназначался запрос.
5) Контроллер делегирует выполнение бизнес-логики определенному сервису.
6) При работе с данными из базы сервис обращается к слою репозиториев, которые непосредственно выполняют обращения к базам данных.

Отправка ответа происходит также, но в обратном порядке.

Так как Spring Boot скрывает многие операции от разработчика, то нет необходимости работать непосредственно с сервлетом-диспетчером, в приложе-

нии будут создаваться контроллеры, к которым будут приходить запросы, сервисы и репозитории.

3.2. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ

3.2.1. Основные принципы организации приложения

Веб-приложение предлагает автоматизацию работы над проектом или группой задач, в том числе и в командной среде. Для построения фундамента системы был изучен материал по управлению проектами, в частности в IT-сфере. Одним из таких источников является книга Джеффа Сазерленда «SCRUM. Революционный метод управления проектами».

Джефф Сазерленд — американский программист, один из разработчиков методологии Scrum и авторов Agile Manifesto. Смысл Agile сформулирован в Agile-манифесте разработки ПО: «Люди и взаимодействие важнее процессов и инструментов. Работающий продукт важнее исчерпывающей документации. Сотрудничество с заказчиком важнее согласования условий контракта. Готовность к изменениям важнее следования первоначальному плану».

К agile философии относят такие фреймворки, как Scrum и Kanban.

Scrum – это «подход структуры». Над каждым проектом работает универсальная команда специалистов, к которой присоединяется еще два человека: владелец продукта и scrum-мастер. Первый соединяет команду с заказчиком и следит за развитием проекта; это не формальный руководитель команды, а скорее куратор. Второй помогает первому организовать бизнес-процесс: проводит общие собрания, решает бытовые проблемы, мотивирует команду и следит за соблюдением scrum-подхода.

Scrum-подход делит рабочий процесс на равные спринты – обычно это периоды от недели до месяца, в зависимости от проекта и команды. Перед спринтом формулируются задачи на данный спринт, в конце – обсуждаются результаты, а команда начинает новый спринт. Спринты очень удобно сравнивать между собой, что позволяет управлять эффективностью работы.

Kanban – это «подход баланса». Его задача – сбалансировать разных специалистов внутри команды и избежать ситуации, когда дизайнеры работают сутками, а разработчики жалуются на отсутствие новых задач.

Вся команда едина – в kanban нет ролей владельца продукта и scrum-мастера. Бизнес-процесс делится не на универсальные спринты, а на стадии выполнения конкретных задач: «Планируется», «Разрабатывается», «Тестируется», «Завершено» и др.

Главный показатель эффективности в kanban – это среднее время прохождения задачи по доске. Задача прошла быстро – команда работала продуктивно и слаженно. Задача затянулась – надо думать, на каком этапе и почему возникли задержки и чью работу надо оптимизировать.

Для визуализации agile-подходов используют доски: физические и электронные. Они позволяют сделать рабочий процесс открытым и понятным для всех специалистов, что важно, когда у команды нет одного формального руководителя. [10, с.139]

Данные подходы не являются «железными». Множество команд на реальной работе комбинируют данные и иные agile-подходы, создавая новую систему коммуникаций.

В данной задаче фундамент будет придерживаться принципов и Kanban, и Scrum:

- существует доска и задачи в ней;
- у задачи есть свои статусы выполнения;

- команда едина, нет отдельных ролей и специальностей, люди считаются универсалами с точки зрения подхода;
- использование временных штампов для задач и списков;
- у задач есть свой уполномоченный;
- задачи имеют свойство зависимости друг от друга;
- команды могут кооперировать и либо сливать, либо разъединять задачи.

3.2.2. Математический аппарат

3.2.2.1. Помощь пользователю в распределении работников

Сущность доски (имеется в виду проект в рамках работы) представляется в виде графа. Соответственно, список задач – это часть графа, а задача – это атомарный узел.

В соответствии с требованиями из п. 2.1 в веб-приложении должна быть возможность просмотра оптимального выполнения задач в графе. Условимся, что список задач – это всегда связанный граф, то есть каждая задача каким-либо образом связана с другими. Списки задач – необязательно связанные структуры.

Таким образом, оптимальный обход в рамках данного решения является равномерным выполнением задач по мере их поступления определенному числу лиц. Данное решение представлено графически на рисунке 3.4.

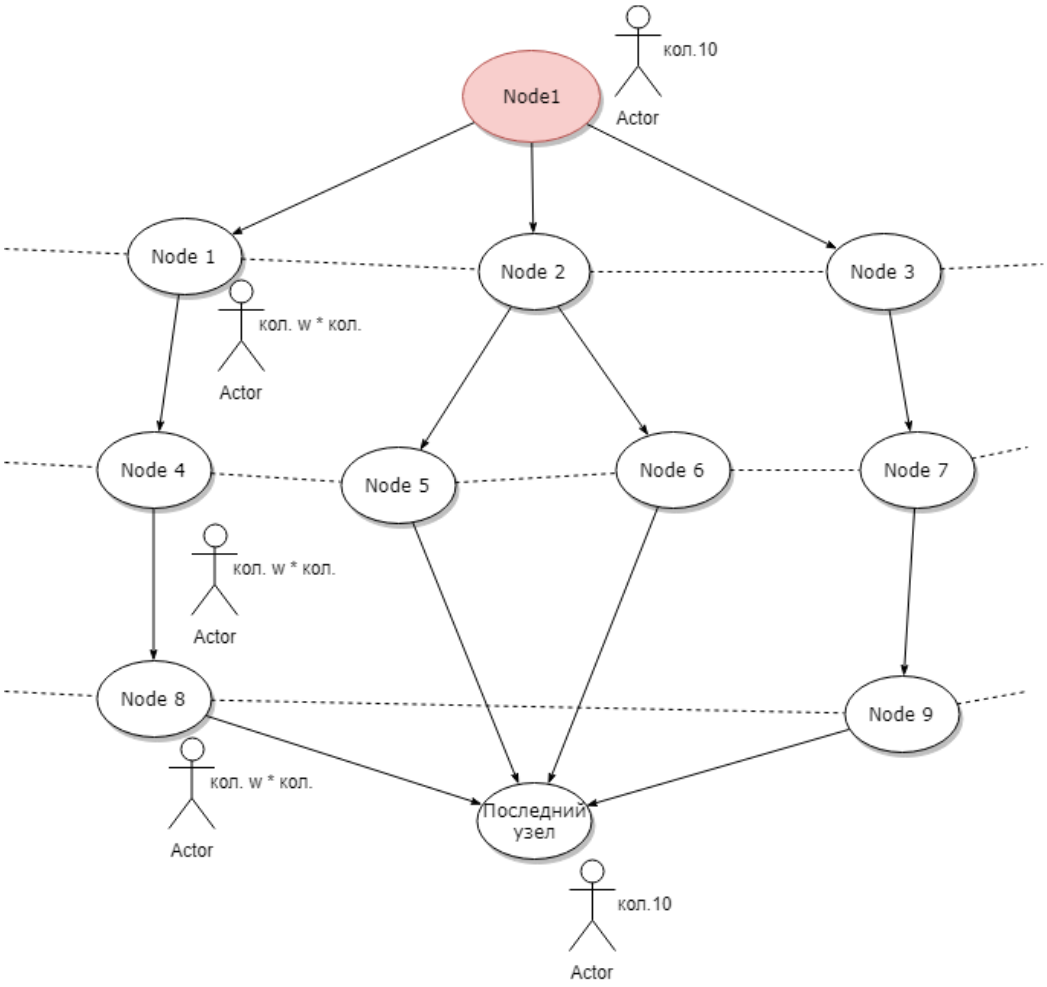

Рисунок 3.4 – Выполнение группой людей связанного списка задач

В качестве примера был выбран список задач, на который назначены 10 человек. После выполнения самой первой корневой задачи всеми людьми в количестве 10 данная группа равномерно распределяется по задачам в прямой пропорциональности от сложности следующей задачи. Количество людей на данном узле вычисляется согласно формуле (1)

$$n = W * N, \quad (1)$$

где n – текущее количество людей на узле,

W – вес узла,

N – общее число людей.

Вес определяется как отношение сложности текущего задания (узла) к общей сложности всех заданий на данном этапе прохождения графа согласно формуле (2)

$$W = –, \qquad (2)$$

где c – сложность данного узла,

C – сумма сложности всех узлов данного этапа;

W – вес задания.

Для прохождения графа данным образом необходимо использовать алгоритм поиска в ширину графа. При обходе в ширину, узлы графа, равноудаленные от начальной вершины, обрабатываются одним шагом алгоритма. За счет этого, алгоритм находит путь, который содержит наименьшее количество узлов.

Для работы такого алгоритма необходимо поддерживать очередь необработанных вершин (именно очередь, т.к. выбор вместо нее стека превратит наш алгоритм в поиск в глубину).

Алгоритм работает следующим образом: сначала обрабатываются все вершины, смежные с текущей, а лишь потом – их потомки. Вместо стека для запоминания еще не обработанных вершин используется очередь (первый пришел – первый вышел). Последовательность действий:

1. Задать стартовую вершину (аналог корневой вершины при обходе дерева).
2. Обработать стартовую вершину и включить ее во вспомогательный список обработанных вершин.
3. Включить в очередь все вершины, смежные со стартовой
4. Организовать цикл по условию опустошения очереди и внутри цикла выполнить:
   а) извлечь из очереди очередную вершину;

b) проверить по вспомогательному списку обработанность этой вершины;

c) если вершина уже обработана, то извлечь из очереди следующую вершину;

d) если вершина еще не обработана, то обработать ее и поместить в список обработанных вершин;

e) просмотреть весь список смежных с нею вершин и поместить в очередь все еще не обработанные вершины.

Реализация в сервисном слое исходного кода приложения данного алгоритма на языке Java приведена на листинге 3.1.

Листинг 3.1 – Реализация поиска в ширину в графе на языке Java

```
public void BFS(Task headerTask, int userNum){
        int stage = 0,currentStage = 0;
        ArrayDeque<Task> taskQueue = new ArrayDeque<>();
        List<Task> stageBuffer = new ArrayList<>();
        List<Task> allTasks = getAllTaskOf-
List(headerTask.getTaskList().getId());
        headerTask.setStage(stage);
        taskQueue.addLast(headerTask);

        while (!taskQueue.isEmpty()) {
            Task currentTask = taskQueue.pop();
            stageBuffer.add(currentTask); //???
            if (currentTask.getStage() == currentStage){
                currentStage = currentStage + 1;
            }


            for (Task childTask:
                 currentTask.getTaskRelation()) {
                if (checkDouble(childTask,taskQueue) || isRelatedWithA-
nother(childTask,currentTask,allTasks)) continue;
                childTask.setStage(currentStage);
                taskQueue.addLast(childTask);
            }

            if(!taskQueue.isEmpty() && taskQueue.getFirst().getStage() >
currentTask.getStage() || taskQueue.isEmpty()){
                processNodes(stageBuffer,userNum);
```

```
                stageBuffer.clear();
            }
        }
    }
```

В этой же функции параллельно вычисляется необходимое количество людей на один узел.

3.2.2.2. Надежность выполнения задачи

В проекте при создании задачи пользователем сервис сможет дать оценку реальности выполнения его на основе показателя сложности, выданного самим пользователем ранее, и на основе эффективности выполнения задач в данном списке задач пользователями, которые также были выбраны данным пользователем в качестве работников будущей задачи.

Показатель надежности выполнения задачи в проекте – это отношение показателя эффективности выполнения задач в данном списке всеми пользователями, которые поставлены на новое задание, к показателю эффективности выполнения новой задачи к последнему дню, выданному на выполнение.

Показатель эффективности выполнения одной задачи одним пользователем - это отношение сложности задачи к количеству дней, которое потребовалось на выполнение его, умноженное на коэффициент ценности пользователя. Коэффициент ценности прямо пропорционален количеству успешно выполненных задач в срок данным пользователем. При выполнении очередного задания, его коэффициент закономерно увеличится. Следует заметить, что подразумевается именно реальное число дней выполнения, а не заданное при создании задачи максимальный срок. Это отношение можно описать формулой (3)

$$E = \quad -, \tag{3}$$

где E – эффективность выполнения задачи;

C – сложность задачи;

n – количество дней на выполнение;

v – коэффициент ценности пользователя.

Пользователь на момент создания новой задачи мог выполнить уже множество задач в данном списке. Таким образом, необходимо взять среднее арифметическое по всем данным заданиям по формуле (4)

$$\text{польз.} = \quad — \tag{4}$$

где k – количество задач, выполненных пользователем;

– эффективность выполнения задачи;

польз. – среднее арифметическое по всем эффективностям задач пользователя.

А также на выполнение новой задачи могут быть выбраны несколько человек. Тогда следует взять среднее арифметическое по всем средним арифметическим каждого из пользователей согласно формуле (5)

$$\text{полн.} = \quad ———, \tag{5}$$

где m – число работников новой задачи;

– средняя эффективность выполнения задач определенного пользователя;

полн. – итоговая средняя эффективность выполнения задач по всем пользователям.

Чтобы оценить реальность выполнения к последнему дню, заданным пользователем, необходимо найти отношение среднего арифметического по всем будущим работникам к эффективности выполнения новой задачи к крайнему дню согласно формуле (6)

$$= \frac{———}{——} \tag{6}$$

где – сложность новой задачи;

– количество дней с нынешнего дня по указанный крайний срок выполнения;

– итоговая средняя эффективность, рассчитанная по формуле (5);

– оценка реальность выполнения новой задачи.

3.3. ОПИСАНИЕ ДАННЫХ

Схема базы данных с использованием UML-диаграмм представлена на рисунке 3.5.

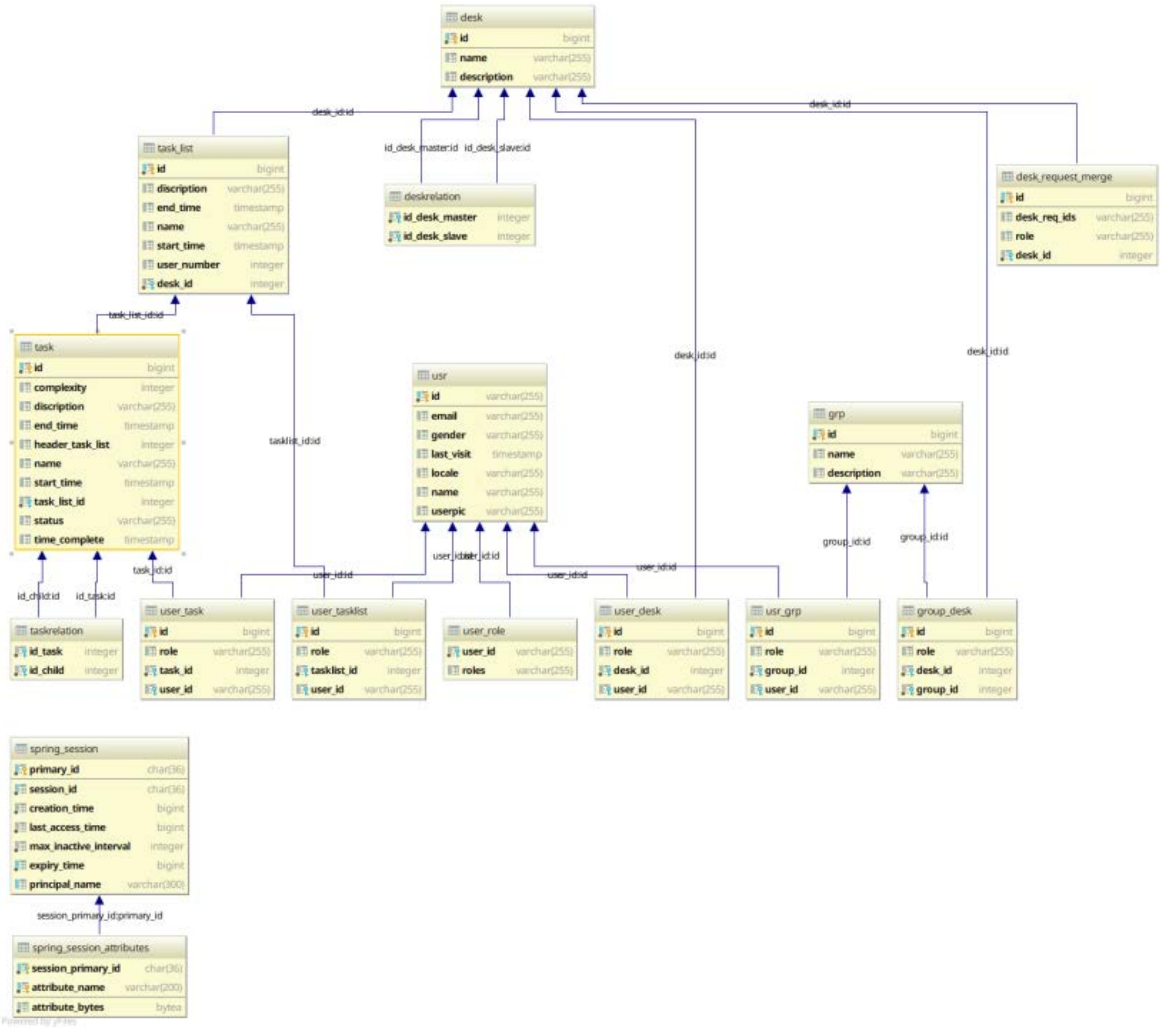

Рисунок 3.5 – Общая схема базы данных

Следующие рисунки 3.6 – 3.9 отражают ту же схему, но в увеличенном масштабе.

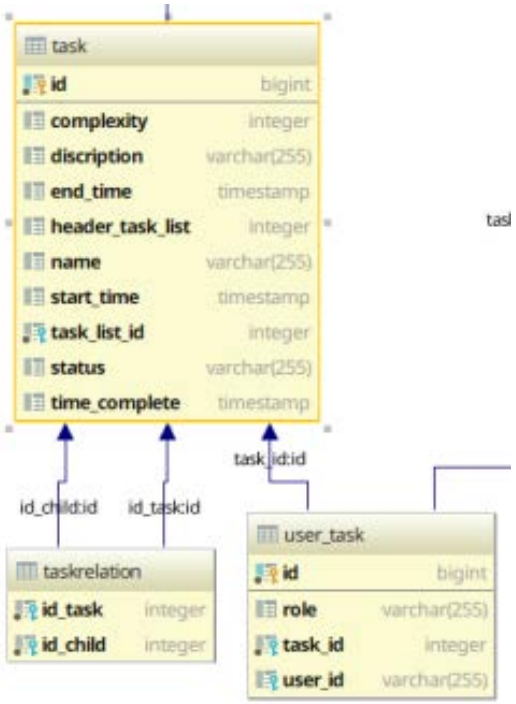

Рисунок 3.6 – Фрагмент схемы базы данных

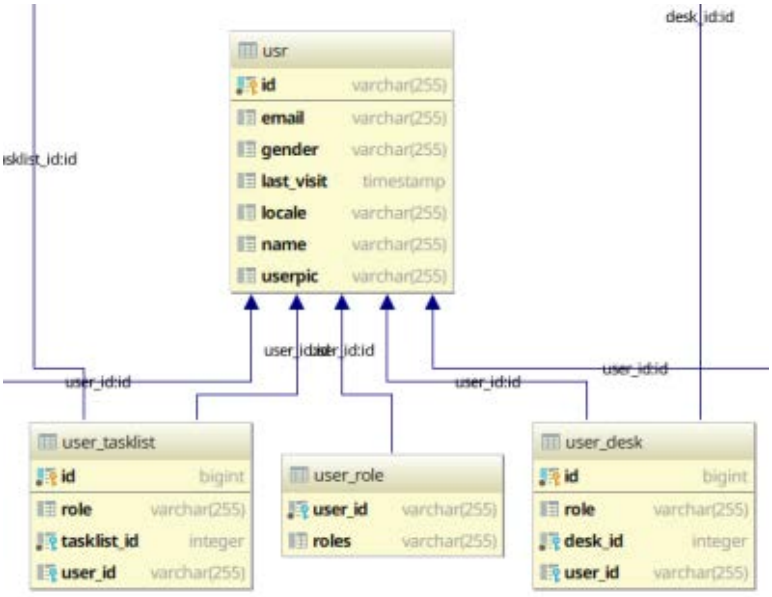

Рисунок 3.7 – Фрагмент схемы базы данных

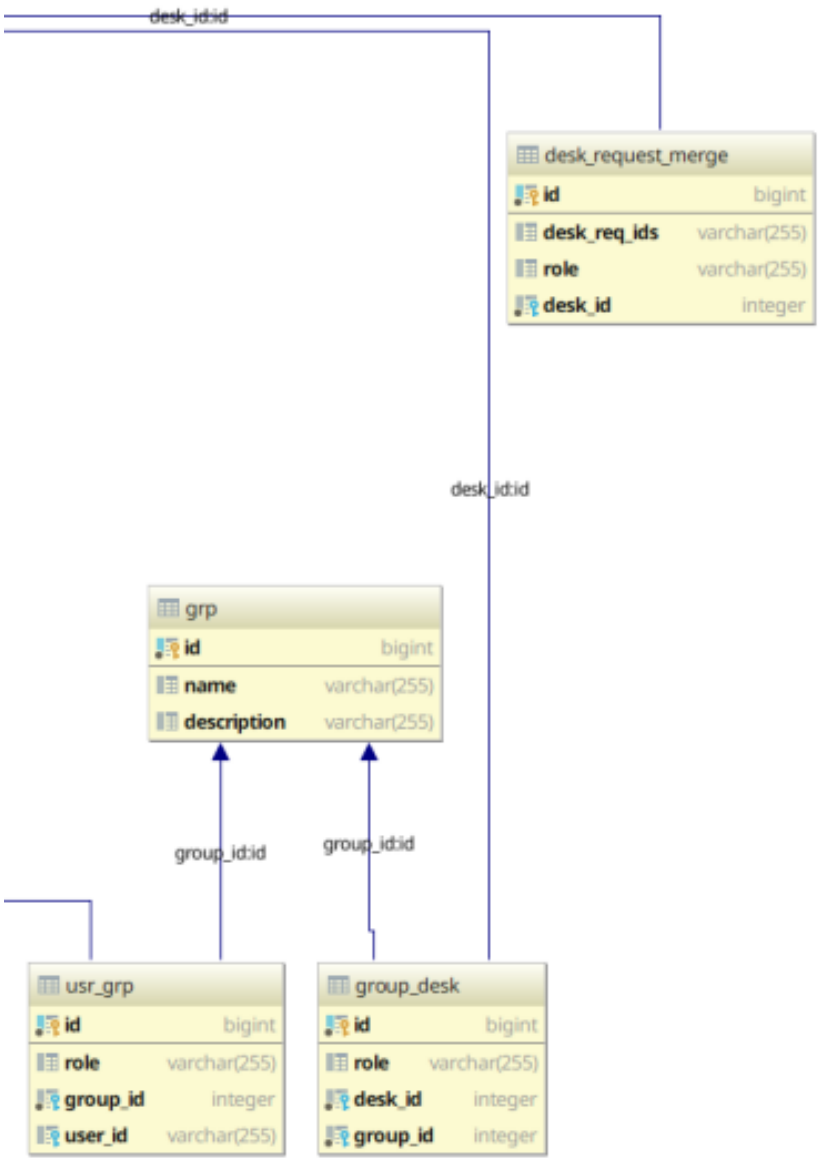

Рисунок 3.8 – Фрагмент схемы базы данных

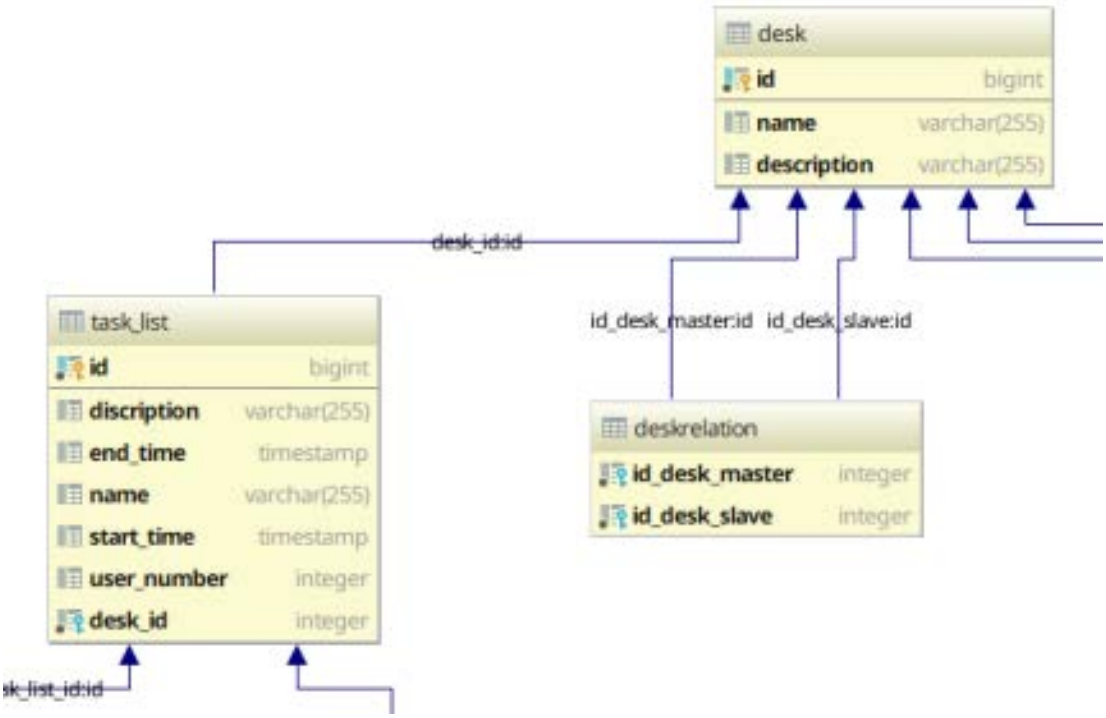

Рисунок 3.9 – Фрагмент схемы базы данных

Таблицы:

1) desk

Данная таблица представляет сущность доски проекта.

Таблица 3.1 – Таблица desk

Поле	**Тип**	**Пояснение**
Id	bigint	Идентификатор доски
name	varchar	Наименование доски

2)grp

Данная таблица представляет сущность группы пользователей.

Таблица 3.2 – Таблица grp

Поле	**Тип**	**Пояснение**
Id	bigint	Идентификатор группы

Продолжение таблицы 3.2

Поле	Тип	Пояснение
name	varchar	Наименование группы

3)task

Данная таблица представляет сущность задания.

Таблица 3.3 – Таблица task

Поле	Тип	Пояснение
Id	bigint	Идентификатор задания
complexity	integer	Безразмерная сложность задания
description	varchar	Описание задания
Start_time	timestamp	Начало выполнения задания
End_time	timestamp	Окончание выполнения задания
Header_task_list	integer	Флаг, обозначающий что задание является первым в списке
name	varchar	Наименование задания
Task_list_id	integer	Внешний ключ идентификатора списка заданий

4)task_list

Данная таблица представляет сущность списка заданий.

Таблица 3.4 – Таблица task_list

Поле	Тип	Пояснение
Id	bigint	Идентификатор списка заданий
description	varchar	Описание списка задания
Start_time	timestamp	Начало выполнения списка заданий
End_time	timestamp	Окончание выполнения спика заданий
User_number	integer	Количество пользователей в данном списке
Desk_id	integer	Внешний ключ идентификатора доски

5)taskrelation

Данная таблица представляет связь многие ко многим заданий друг к другу. Первое поле указывает на идентификатор данного задания, а второй – на его дочернее задание.

Таблица 3.5 – Таблица taskrelation

Поле	Тип	Пояснение
Id_task	integer	Идентификатор задания
Id_child	integer	Идентификатор дочернего задания

6)user_desk

Данная таблица представляет связь между таблицами usr и desk.

Таблица 3.6 – Таблица user_desk

Поле	Тип	Пояснение
id	bigint	Идентификатор связи
User_id	varchar	Идентификатор пользователя
Desk_id	integer	Идентификатор доски
role	varchar	Роль данного пользователя в данной доске

7)user_role

Данная таблица представляет сопоставление каждого пользователя в системе и его роль (полномочия) глобально во всем приложении.

Таблица 3.7 – Таблица user_role

Поле	Тип	Пояснение
User_id	varchar	Идентификатор пользователя
roles	varchar	Роли пользователя в системе

8)user_task

Данная таблица представляет связь между таблицами usr и task.

Таблица 3.8 – Таблица user_task

Поле	Тип	Пояснение
id	bigint	Идентификатор связи
User_id	varchar	Идентификатор пользователя
Task_id	integer	Идентификатор задания
role	varchar	Роль данного пользователя в данном задании

9)user_tasklist

Данная таблица представляет связь между таблицами usr и task_list.

Таблица 3.9 – Таблица user_tasklist

Поле	Тип	Пояснение
id	bigint	Идентификатор связи
User_id	varchar	Идентификатор пользователя
Task_id	integer	Идентификатор списка заданий
role	varchar	Роль данного пользователя в данном списке заданий

10)usr

Данная таблица представляет сущность пользователя в системе.

Таблица 3.10 – Таблица usr

Поле	Тип	Пояснение
id	varchar	Идентификатор пользователя
email	varchar	Электронная почта пользователя
Gender	varchar	Пол пользователя
Last_visit	Timestamp	Последний вход пользователя в систему
locale	varchar	Язык системы данного пользователя
userpic	varchar	Хранит ссылку на пользовательское изображение своей записи

11)user_grp

Данная таблица 3.11 представляет связь между таблицами usr и grp.

Таблица 3.11 – Таблица user_grp

Поле	Тип	Пояснение
id	bigint	Идентификатор связи
User_id	varchar	Идентификатор пользователя
Group_id	integer	Идентификатор группы
role	varchar	Роль данного пользователя в данной группе

3.4. Описание классов сущностей

В папке domain бэкэнд приложения располагаются все классы сущностей необходимых для работы. Данные классы соотносятся с определенными таблицами из базы данных. Для указания, что данный класс описывает структуру предметной области при его описании его класса, необходимо добавить аннотацию @Entity. Исходный код основных классов сущностей и перечисления ролей приведен в листингах А.1. – П.1. приложений А – П.

В таблице 3.12 приведено описание пользователей и групп.

Таблица 3.12 – Классы описания пользователей и групп.

Наименование	**Назначение**	**Поля**
User	Описание сущности пользователя	Id; Name; userpic; Email; Gender; Locale; LastVistit; Roles; UserGroups; UserDesks; UserTasklists UserTasks.
Group	Описание сущности группы	Id; Name; UserGroups.

Продолжение таблицы 3.12

Наименование	Назначение	Поля
UserGroups	Описание связи между сущностью пользователя и группы	Id; User; Group; Role.

Ниже представлена таблица 3.13 с единицами заданий пользователей.

Таблица 3.13 – Классы сущностей заданий и проектов.

Наименование	Назначение	Поля
Task	Описание сущности задания	Id; Name; Description; Start_time; End_time; Complexity; HeaderTaskList; Stage; UserComplexity; TaskList; UserTask; TaskRelation.

Продолжение таблицы 3.13

Наименование	Назначение	Поля
TaskList	Описание сущности списка заданий	Id; Name; Description; Start_time; End_time; UserNumber; Tasks; Desk; userTaskLists.
Desk	Описание сущности доски	Id; Name; TaskLists; UserDesks.

Ниже представлена таблица 3.14, описывающая классы для связи между сущностью пользователя и единиц заданий или проектов.

Таблица 3.14 – Классы описания связи между пользователем и заданиями.

Наименование	Назначение	Поля
UserTask	Описание связи между сущностью пользователя и задания	Id; User; Task; Role.

Продолжение таблицы 3.14

Наименование	Назначение	Поля
UserTaskList	Описание связи между сущностью пользователя и списка заданий	Id; User; Tasklist; Role.
UserDesk	Описание связи между сущностью пользователя и доски	Id; User; Desk; Role.

В таблице 3.15 указаны классы для описания ролей в системе.

Таблица 3.15 – Классы описания ролей.

Наименование	Назначение	Поля
Role	Описание ролей (полномочий) в системе	USER; ADMIN; GUEST; USER_REQ.

3.5. Описание классов репозиториев

Для обращения к базе данных необходим слой репозиториев. Java Spring Boot берет на себя работу с подключением и настройкой базы данных. Необходимо в свойствах программы указать необходимый адрес, пользовательское имя и пароль.

Листинг 3.2 – Задаваемые свойства в приложении

```
spring.datasource.url=jdbc:postgresql://localhost/feather
spring.datasource.username=denisdb
spring.datasource.password=123
```

От разработчика лишь требуется декларативно описать интерфейс и его функции, которые выполняют строго определенную задачу. Например, вытащить из базы данных все записи пользователей. Также фреймворк берет на себя и десериализацию в объекты.

Ниже представлена таблица 3.16 с описанием репозиториев пользователей и групп.

Таблица 3.16 – Репозитории пользователей и групп.

Наименование	**Назначение**	**Функции**
UserRepo	Репозиторий для обращения к таблице usr	FindUserById; FindUserByName; FindAllById;
GroupRepo	Репозиторий для обращения к таблице grp	FindGroupById;
UserGroupRepo	Репозиторий для обращения к таблице связи usr_grp	findUserGroupById findUserGroupByUserAndGroup FindUserGroupsByGroup findUserGroupsByUser

В таблице 3.17 указаны репозитории для обращения к таблицам заданий и проектов, а также связей между ними и пользователем.

Таблица 3.17 – Репозитории заданий и проектов.

Наименование	**Назначение**	**Функции**
TaskRepo	Репозиторий для обращения к таблице task	FindTaskById;
TaskListRepo	Репозиторий для обращения к таблице task_list	FindTaskListById; FindTaskListByDesk;
DeskRepo	Репозиторий для обращения к таблице desk	Find DeskById;
UserTaskRepo	Репозиторий для обращения к таблице связи user_task	FindUserTaskByUserAndTask; FindUserTaskByUser; FindUserTaskByTask;
UserTaskListRepo	Репозиторий для обращения к таблице связи user_list	FindUserTasklistByUserAndTaskList; FindUserTasklistByUser; FindUserTasklistByTaskList;
UserDeskRepo	Репозиторий для обращения к таблице user_desk	FindUserDeskByUserAndDesk; FindUserDesksByUser; FindUserDesksByDesk;

В таблице 3.18 указан репозиторий для системы аутентификации.

Таблица 3.18 – Репозиторий для аутентификации.

Наименование	Назначение	Функции
UserDetailsRepo	Репозиторий для аутентификации	

Исходный код классов репозиториев приведен в листинах С.1 – У.1 приложений С – У.

3.6. Описание классов конфигурации и ошибок

Для функционирования приложения на фреймворке Java Spring Boot необходимо провести начальную конфигурацию. Описание класса для конфигурации располагается в таблице 3.19.

Таблица 3.19 – Класс конфигурации.

Наименование	Назначение	Функции
WebSecurityConfig	Описание конфигурации аутентификации пользователя и начальной регистрации	Configure; PrincipalExtractor;

В данном классе описывается доступ к файлам проекта и ресурсам пользователей входящих в систему, а также в методе PrincipalExtractor производится начальная регистрация пользователя с выдача соответствующих прав.

Также в таблице 3.20 приведен класс для описания ошибок на стороне сервера. Исходный код класса ошибки приведен в листинге Ф.1. приложения Ф.

Таблица 3.20 – Класс для описания и выдачи ошибок.

Наименование	Назначение	Функции
NotFoundException	Появление объекта данного класса – означает, что запрашиваемый объект не был найден	

3.7. Описание классов сервисов

Для обработки какой-либо сложной бизнес-логики существует слой сервисов. В данном слое в проекте обрабатывается работа с графами. В таблице 3.21 приведен список сервисов. Исходный код классов сервисов приведен в листингах Ц.1. – Щ.1. приложений Ц – Щ.

Таблица 3.21 – Классы сервисов приложения.

Наименование	Назначение	Поля	Функции
EffectiveWalkService	Класс предназначен для описания логики оптимального прохождения по списку заданий	TaskRepo; TaskListRepo;	BFS(); CheckDouble(); ProcessNodes(); GetAllTaskOfList(); IsRelatedWithAnother().

В данном классе для работы с графами необходимы ссылки на репозитории – TaskRepo и TaskListRepo. С помощью данных полей есть возможность совершения чтения или записи базы данных.

Главный с точки зрения функционала метод в классе - BFS(). Данный метод реализует основную логику оптимального прохождения графа. В нем инициализируется и наполняется очередь.

В функции ProcessNodes() обрабатываются узлы которые располагались на одной ширине с точки зрения алгоритма, то есть равноудаленные от корня графа. В последней упомянутой функции реализуется подсчет выдачи количества людей на узел.

В функции checkDouble() проверяется, не является ли дублирующей данный в очереди узел.

GetAllTaskOfList() выдает список всех заданий данного списка.

Метод IsRelatedWithAnother() проверяет, является ли проверяемый узел дочерним от другого, ещё не пройденного узла или нет. Последний метод необходим для устранения ошибки наполнения очереди преждевременными узлами.

3.8. Описание контроллеров

Контроллеры в программе не имеют особой бизнес-логики. Основное назначения контроллер принятие запросов, делегирование работы на необходимые сервисы или репозитории, и создание последующего ответа.

Контроллеры в программе разделены на три группы: REST-контроллеры для обращения к сущностям пользователей и групп, REST-контроллеры для обращения к сущностям заданий и проектов, REST-контроллеры для обращения к сущностям связей между заданиями или проектам и пользователями.

Ниже представлена таблица 3.22 с описанием контроллеров сущностей пользователя и групп. Исходный код контроллеров представлен в листингах Я.1. – J.1. приложений Я – J, точки входа в программу в листинге K.1. приложения K.

Таблица 3.22 – Контроллеры сущностей пользователя и групп.

Наименование	**Назначение**	**Поля**	**Методы**
UserRestController(“rest/user”)	Контроллер для работы с сущностью пользователя	UserRepo; GroupRepo;	FindAll(); FindById(); SetRole();
GroupRestController(“rest/group”)	Контроллер для работы с сущностью группы	GroupRepo;	FindById(); Add(); Delete();
UserGroupRestController(“rest/user-group”)	Контроллер для работы с таблицей связи пользователя и группы	UserRepo; GroupRepo; UserGroupRepo;	GetAllUserGroups(); GetUserGroup(); GetByUserId(); GetByGroupId(); SetRelation(); DeleteRelation();

В таблице 3.23 указаны контроллеры для работы с сущностями отдельного задания, списка заданий и проектов.

Таблица 3.23 – Контроллеры заданий, списков и проектов.

Наименование	Назначение	Поля	Методы
TaskRestController(“rest/task”)	Контроллер для работы с задания-ми	TaskRepo; TaskListRepo;	FindAllTasks(); FindTaskById(); FindTaskByTask-List(); AddTask(); Delete(); GetParentTaskList(); SetTaskRel();
TaskListRestController (“rest/tasklist”)	Контроллер для работы со списком заданий	TaskRepo; TaskListRepo; DeskRepo; EffectiveWalkServi ce;	FindAllTaskLists(); FindTaskListBiId(); AddTaskList(); Delete(); GetAllTasksOf-Desk2();
DeskRestController(“rest/desk”)	Контроллер для работы с дос-кой(проектом)	DeskRepo; TaskListRepo; TaskRepo; EffectiveWalkServi ce;	FindAllDesks(); FindDeskById(); AddDesk(); Delete(); GetAllTasksOf-Desk();

В таблице 3.24 указаны контроллеры для работы с таблицами связей заданий или проектов и пользователей.

Таблица 3.24 – Контроллеры для работы с таблицами связей заданий или проектов и пользователей.

Наименование	**Назначение**	**Поля**	**Методы**
UserTaskController	Контроллер для работы с таблицей связи user_task	UserRepo; TaskRepo; UserTaskRepo;	FindAll(); FindUserTask(); FindAllUsersByTask(); FindAllTasksByUser(); SetRelation(); DeleteRelation();
UserTaskListController	Контроллер для работы с таблицей связи user_taslist	UserRepo; UserListRepo; UserTaskListRepo;	FindUserTaskList(); FindAllUsersByTasklist(); FindAllTasklistsByUser(); SetRelation(); DeleteRelation().
UserDeskController	Контроллер для работы с таблицей связи user_desk	UserRepo; DeskRepo; UserDeskRepo;	FindUser-Desk(); findAllUsersByDesk();FindAllDesksByUser(); SetRelation(); DeleteRelation().

3.9. Описание приложения SPA

Для реализации SPA (Single Page Application) был выбран в п.1 фронт-энд фреймворк Vue.js. Для построения больших приложений разработчики данного фреймворка добавили возможность создания однофайловых компонентов. Однофайловые компоненты – это соединенные в одном .vue файле отображение в формате HTML(Hyper Text Markup Language), скрипты на языке JavaScript и стили с использованием CSS(Cascading Style Sheets) .

В SPA реализован только единственный index.html файл. В данном файле присутствует следующая строка Листинга 3.3.

Листинг 3.3 – Файл index.html

```
<div id="app"></div>
```

Данная строка и представляет все SPA приложение.

В скриптах был создан файл main.js. Основное назначение данного файла – запуск SPA. В листинге 3.4 приведен основной исходный код файла.

Листинг 3.4 – Файл main.js

```
new Vue({
     el: '#app',
     router,
     store,
     render: a => a(App)
})
```

Важно отметить, что router и store – это особые компоненты. Первый необходим для маршрутизации по приложению, второй – для централизованного кэшированного хранения данных. Последнее особо нужно при многокомпонентном приложении, каковым и является текущий проект.

В таблице 3.25 указаны самые основные однофайловые компоненты.

Таблица 3.25 – Основные однофайловые компоненты.

Название	Назначение
App.vue	Основной шаблон приложения
MainMenu.vue	Шаблон главного меню приложения

В таблице 3.26 указаны однофайловые компоненты администратора.

Таблица 3.26 – Однофайловые компоненты администратора.

Название	Назначение
AdminVue.vue	Шаблон страницы админ панели

В таблице 3.27 указаны однофайловые компоненты списка заданий или проектов.

Таблица 3.27 – Однофайловые компоненты списка заданий или проектов.

Название	Назначение
DeskList.vue	Шаблон списка досок
TaskList.vue	Шаблон списка заданий
TasklistList.vue	Шаблон списка листа заданий

В таблице 3.28 указаны однофайловые компоненты создания заданий или проектов.

Таблица 3.28 – Однофайловые компоненты создания заданий или проектов.

Название	Назначение
Construct.vue	Шаблон главного меню создания

Продолжение таблицы 3.28

Название	Назначение
DeskConsctruct.vue	Шаблон создания досок
TaskListConstruct.vue	Шаблон создания списка заданий
TaskConstruct.vue	Шаблон создания заданий

В таблице 3.29 указаны однофайловые компоненты редактирования заданий или проектов.

Таблица 3.29 – Однофайловые компоненты редактирования заданий или проектов.

Название	Назначение
DeskEdit.vue	Шаблон редактирования доски
TaskListEdit.vue	Шаблон редактирования списка заданий

В таблице 3.30 указаны однофайловые компоненты графа заданий.

Таблица 3.30 – Однофайловые компоненты графа заданий.

Название	Назначение
AllTasksOfDesk.vue	Шаблон отображения графа заданий доски
EffectiveViewOfTaskList.vue	Шаблон отображения графа оптимального выполнения списка заданий

В таблице 3.31 указаны однофайловые компоненты сообщества.

Таблица 3.31 – Однофайловые компоненты сообщества.

Название	**Назначение**
CommunityProject.vue	Шаблон отображения списка всех проектов в системе
CommunityView.vue	Шаблон отображения главного меню сообщества

В таблице 3.32 указаны однофайловые компоненты уведомления.

Таблица 3.32 – Однофайловые компоненты сообщества.

Название	**Назначение**
Agreement.vue	Шаблон отображения уведомления о статусе пользователя в системе

Примеры исходного кода однофайловых компонентов представлены в листингах L.1. – M.1. приложений L – M.

4. РЕАЛИЗАЦИЯ

4.1. Форма входа в приложение

При входе в приложение неавторизованный пользователь попадает на главное домашнее окно, представленное на рисунок 4.1.

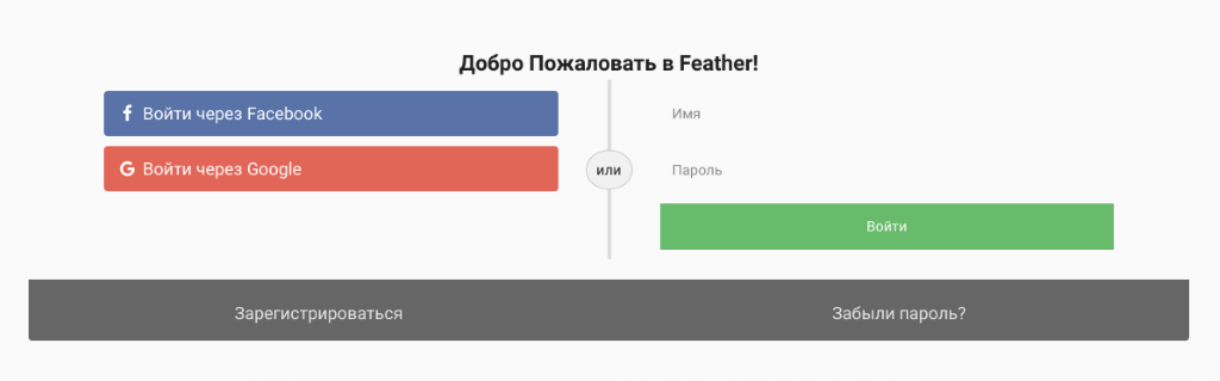

Рисунок 4.1 – Домашнее окно гостевого пользователя

Если пользователь принял решение войти в систему, ему необходимо авторизоваться через Google OAuth2.0. Данное окно представлено на рисунок 4.2.

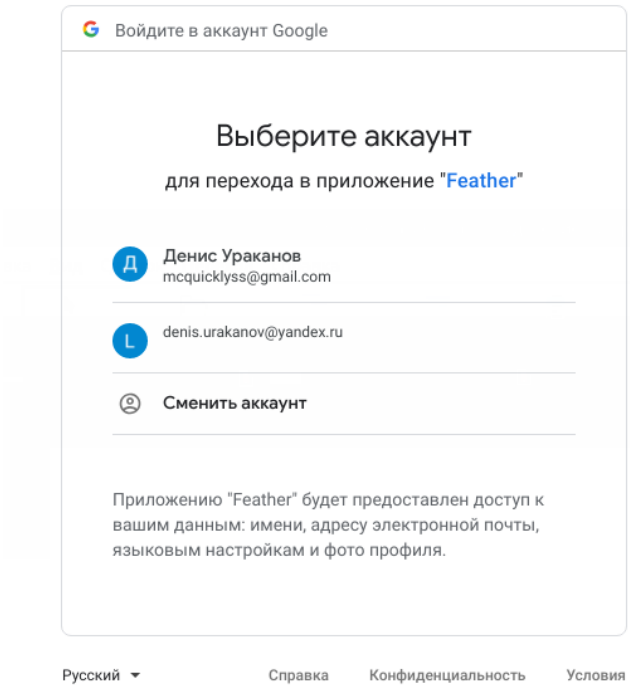

Рисунок 4.2 – Аутентификация пользователя через Google OAuth2.0

4.2. Реализация OAuth 2.0

Общая схема работы приложения, использующего OAuth, такова:

1. Получение авторизации.
2. Обращение к защищенным ресурсам.

Результатом авторизации является access token — некий ключ (обычно просто набор символов), предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного access token'a.

В протоколе описано несколько вариантов авторизации, подходящих для различных ситуаций:

- авторизация для приложений, имеющих серверную часть (чаще всего, это сайты и веб-приложения);
- авторизация для полностью клиентских приложений (мобильные и desktop-приложения);
- авторизация по логину и паролю;
- восстановление предыдущей авторизации.

Общая схема авторизации приложения представлена на рисунке 4.3.

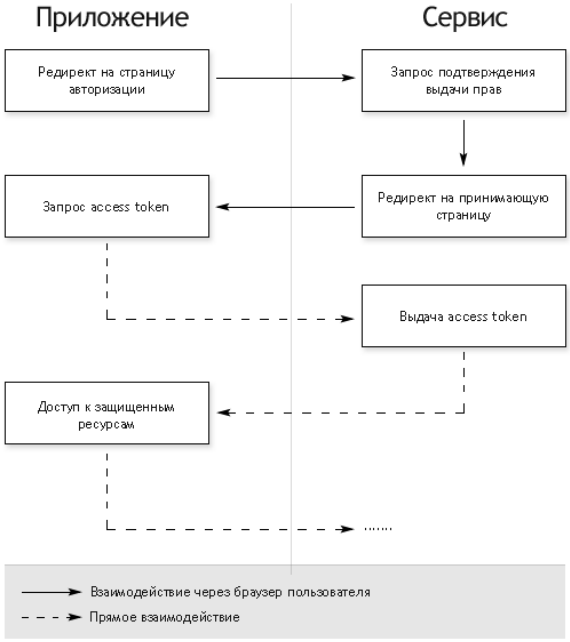

Рисунок 4.3 – Схема авторизации приложения OAuth 2.0

Шаги выполнения авторизации:

1. Редирект на страницу авторизации.
2. На странице авторизации у пользователя запрашивается подтверждение выдачи прав.
3. В случае согласия пользователя, браузер редиректится на URL, указанный при открытии страницы авторизации, с добавлением в GET-параметры специального ключа — authorization code.

4. Сервер приложения выполняет POST-запрос с полученным authorization code в качестве параметра. В результате этого запроса возвращается access token.

Окно личного кабинета для управления авторизацией своими веб-приложениями в Google API представлено на рисунке 4.4.

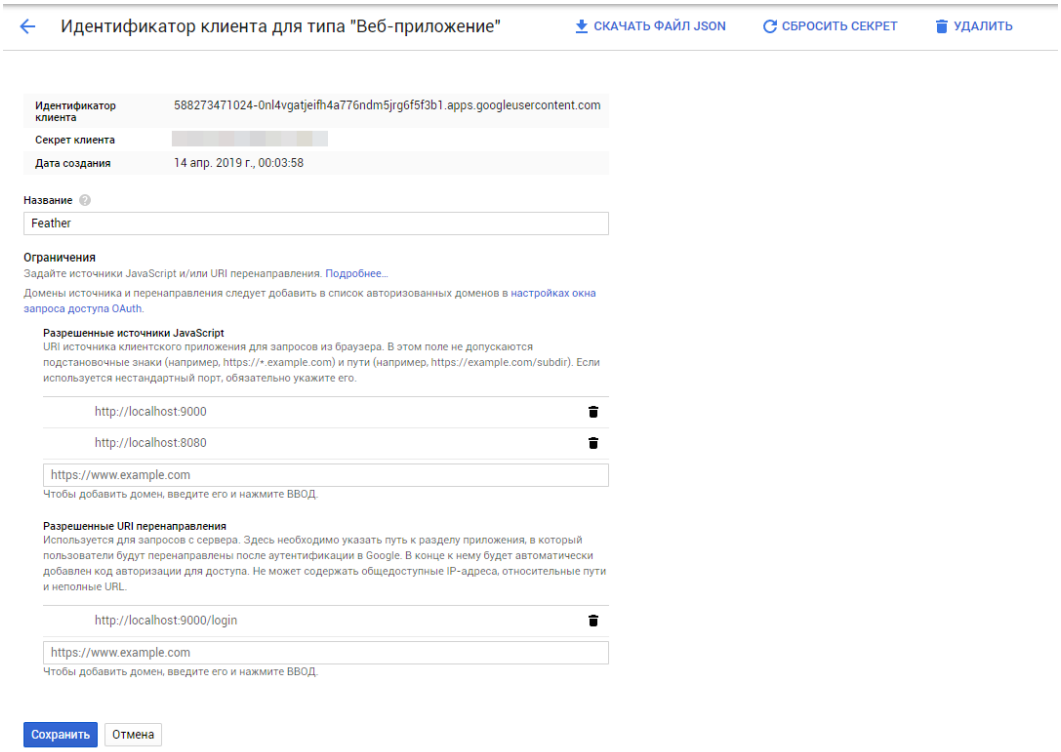

Рисунок 4.4 – Личный кабинет Google API

4.3. Главное окно приложения

Сразу же после входа в приложение окно пользователя представляет собой главное меню с плашками, ассоциирующими основные маршруты по сервису:

- мои проекты;
- мои команды;
- создание проекта(доски);

- создание списка заданий;
- создание заданий;
- сообщество “Feather”.

Данная страница представлена на рисунке 4.5.

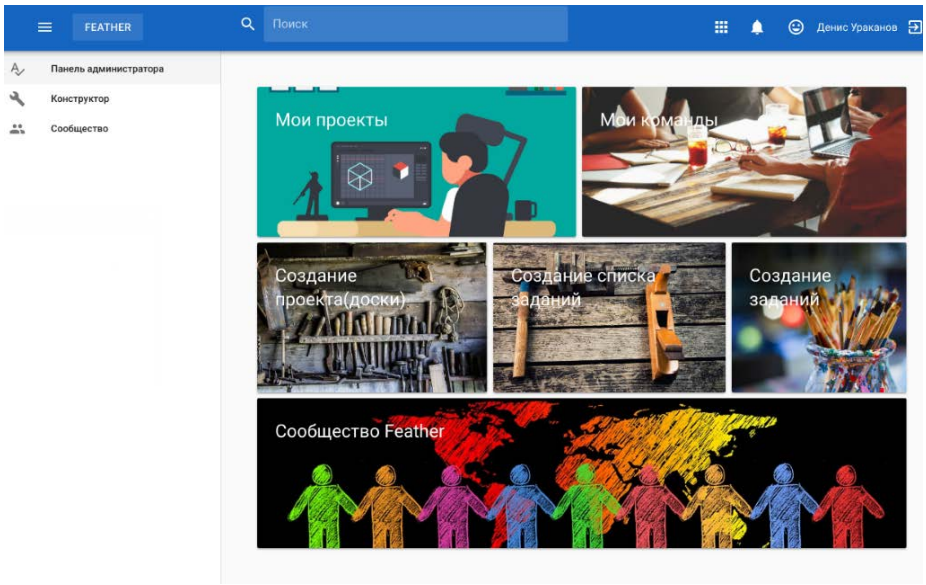

Рисунок 4.5 – Главная страница приложения

Боковую панель всего приложения можно в любой момент спрятать, нажав на соответствующий значок в виде трёх горизонтальных полос в верхнем левом углу экрана. Меню без боковой панели представлена на рисунке 4.6.

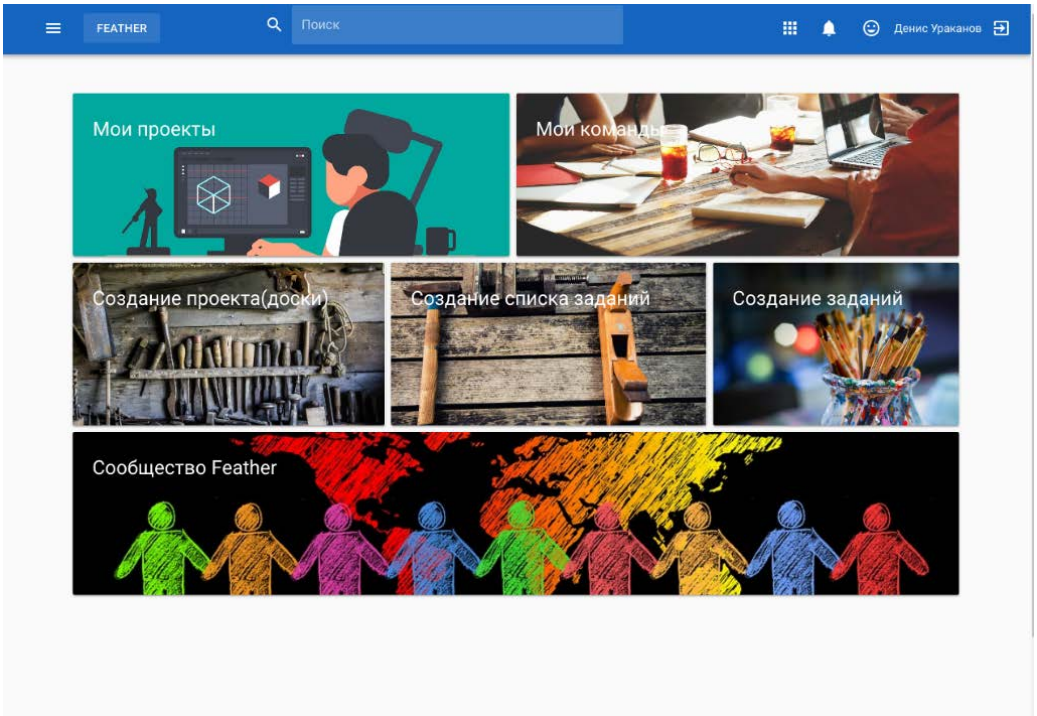

Рисунок 4.6 – Главная страница со спрятанной боковой панелью

Следует отметить, что оформление веб-сервиса является адаптивным к устройствам вывода информации. К примеру, на рисунках 4.7 – 4.8 представлены изображения мобильных версий в браузере Google Chrome версии 72.0.3626.119 с использованием инструментов для веб-разработчика по тестированию отображения на различных устройствах.

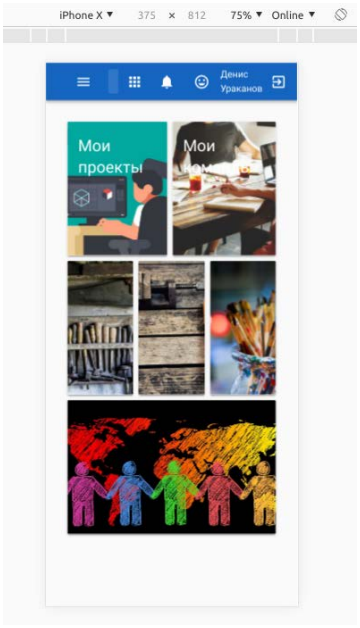

Рисунок 4.7 – Пример главной страницы на iPhone X

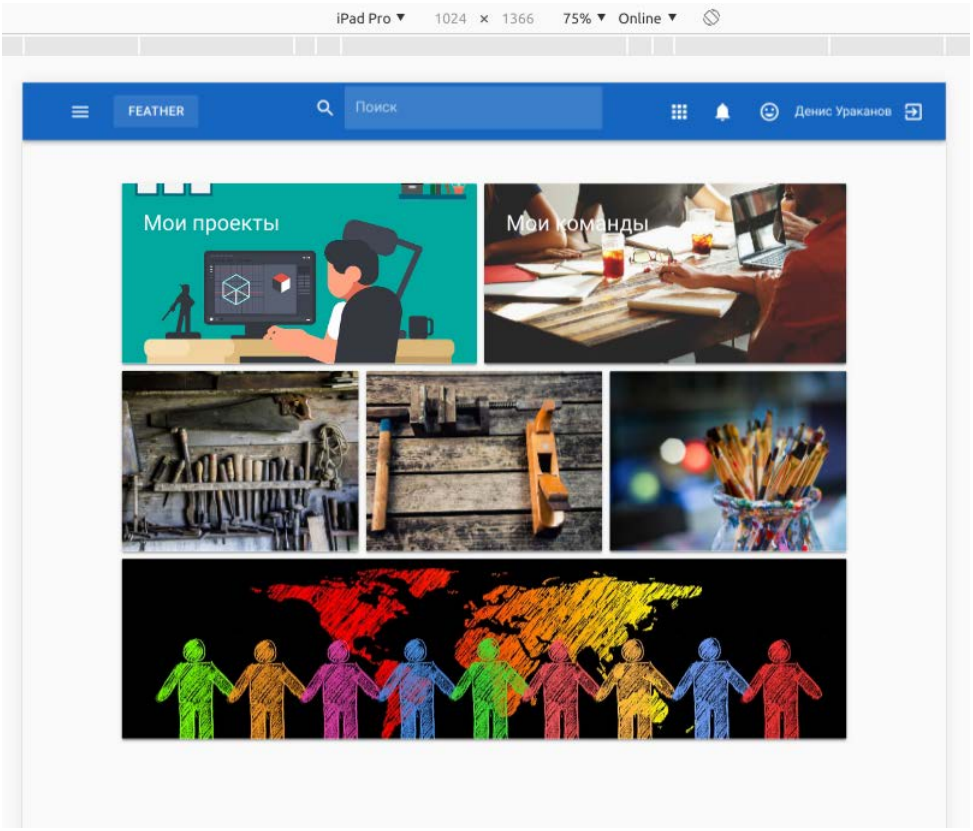

Рисунок 4.8 – Пример главной страницы на iPad Pro

4.4. Личный кабинет пользователя

Любому авторизованному пользователю предоставляется личный кабинет, то есть страница, на которой доступно изменение параметров его учетной записи и статусы его заявок или приглашений в группу или проект. На рисунке 4.9 представлен скриншот личного кабинета пользователя.

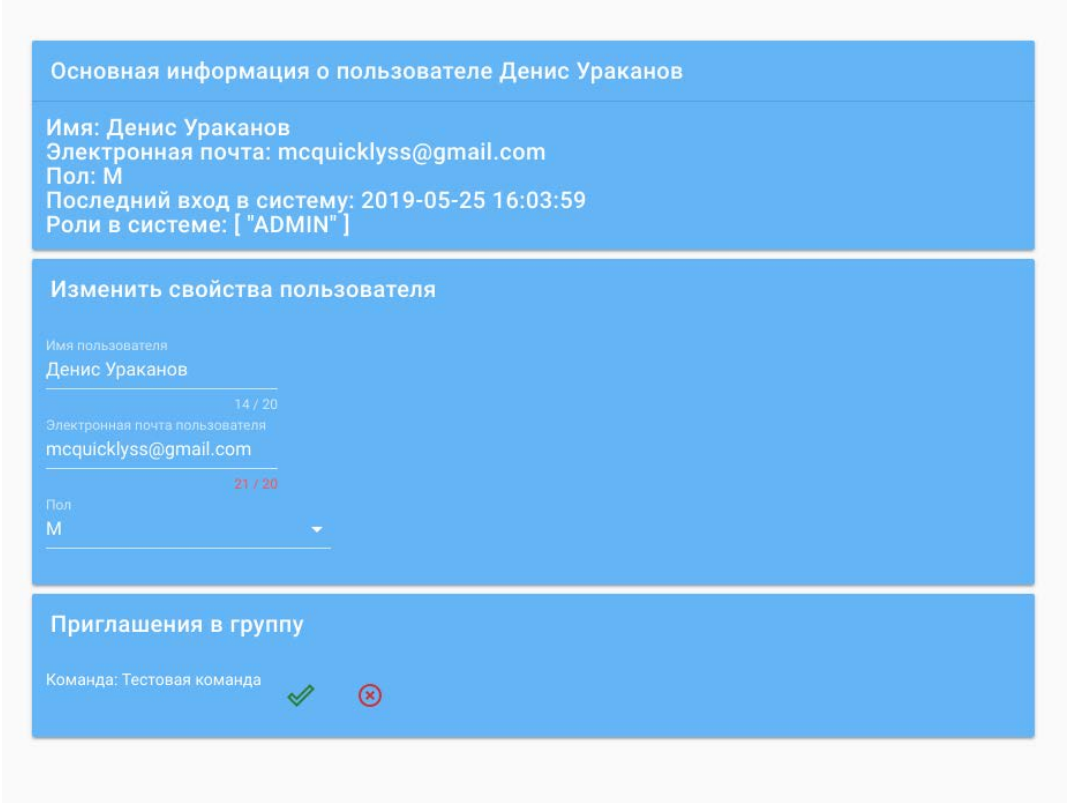

Рисунок 4.9 – Личный кабинет пользователя

4.5. Создание и отображение заданий и проектов

Переход на создание проекта (т.е. доски) происходит несколькими путями. Первый при нажатии на соответствующую плашку в главном меню (рисунок 4.6). Второй – при нажатии в боковом меню «Конструктор» и выбрав «Создать новую доску». Пример второго способа представлен на рисунке 4.10.

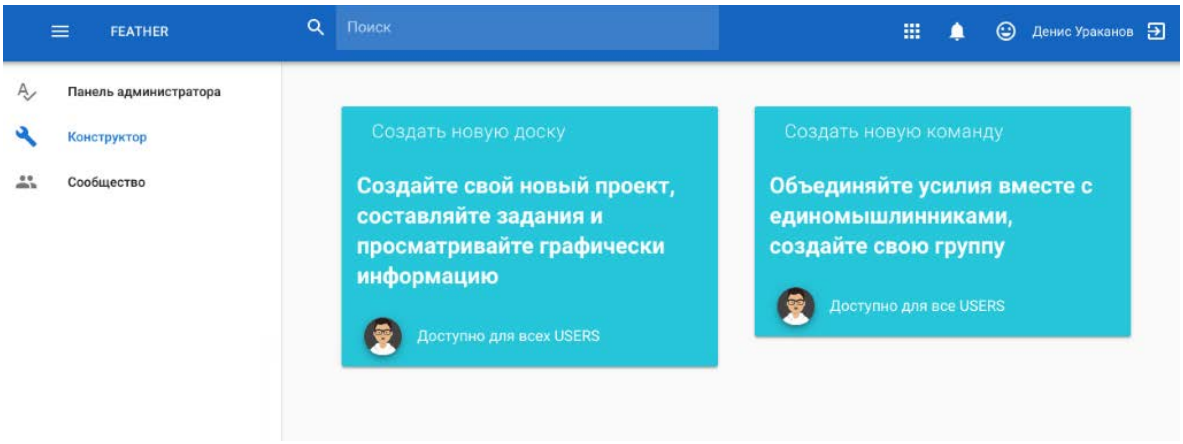

Рисунок 4.10 – Меню конструктора

Последний, третий вариант, связан с созданием проекта, привязанным к конкретной команде. Этот случай будет рассмотрен позже.

Первые два варианта ведут в меню создания проекта. На рисунках 4.11 – 4.13 представлены шаги создания проекта.

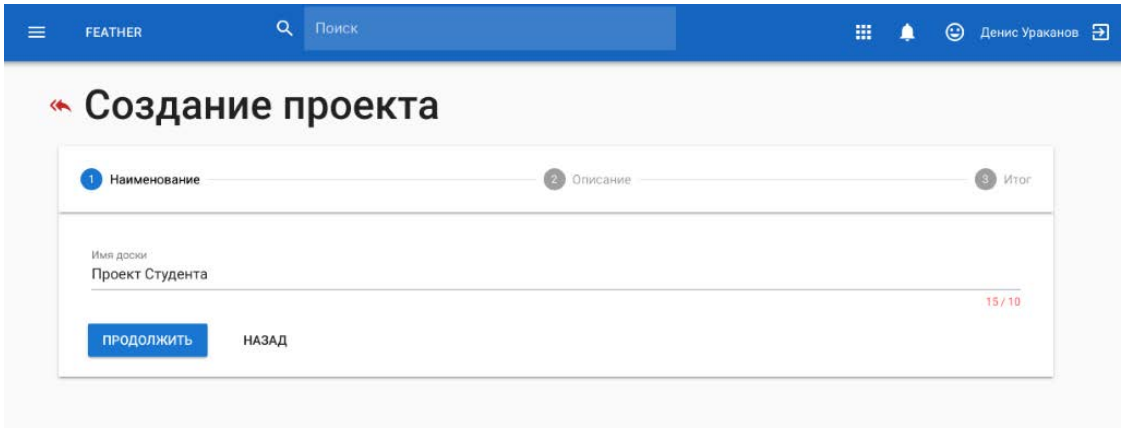

Рисунок 4.11 – Первый шаг создания проекта

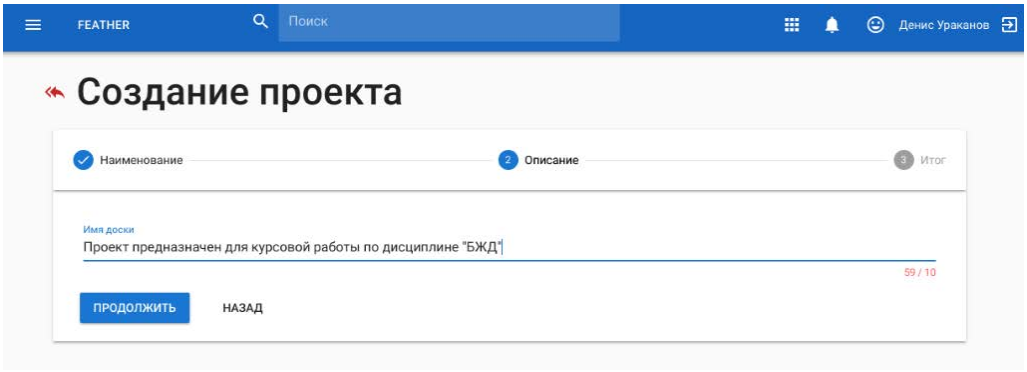

Рисунок 4.12 – Второй шаг создания проекта

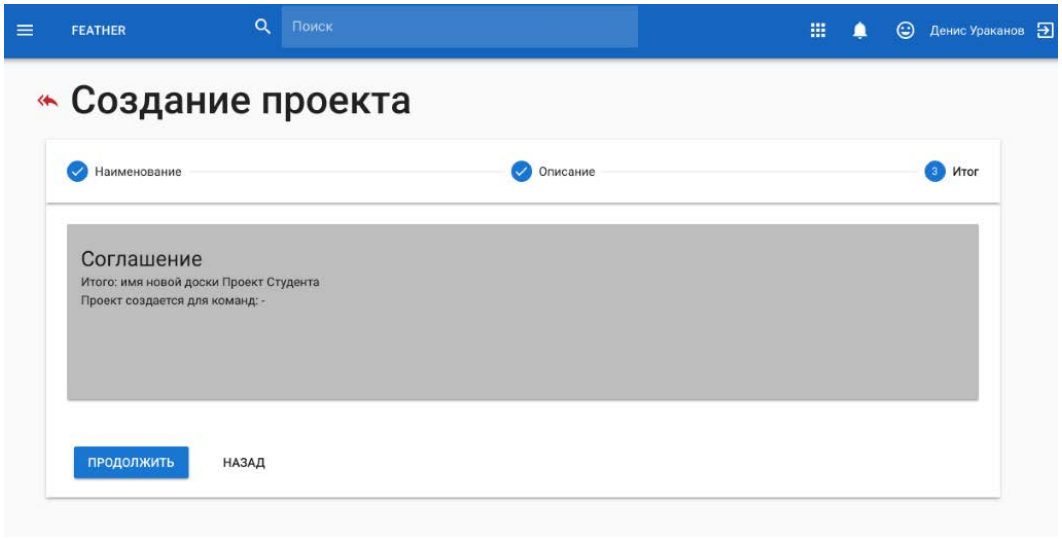

Рисунок 4.13 – Третий шаг создания проекта

После нажатия на кнопку «Продолжить» последнего шага пользователь перейдет в список проектов, связанных с данным пользователем (рисунок 4.14).

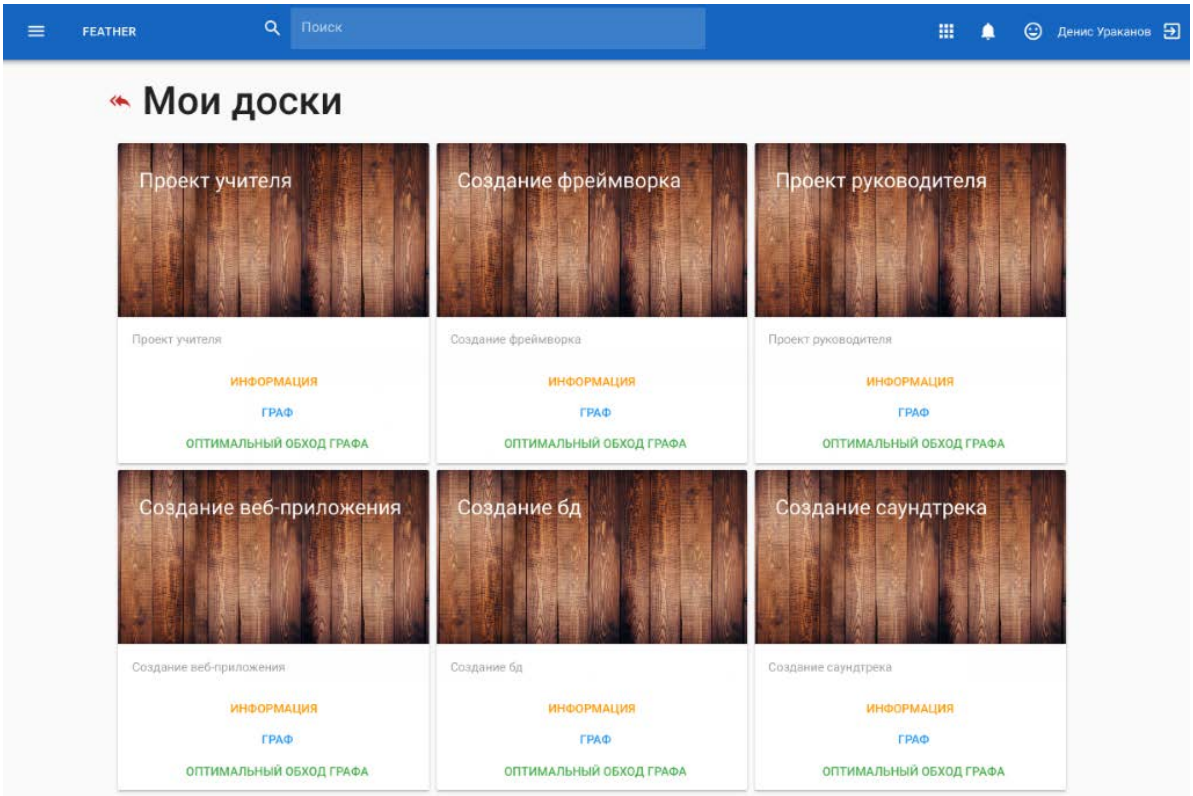

Рисунок 4.14 – Список проектов данного пользователя

Создать список задач (то есть связанные друг с другом атомарные задачи в общем контексте) можно двумя путями: через главное меню (рисунок 4.6) или нажав на определенную доску в списке проектов (рисунок 4.14), чтобы просмотреть списки заданий. Затем следует выбрать соответствующую опцию по «плавающей» кнопке в правом нижнем углу экрана (рисунок 4.15).

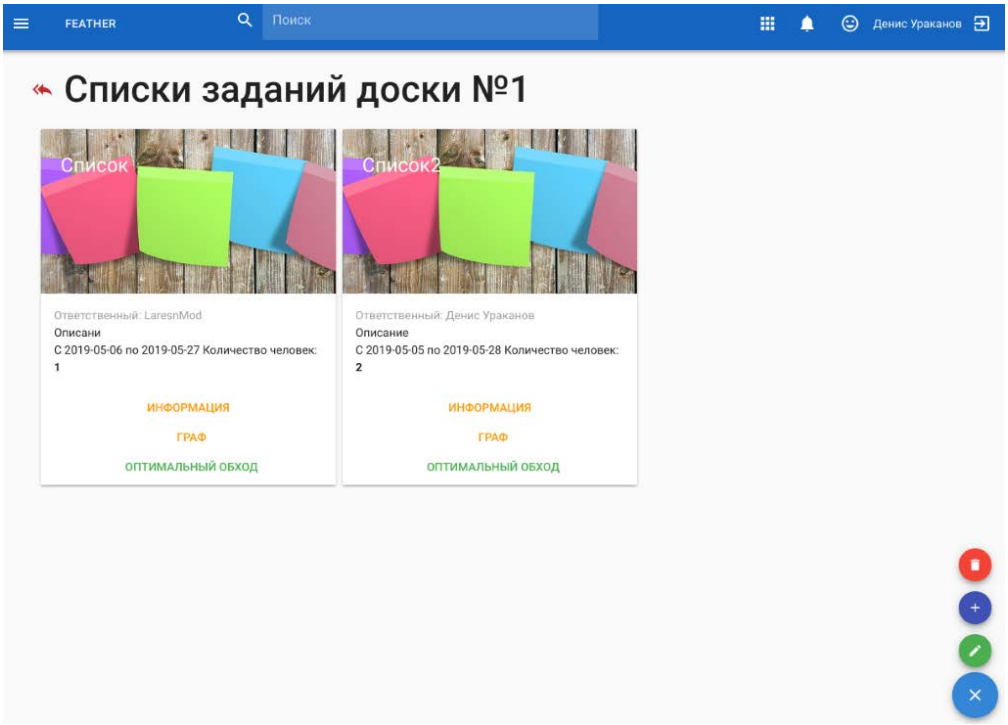

Рисунок 4.15 – Список задач определенного проекта

Данная угловая кнопка является раскрывающейся и при нажатии предлагает удалить данную доску, добавить новый список задач или перейти в редактирование профиля проекта (сверху вниз) (рисунки 4.16 – 4.17).

Рисунок 4.16 – Нераскрытая кнопка

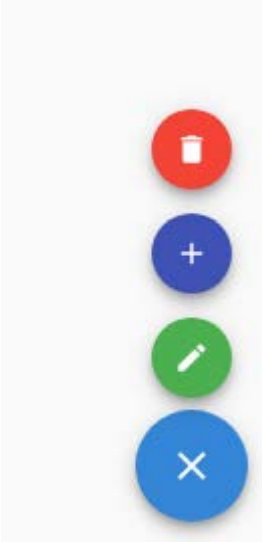

Рисунок 4.17 – Раскрытая кнопка

При создании через главное меню необходимо указать проект на странице связанных с данным пользователем проектов (рисунок 4.18).

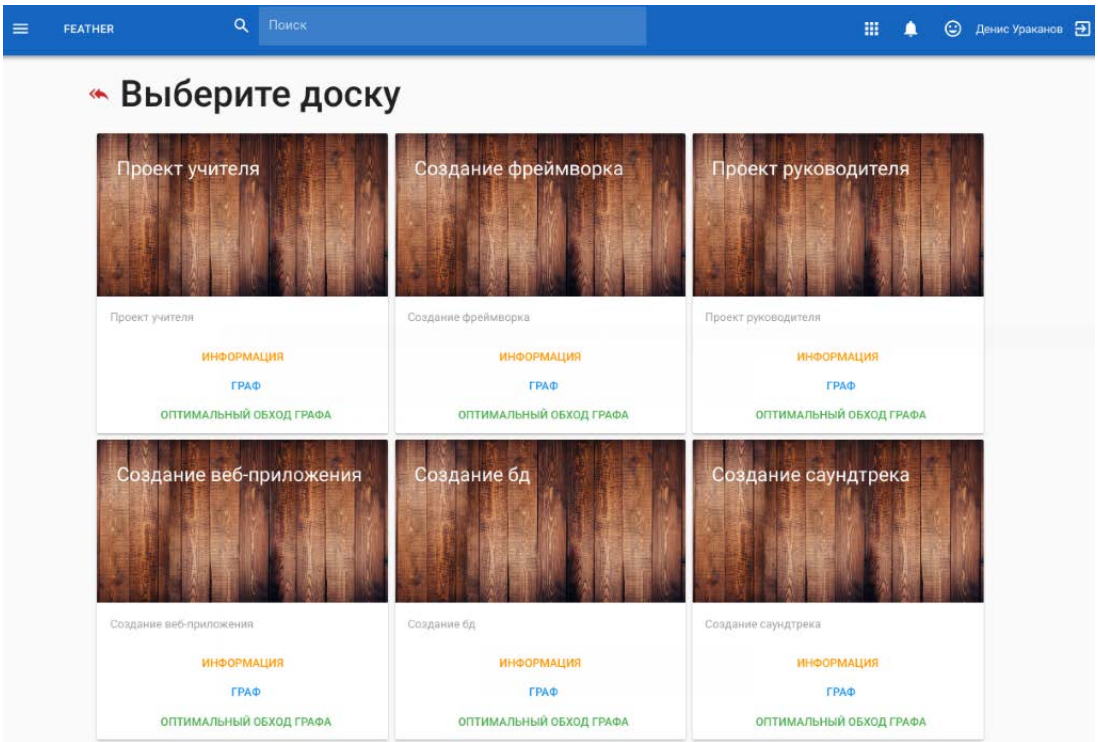

Рисунок 4.18 – Выбор проекта при создании списка задач

На рисунках 4.19 – 4.25 представлены шаги создания списка задач.

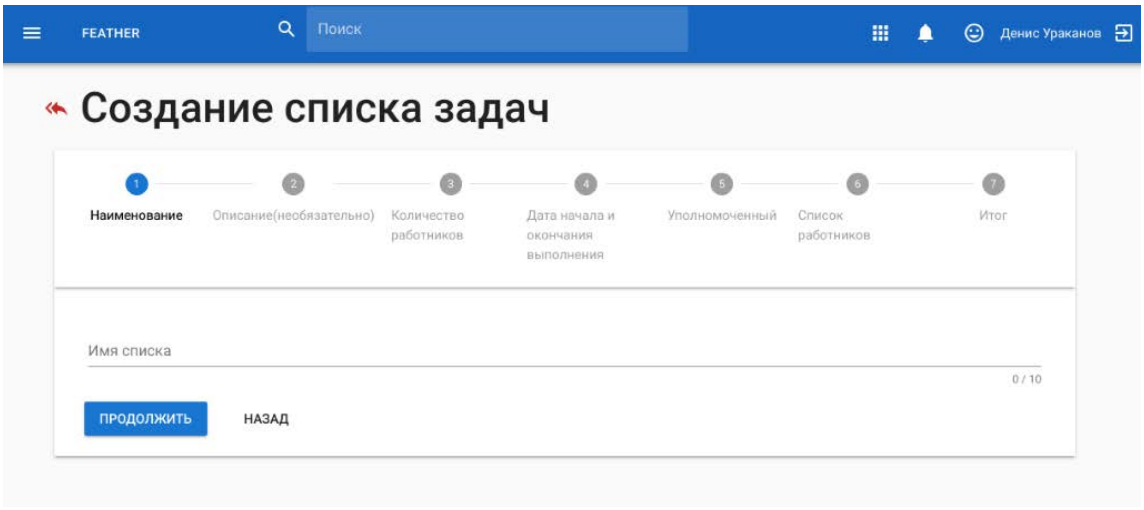

Рисунок 4.19 – Первый шаг создания списка задач

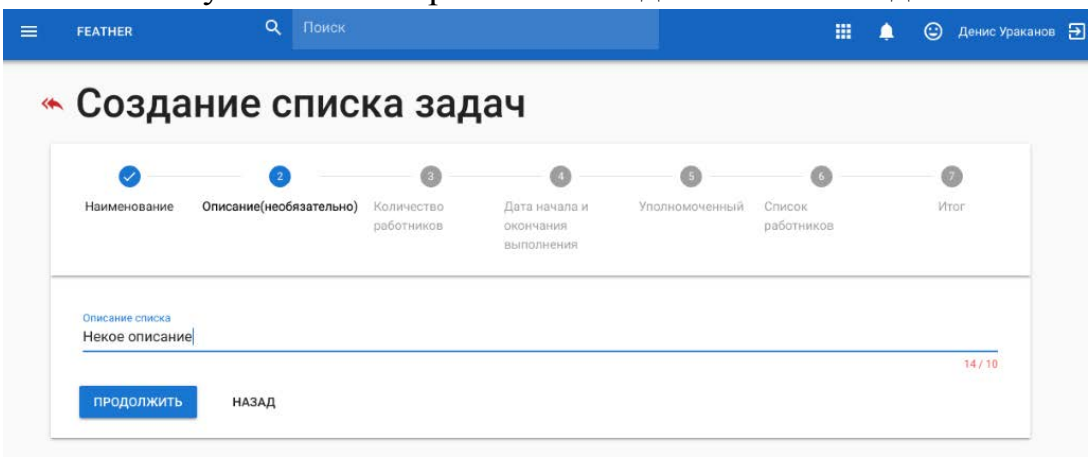

Рисунок 4.20 – Второй шаг создания списка задач

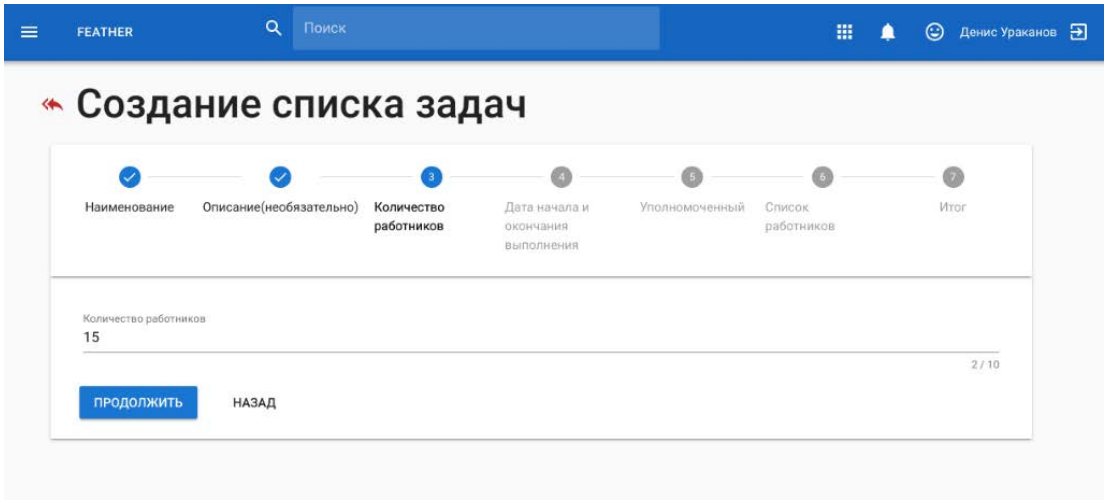

Рисунок 4.21 – Третий шаг создания списка задач

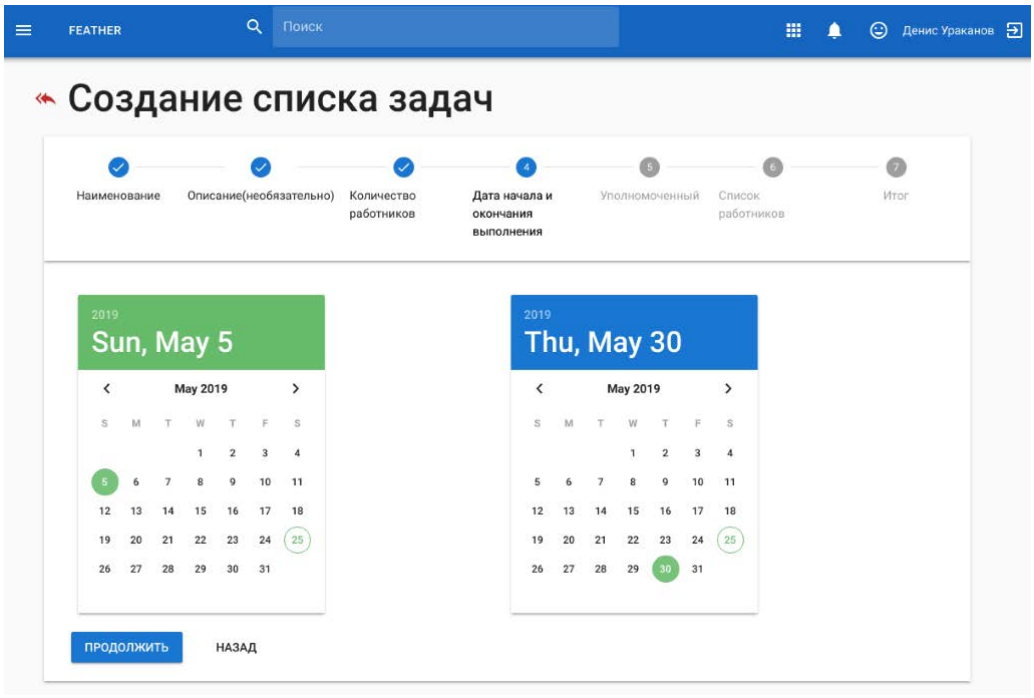

Рисунок 4.22 – Четвертый шаг создания списка задач

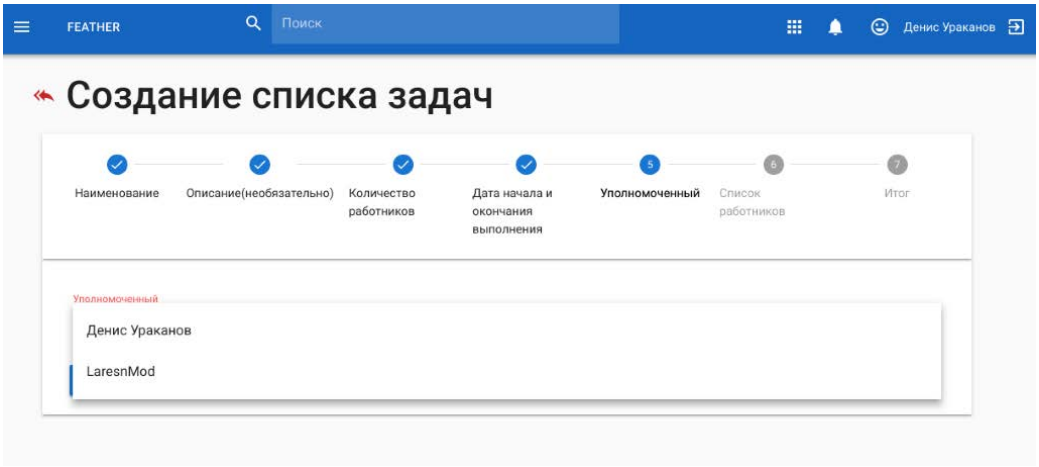

Рисунок 4.23 – Пятый шаг создания списка задач

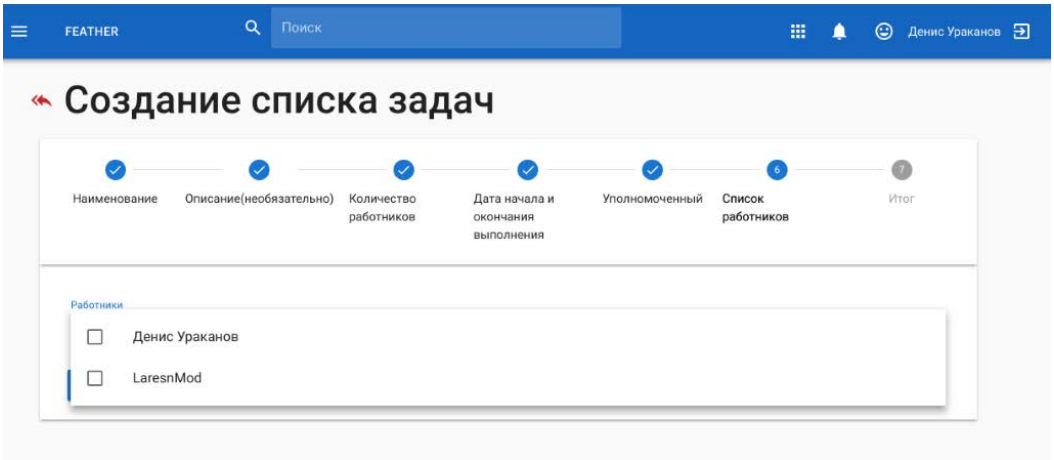

Рисунок 4.24 – Шестой шаг создания списка задач

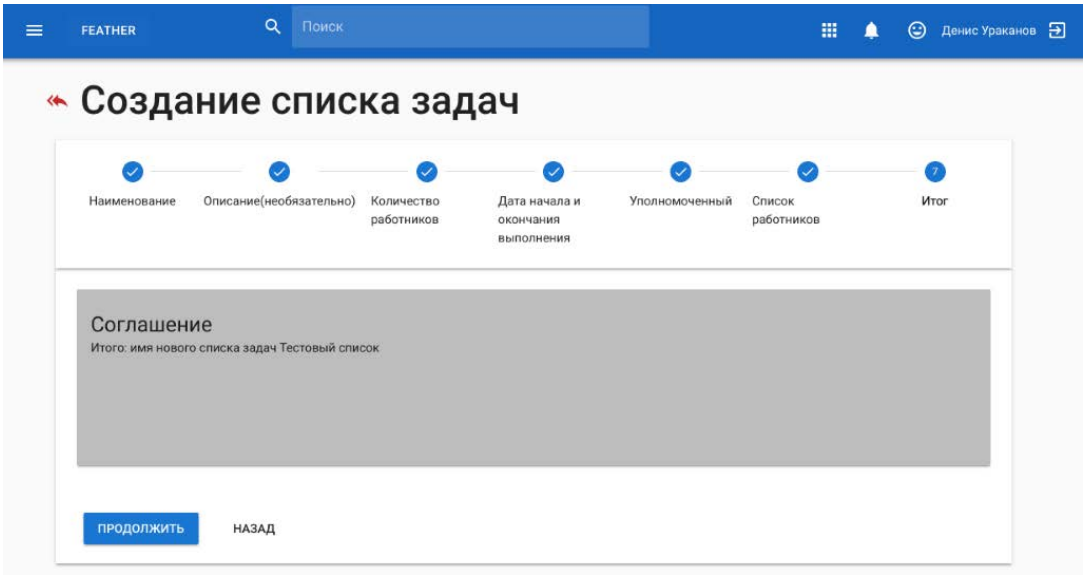

Рисунок 4.25 – Седьмой шаг создания списка задач

Для того, чтобы создать новую задачу, можно воспользоваться опцией в главном меню приложения (рисунок 4.6) или, нажав на соответствующий список, как и в случае со списком задач, раскрыть угловую кнопку и из выбора удаления списка, добавления задачи и редактирования списка выбрать вторую опцию.

При переходе из главного меню пользователю будет предложено выбрать соответствующий проект или список задач (рисунки 4.26 - 4.27).

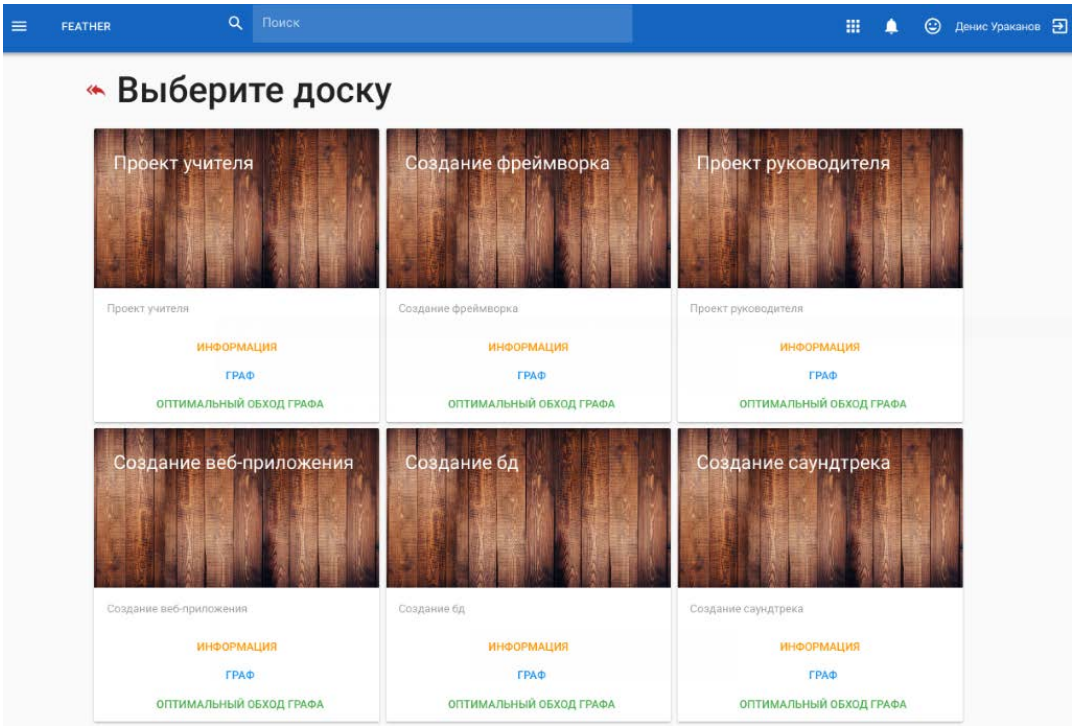

Рисунок 4.26 – Выбор проекта

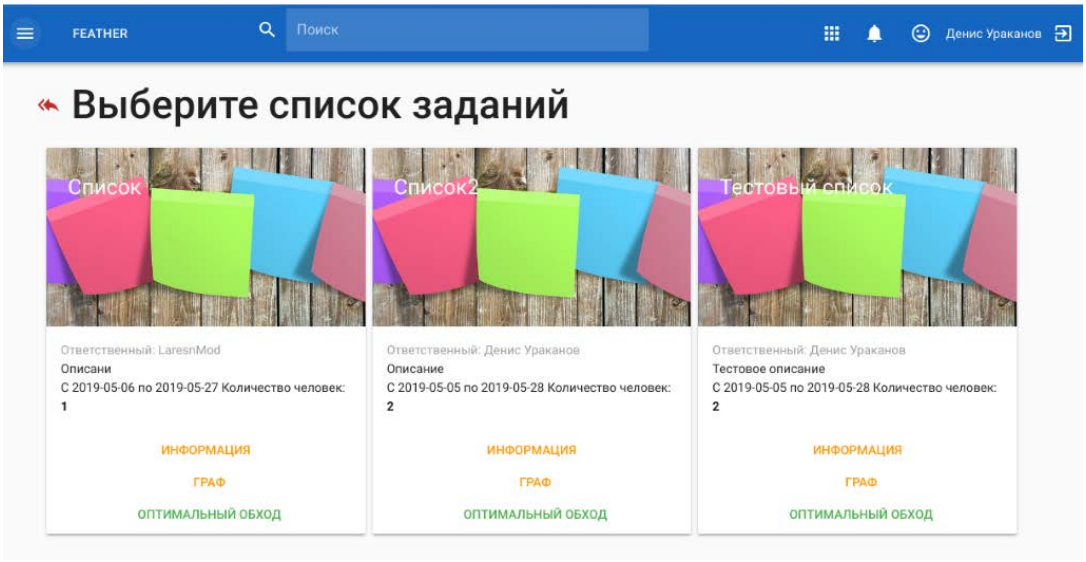

Рисунок 4.27 – Выбор списка задач

Создание задачи представлено на рисунках 4.28 – 4.35.

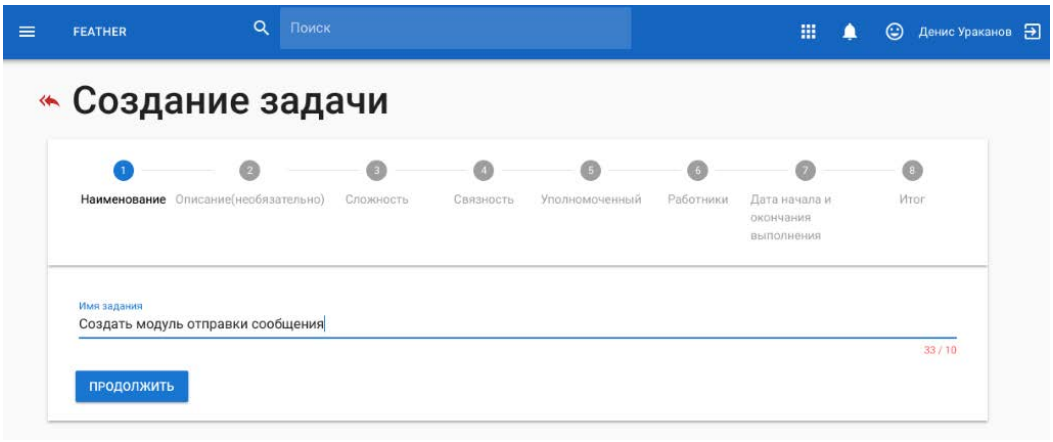

Рисунок 4.28 – Первый шаг создания задачи

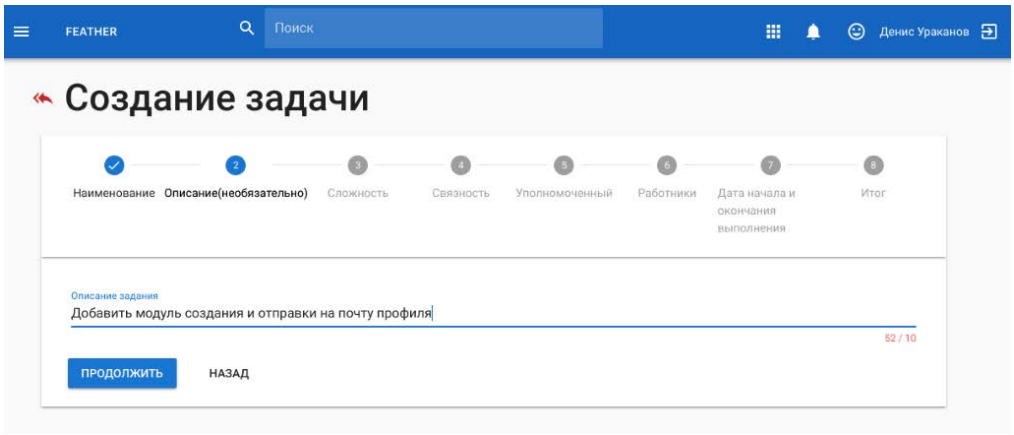

Рисунок 4.29 – Второй шаг создания задачи

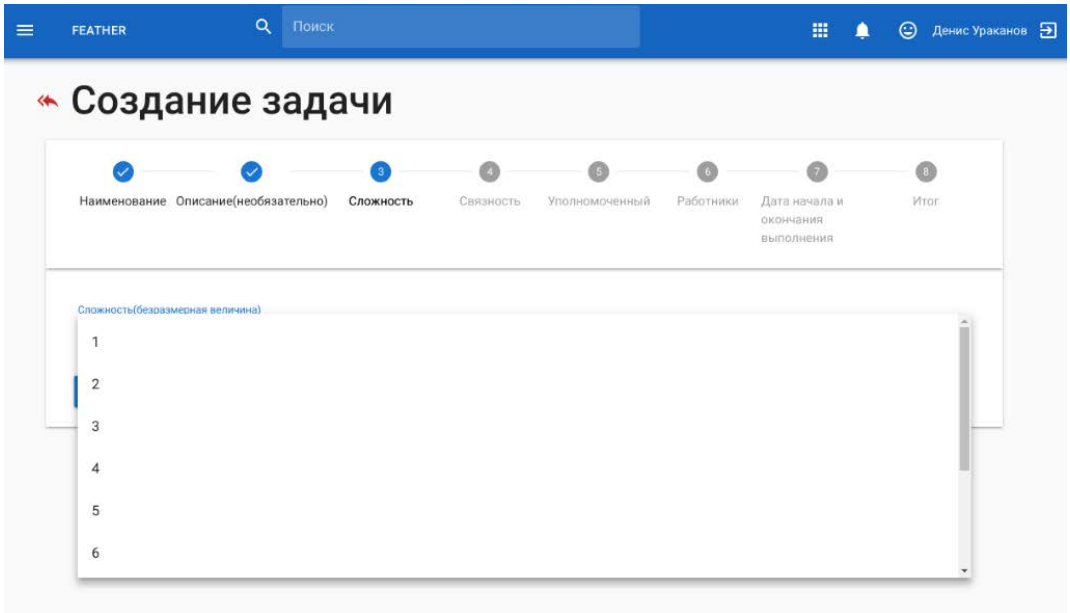

Рисунок 4.30 – Третий шаг создания задачи

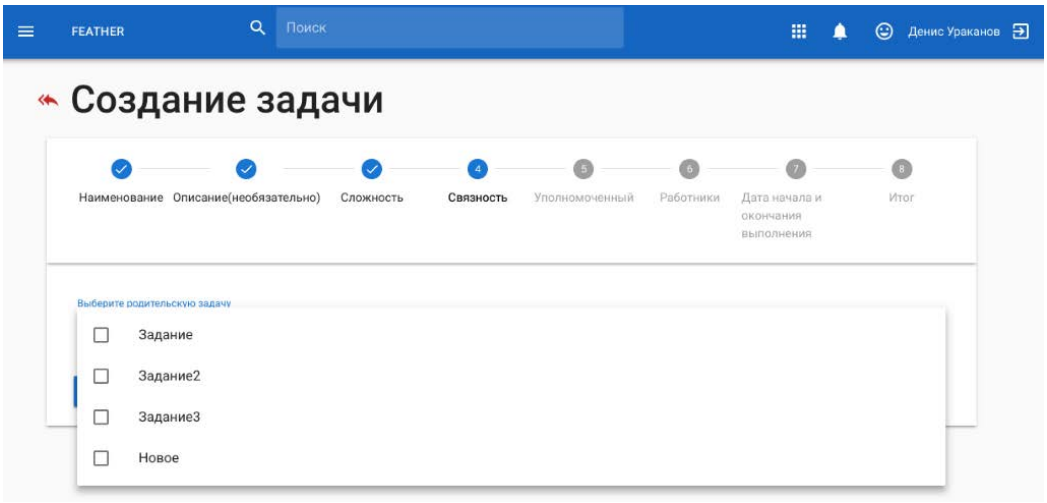

Рисунок 4.31 – Четвертый шаг создания задачи

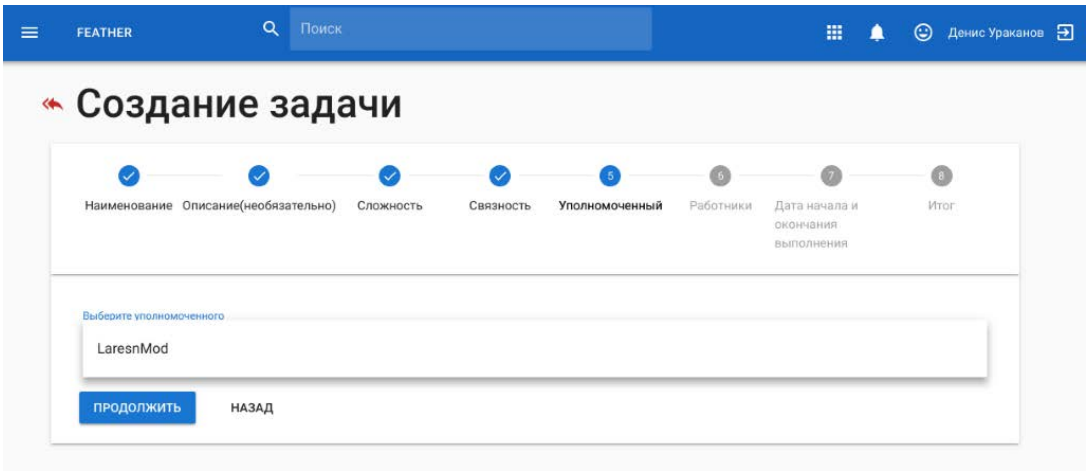

Рисунок 4.32 – Пятый шаг создания задачи

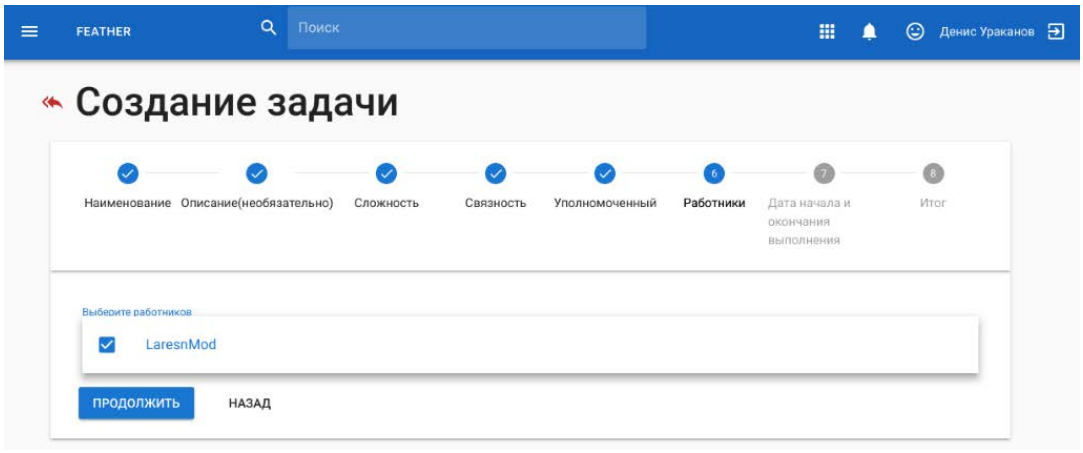

Рисунок 4.33 – Шестой шаг создания задачи

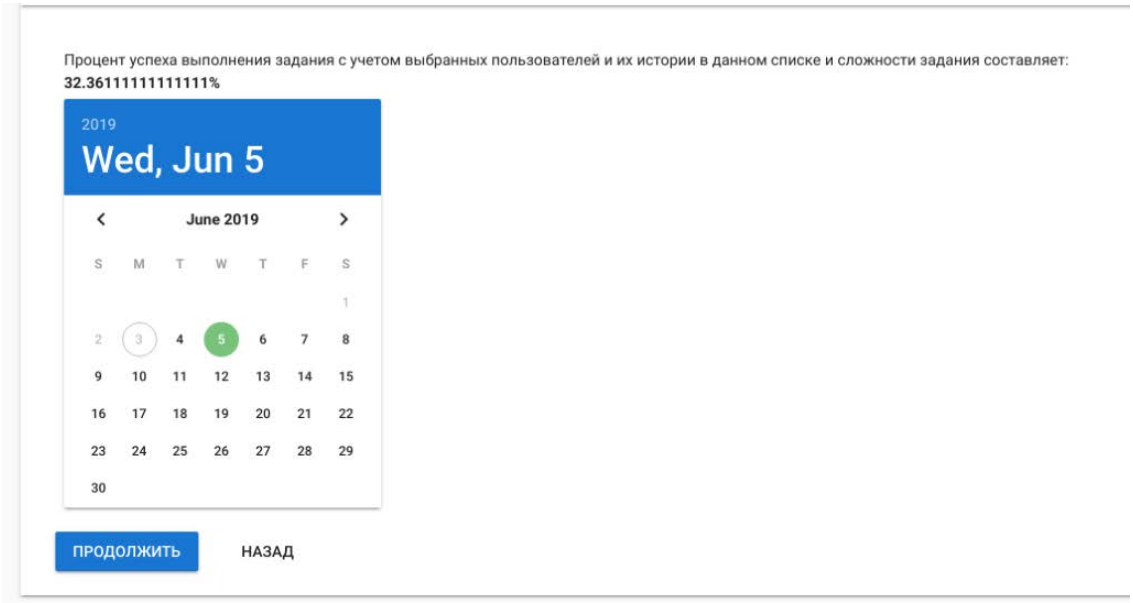

Рисунок 4.34 – Седьмой шаг создания задачи

Следует заметить, что на рисунке 4.34 сервис предоставляет оценку успешного выполнения данного задания при указанной сложности и выбранных пользователях.

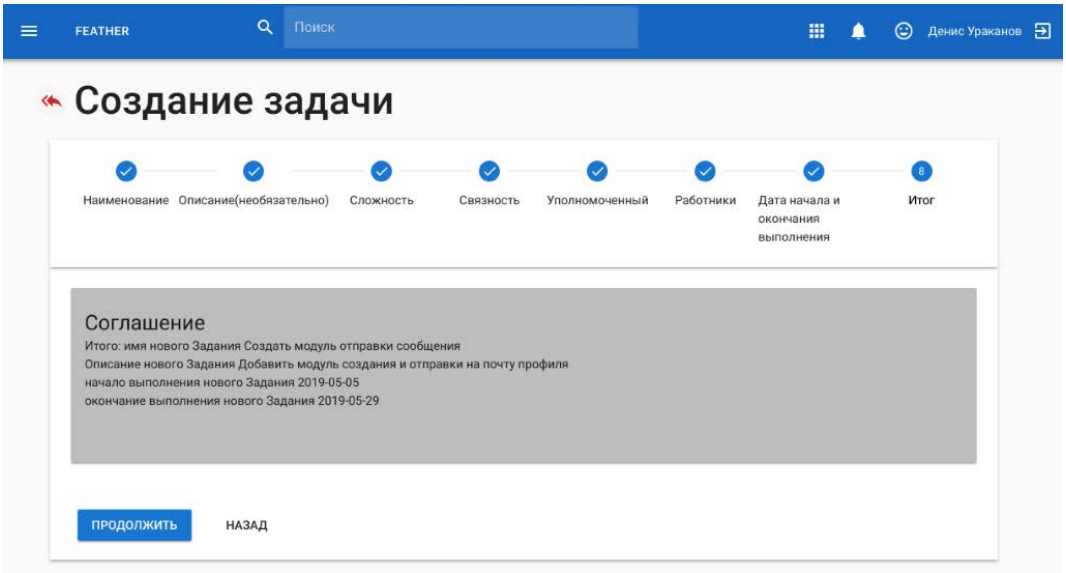

Рисунок 4.35 – Седьмой шаг создания задачи

После создания задачи пользователь переходит на страницу списка задач (рисунок 4.36).

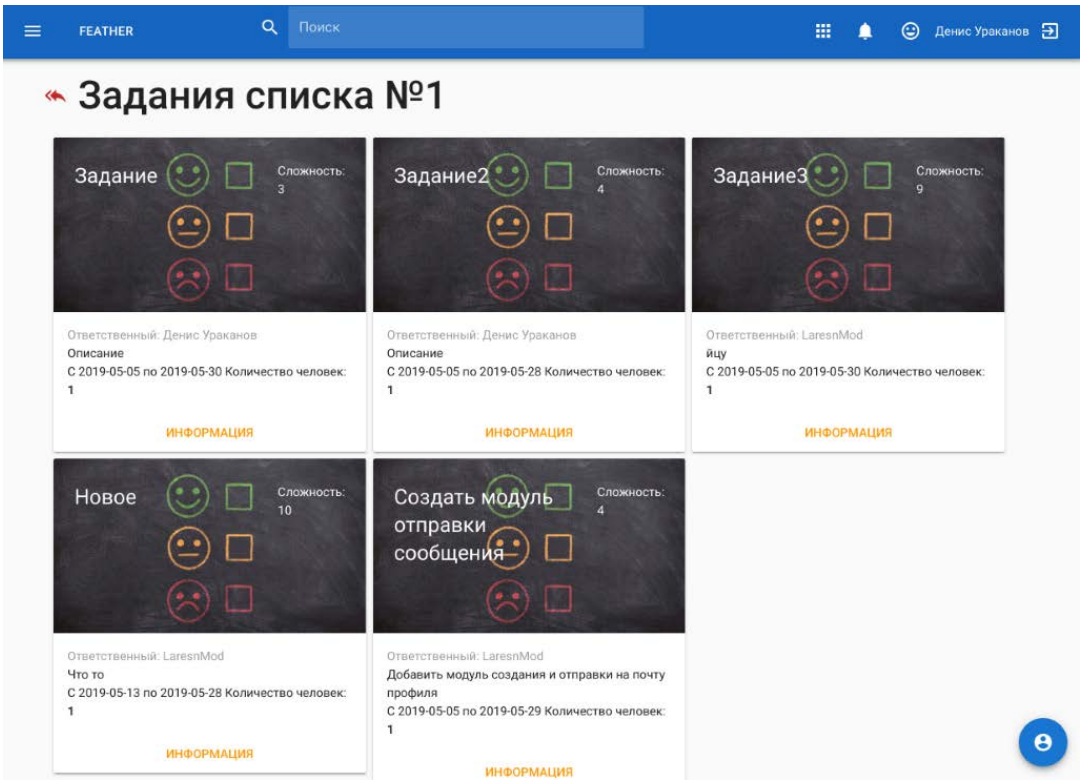

Рисунок 4.36 – Список задач

4.6. Графы задач

В приложении пользователю доступно представление всего проекта в виде графа, где узел графа – это задача, а ребра графа – это связи между задачами. Эти связи заключаются в том, что одна задача должна быть выполнена до другой. Таким образом получаются дочерние и родительские узлы.

Для отображения графа пользователю необходимо в списке проектов (рисунок 4.14) выбрать пункт «Граф». Пример отображения для данного проекта представлено на рисунок 4.37.

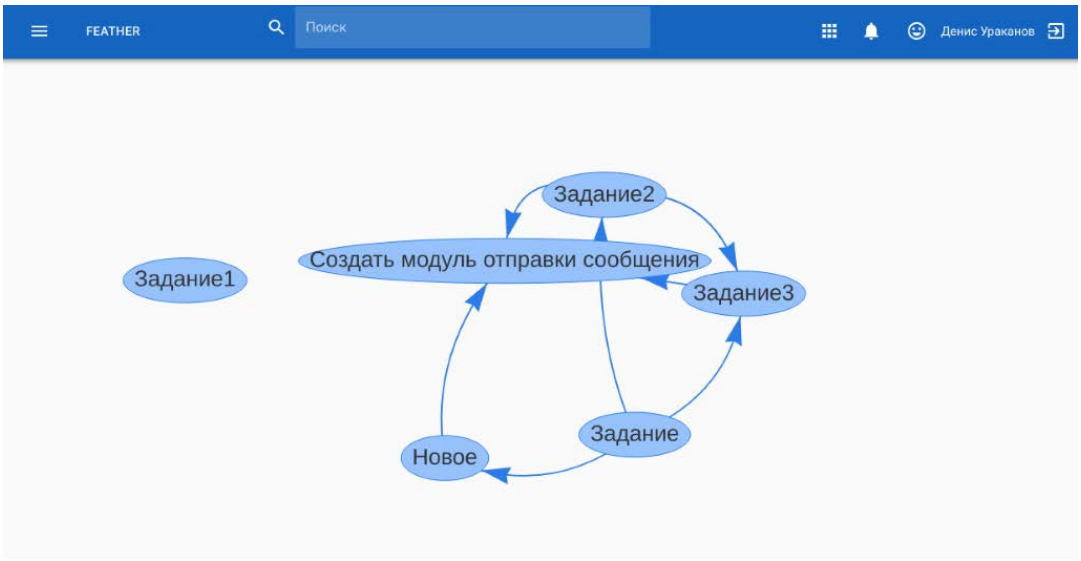

Рисунок 4.37 – Представление проекта в виде графа

Одной из важных отличительных черт данного проекта является помощь пользователю в поиске оптимального выполнения заданий в списке задач на основе сложности каждого и количества человек, заявленных на работу в данный список задач.

Для отображения оптимального выполнения списка задач необходимо выбрать пункт «Оптимальный обход» в необходимом списке задач в общем листе данной структуры (рисунок 4.15). Далее произойдет отображение списка заданий в виде графа, в котором каждый узел имеет подпись. Подпись отражает наименование задания, сложность задания и рекомендуемую нагрузку в виде количества людей на данную задачу. Причем потенциально одновременно выполняющиеся задания будут отображаться одинаковым цветом, а задачи с максимальной нагрузкой будут выделяться красной звездой. Пул раскраски узла состоит из 6 цветов. Был написан такой алгоритм, при котором после последнего выданного цвета из пула на следующей итерации будет предоставлен первый цвет (т.е. циклический подход). Данное решение создано исключительно для визуального разделения непараллельных близких по плоскости расположения задач. Пример такого графа представлен на рисунке 4.38.

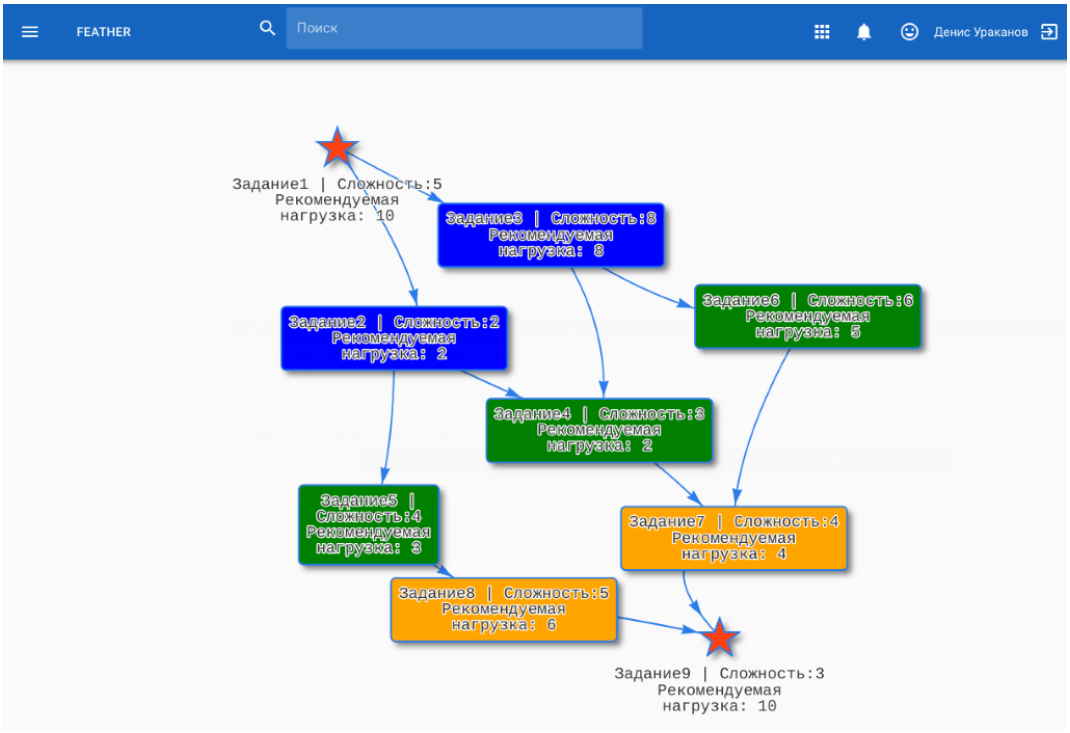

Рисунок 4.38 – Представление оптимального обхода списка задач

4.7. Слияние проектов

В приложении пользователю доступна возможность слияние одного или нескольких проектов определенной группы, в которой он имеет роль “ADMIN”, с проектом другой группы.

Для подачи заявки на слияние первому пользователю необходимо перейти в раздел «Проекты сообщества» и напротив определенного проекта может быть доступна кнопка «Слияние». Данная опция не доступна, если проект является закрытым и владелец не желает контактов с другими участниками сообщества (рисунок 4.39).

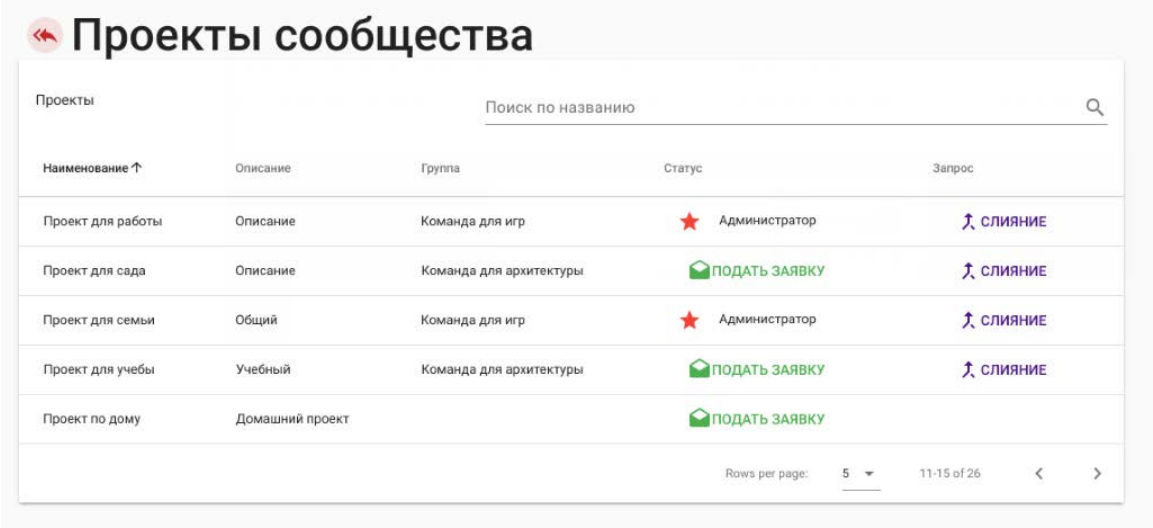

Рисунок 4.39 – Проекты сообщества

После нажатия на кнопку «Слияние» напротив проекта «Проект для учебы» появится диалоговое окно с формой заявки. В данной заявке пользователь может выбрать группу, в которой он имеет роль “ADMIN”, и выбрать проект(ы), которые он желает слить с выбранным в списке сообщества (рисунок 4.40).

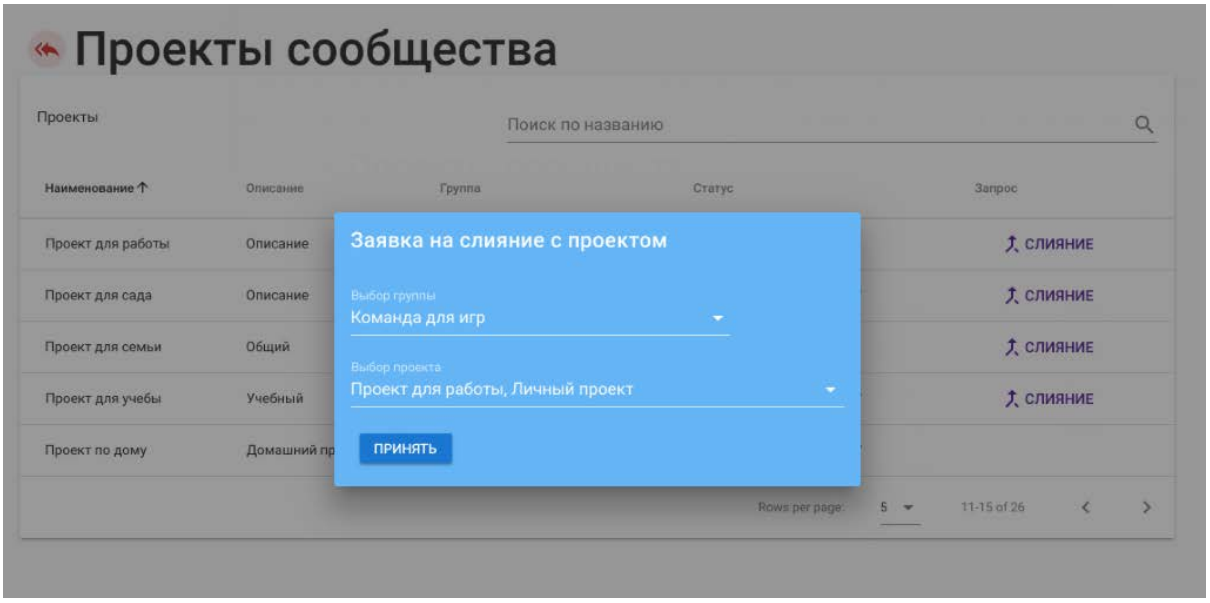

Рисунок 4.40 – Заявка на слияние

На рисунке 4.41 представлен граф проекта первого пользователя «Проект для учебы».

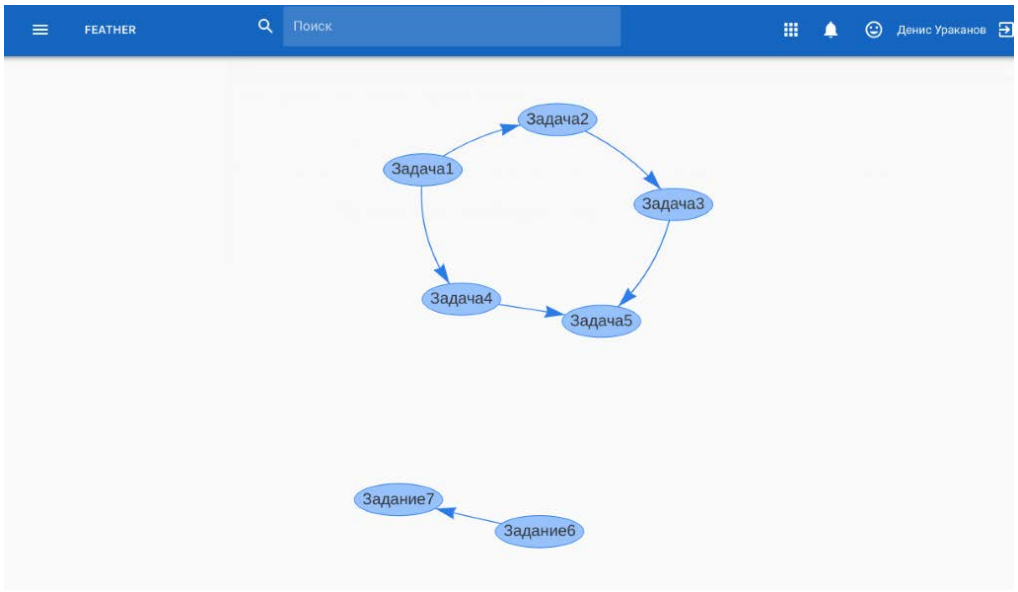

Рисунок 4.41 – Граф задач «Проект для учебы».

На рисунке 4.42 представлен график проекта второго пользователя «Проект для работы».

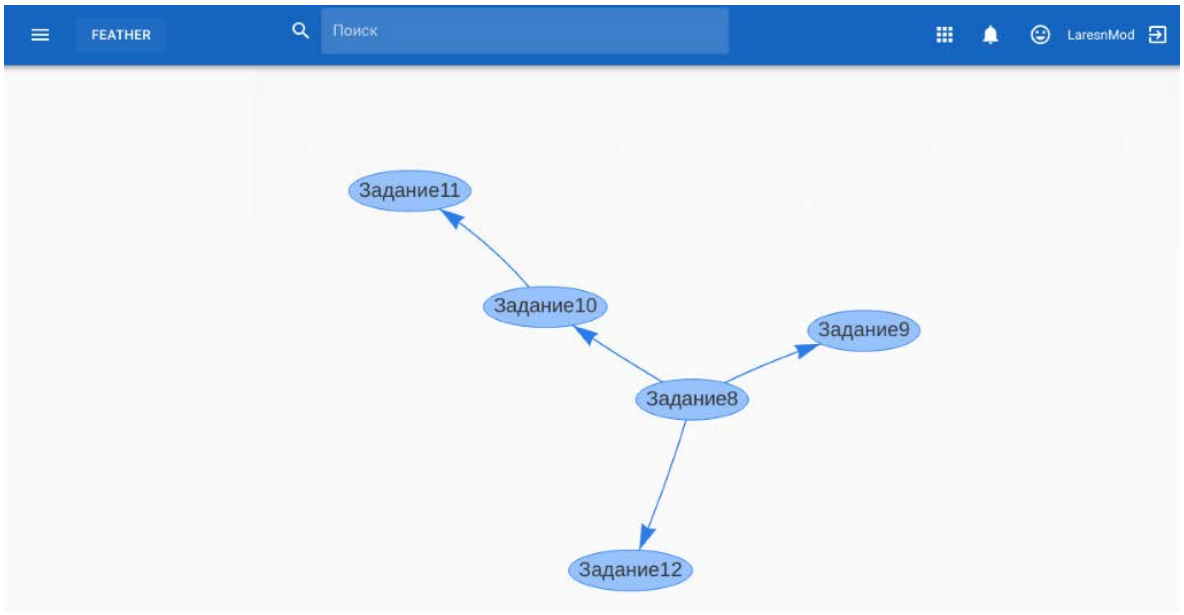

Рисунок 4.42 – Граф задач «Проект для работы»

На рисунке 4.43 представлен график проекта второго пользователя «Личный проект».

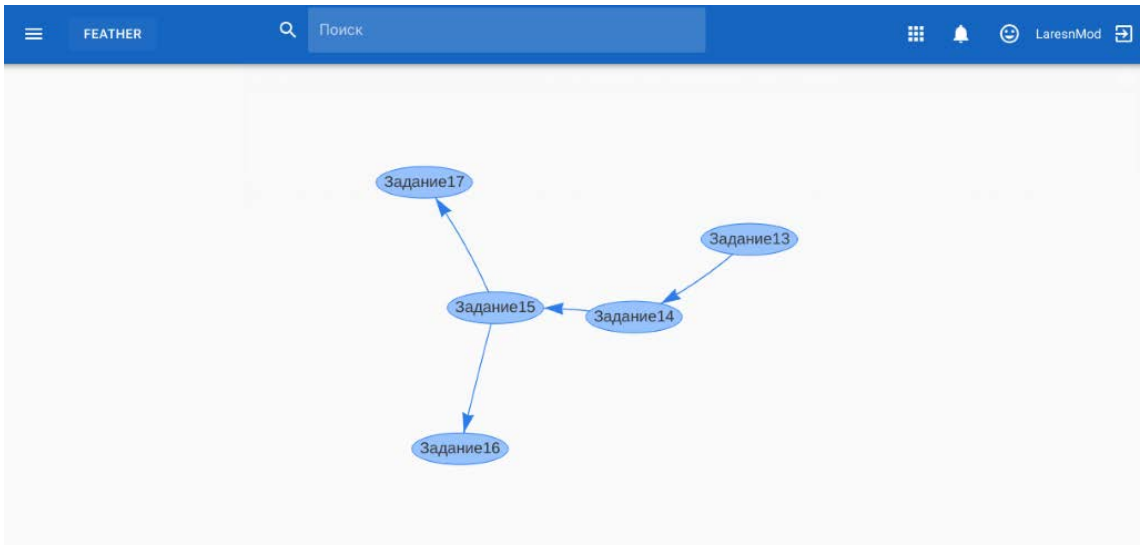

Рисунок 4.43 – Граф задач «Личный проект»

После нажатия на кнопку «Принять» в форме заявки у первого пользователя появляется заявка на слияние с проектов его группы в форме редактирования и информации данной группы (рисунок 4.44).

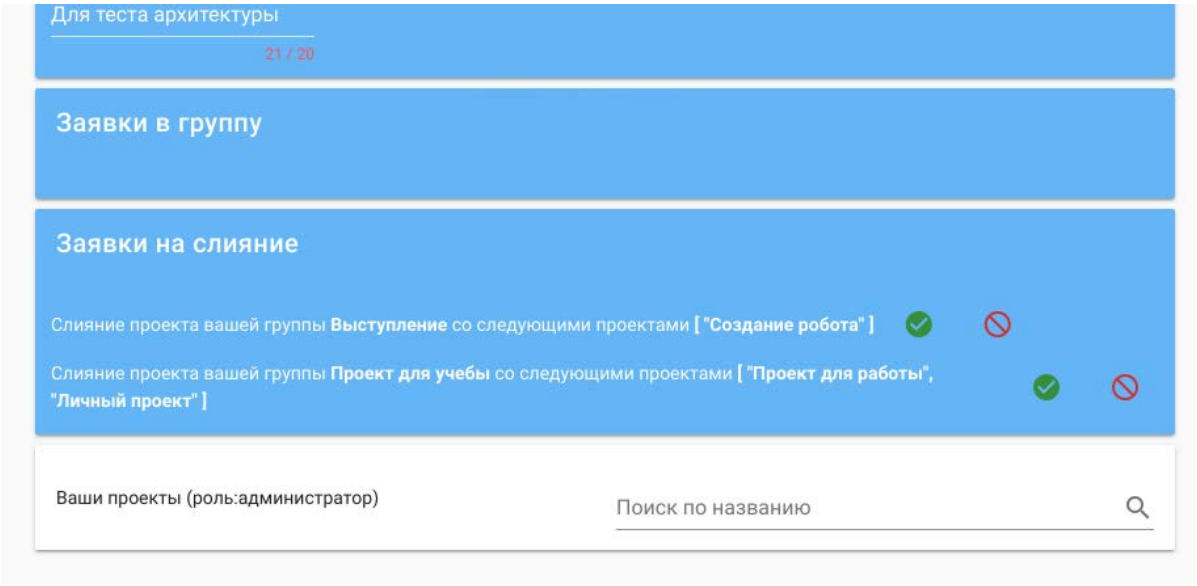

Рисунок 4.44 – Заявка на слияние

В пункте «Заявки на слияние» появилась новая запись. Данное предложение первый пользователь в праве принять или отклонить. Если он примет первое решение, то данные проекты сольются. Все списки задач и задачи будут автоматически привязаны к новому проекту, все пользователи имевшие опреде-

ленные права на бывших проектах будут иметь те же и в новом. Результат слияние представлена на рисунке 4.45.

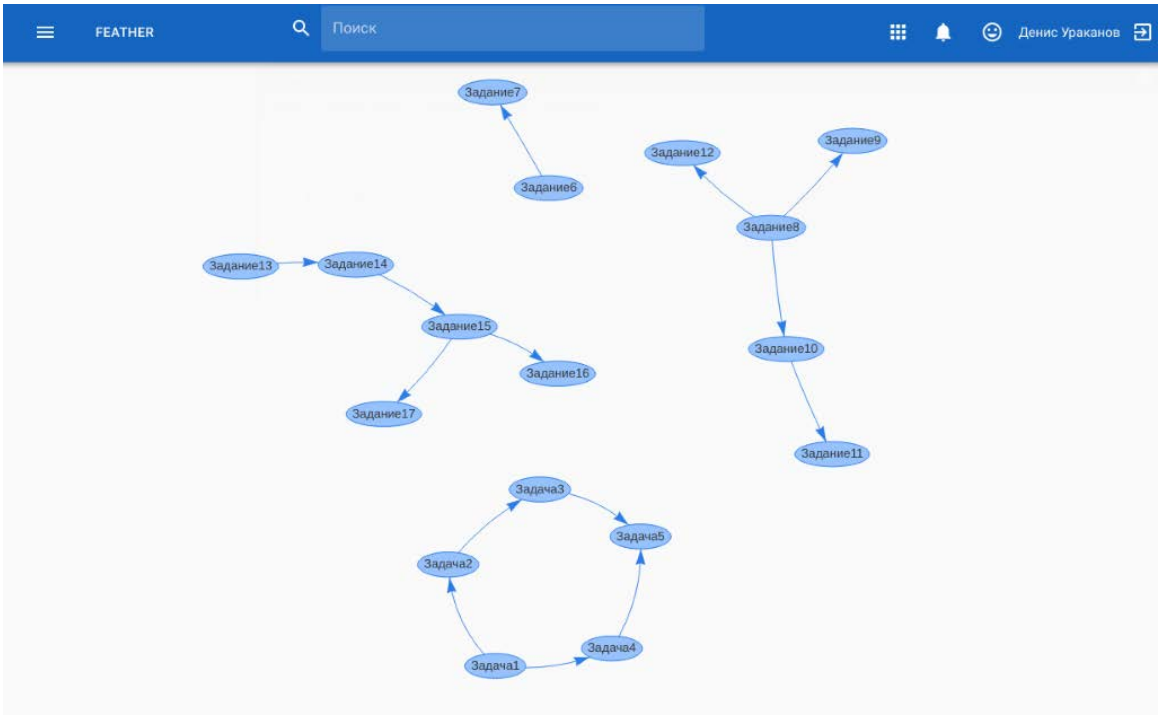

Рисунок 4.45 – Граф задач проекта слияния

По графу видно, что в трех бывших проектах было 4 списка задач. После слияния они все появились в новом проекте (рисунок 4.46).

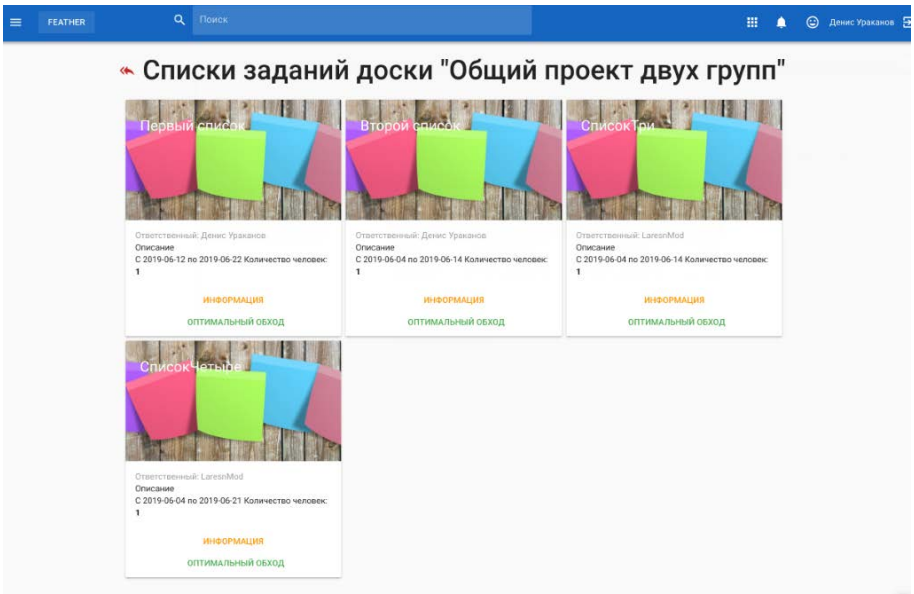

Рисунок 4.46 – Списки заданий проекта слияния

После слияния пользователь также может просмотреть оптимальное выполнение задач с точки зрения равномерного распределения людей на них. Результат данной операции над последним созданным слиянием проектом представлена на рисунке 4.47.

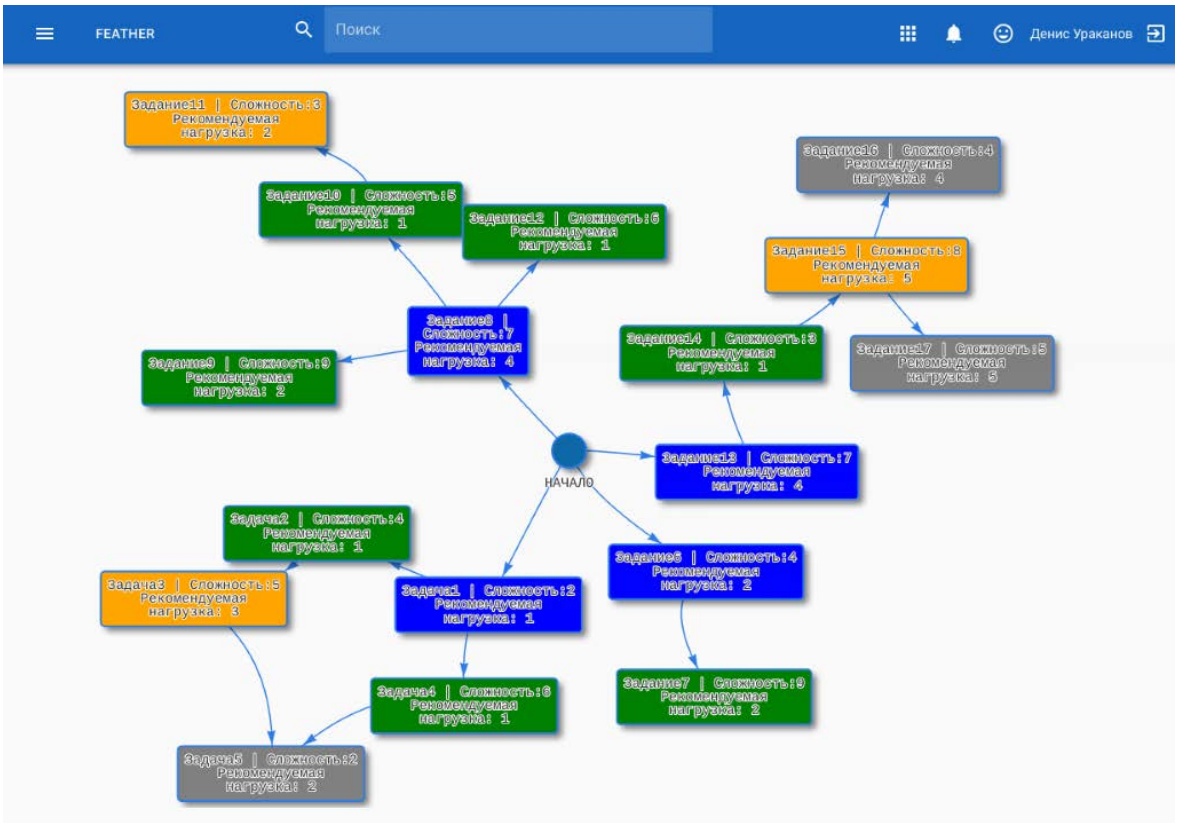

Рисунок 4.47 – Оптимальное выполнение задач после слияния проектов

4.8. Редактирование информации о проекте или списке задач

Пользователь, имеющий в данном проекте или списке задач роль “ADMIN”, имеет возможность изменять основную информацию и свойства структуры.

В свойствах проекта, которые открываются через вкладку «Информация» определенного проекта в списке проектов можно изменить наименование про-

екта, его описание, а также выполнять действия по работе с заявками в проект. Пример редактирования представлен на рисунке 4.48.

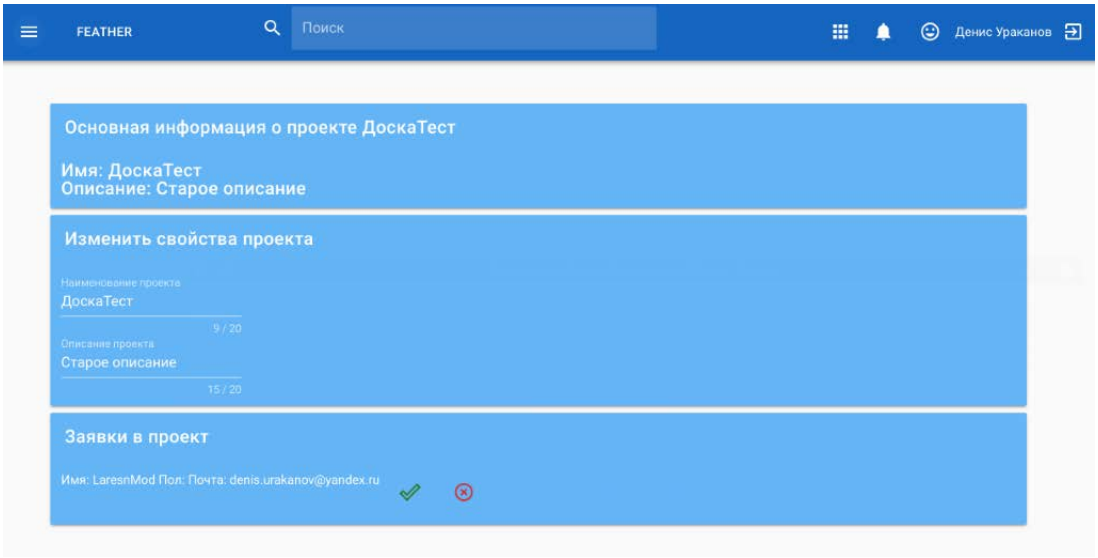

Рисунок 4.48 – Редактирование свойств проекта

В окне свойств списка задач, которое открывается через вкладку «Информация» определенного списка в листе списков можно изменить наименование списка, его описание, дату начала и окончания выполнения, а также изменить ответственного исполнителя. Пример редактирования представлен на рисунке 4.49.

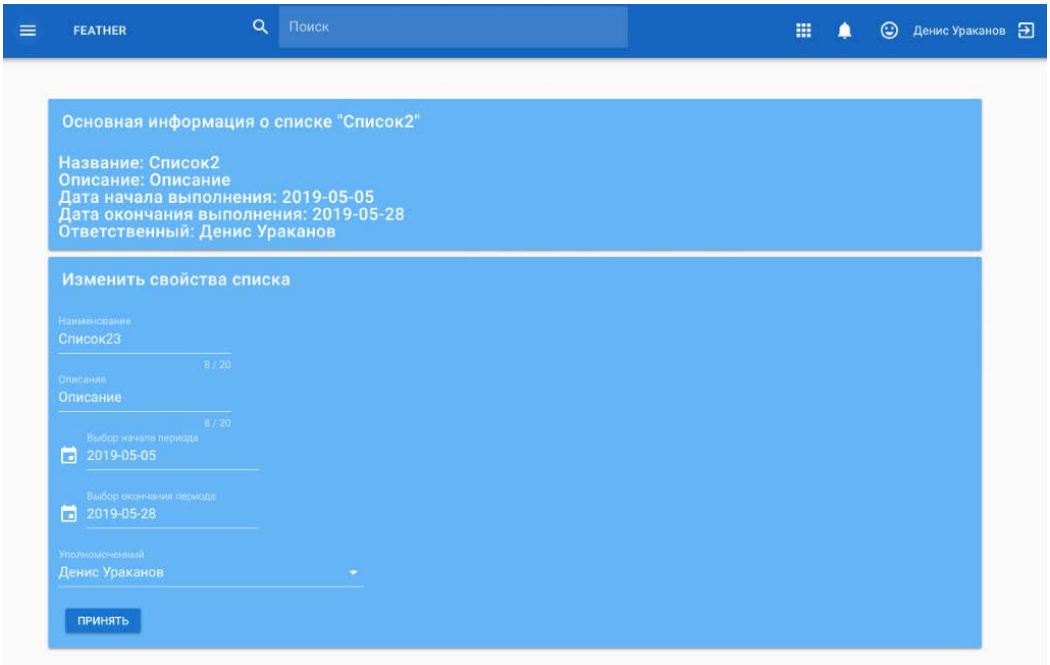

Рисунок 4.49 – Редактирование свойств списка задач

4.9. Работа с командами

Каждый авторизированный пользователь и пользователь, одобренный модератором, имеет возможность создавать группы или быть принятым в группу. Переход на страницу создания команды через меню Сообщества проекта можно перейти через главное меню (рисунок 4.6), а затем выбрать «Создание команды» (рисунок 4.50).

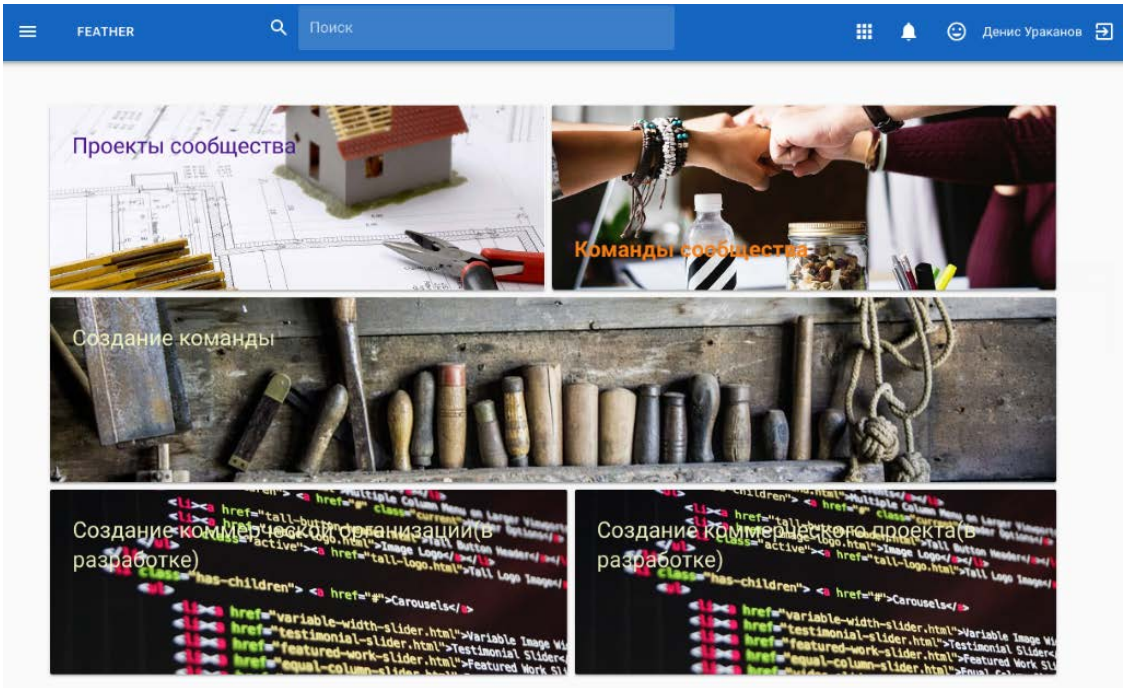

Рисунок 4.50 – Меню сообщества

На рисунках 4.51 – 4.54 представлены шаги создания группы.

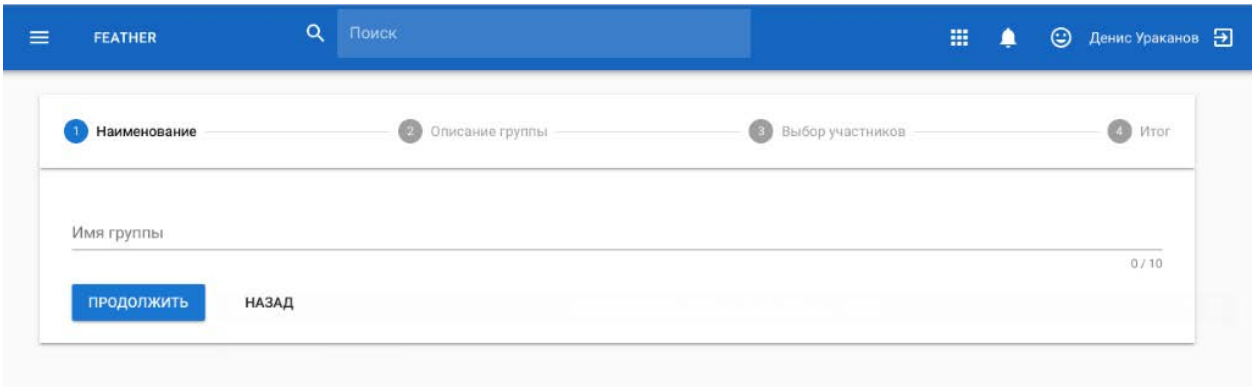

Рисунок 4.51 – Первый шаг создания группы

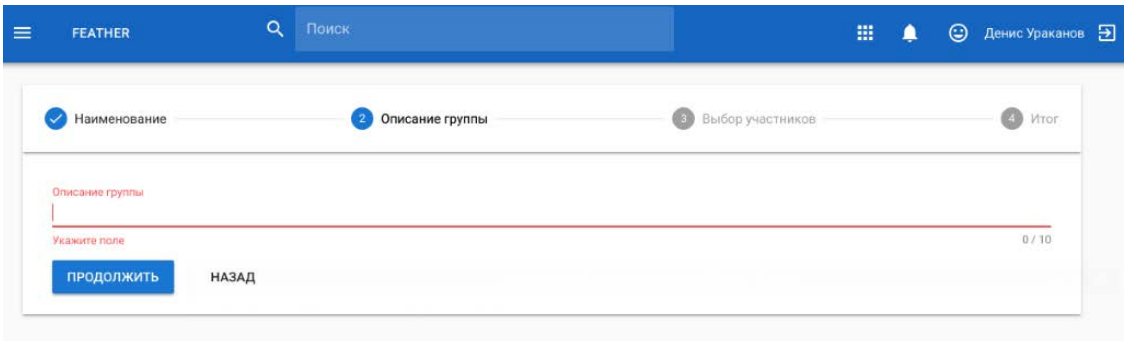

Рисунок 4.52 – Второй шаг создания группы

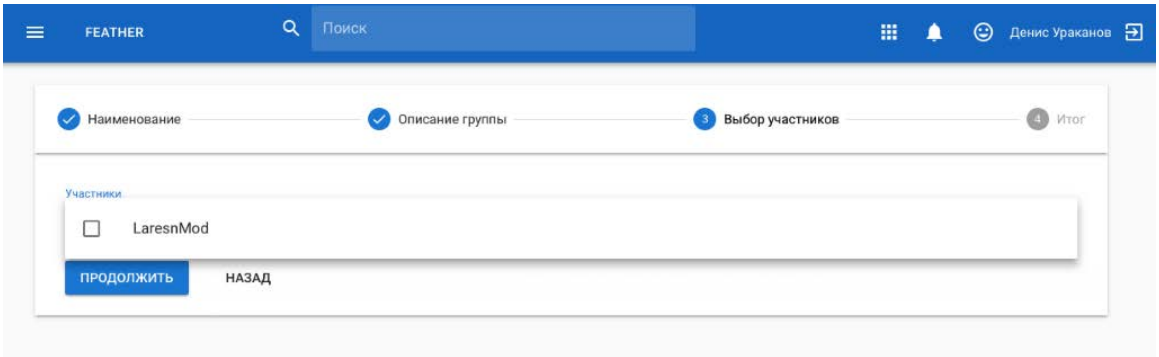

Рисунок 4.53 – Третий шаг создания группы

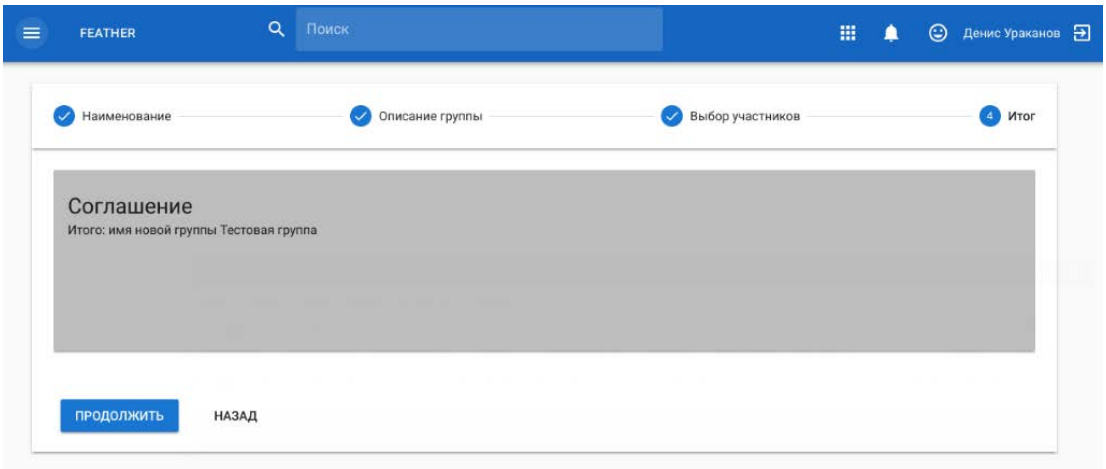

Рисунок 4.54 – Четвертый шаг создания группы

Сразу после создания группы пользователь попадает в список групп, связанных с данным пользователем. Также переход на страницу списка групп возможен через главное меню (рисунок 4.6) и вкладку «Мои группы». Страница со списком групп данного пользователя представлена на рисунок 4.55.

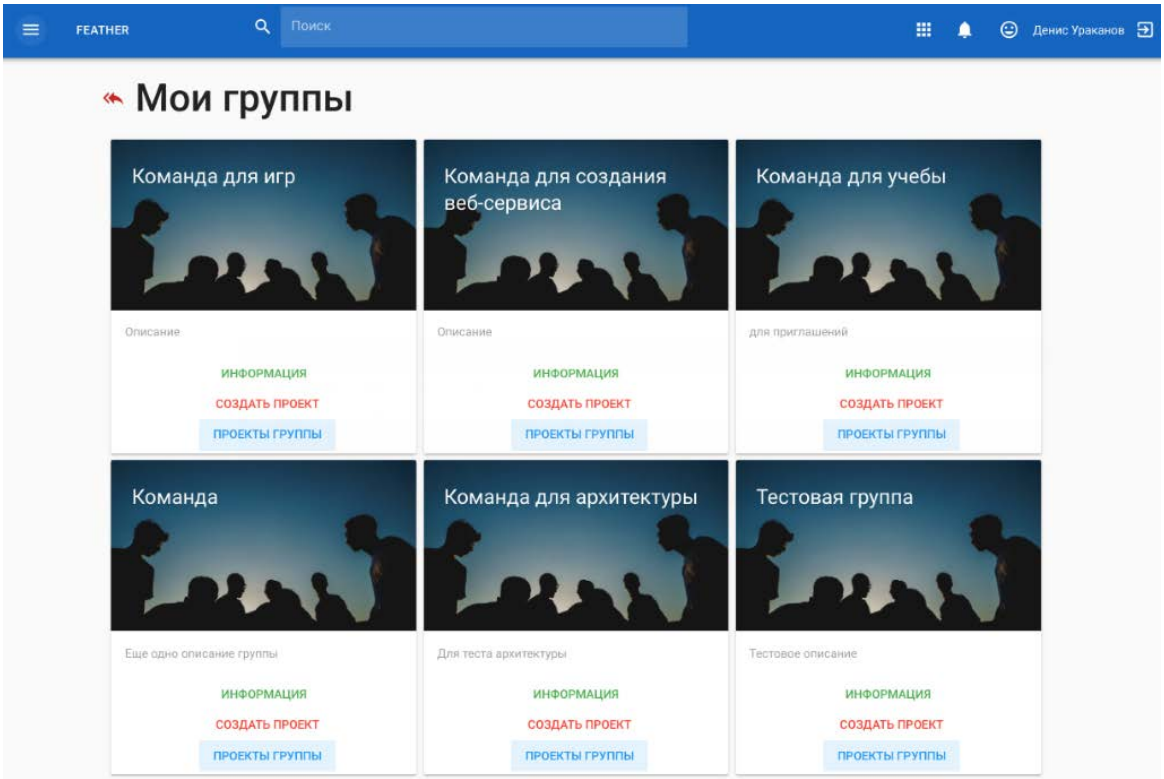

Рисунок 4.55 – Список групп пользователя

При нажатии на кнопку «Информация» в заданной группе этого списка пользователь перейдет на страницу изменения свойства данной группы. Если пользователь данной группы имеет роль “ADMIN”, то он в состоянии изменить наименование группы и её описание, а также принять заявки в группу или совершить отказ (рисунок 4.56).

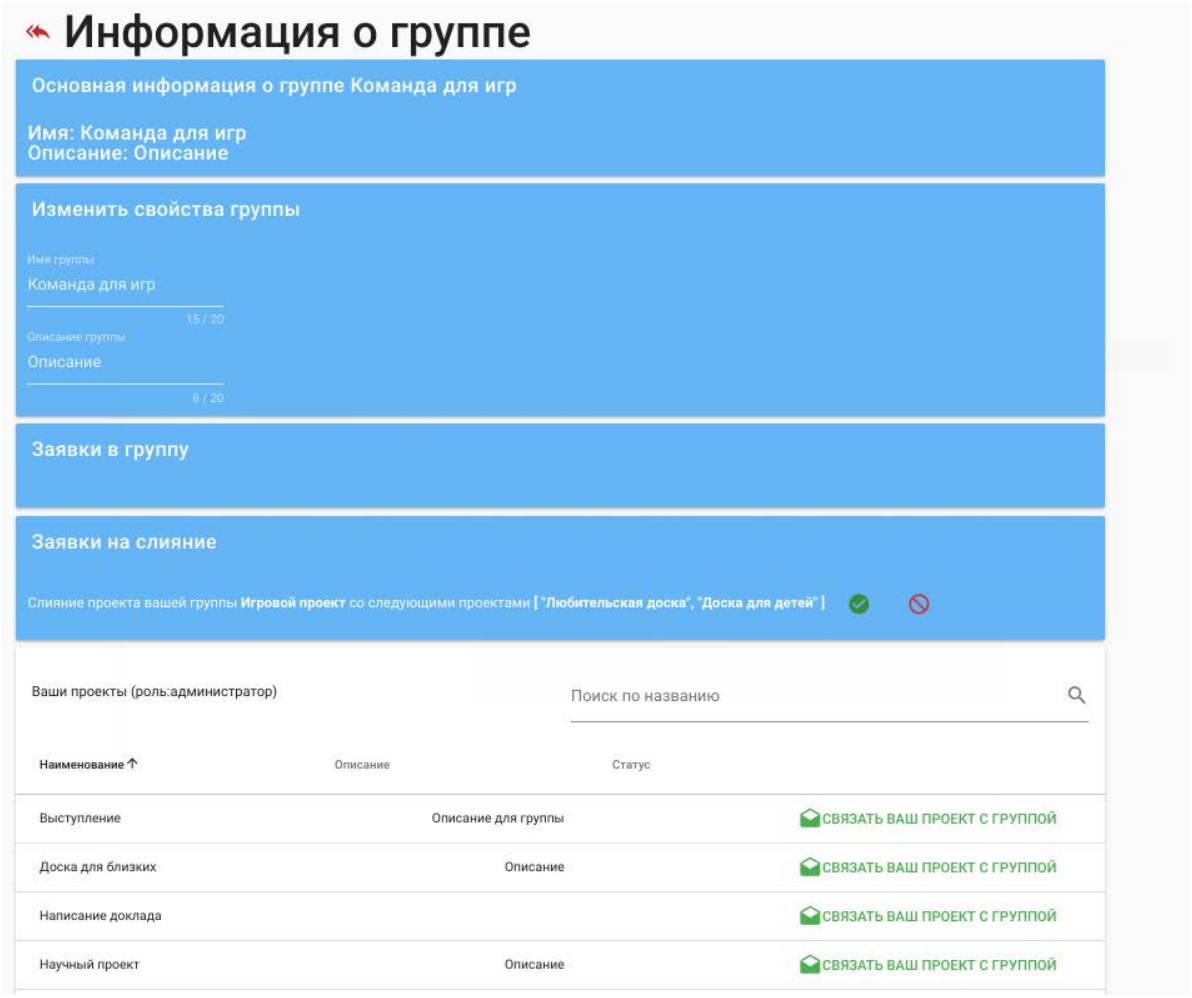

Рисунок 4.56 – Редактирование свойств группы

Также для удобства пользователя, если он имеет роль “ADMIN” в группе, в редактировании которого в данный момент находится, и в определенных проектах, то на данной странице (рисунок 4.56) появиться список с данными проектами, которые являются частными для данного пользователя, но имеют возможность быть привязанными к данной группе. При нажатии на соответствующую кнопку проект будет привязан к группе (рисунок 4.57).

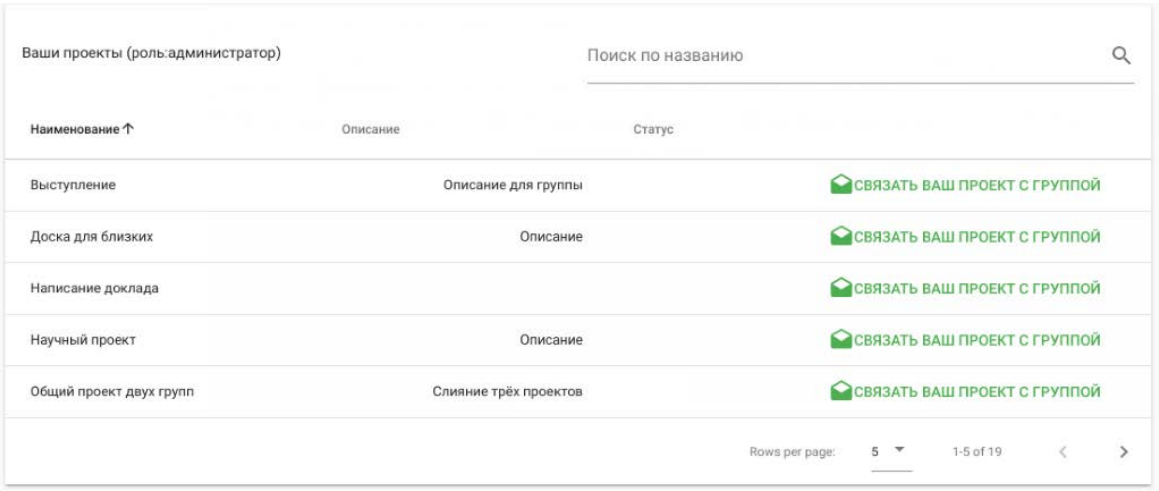

Рисунок 4.57 – Привязка проекта к группе

4.10. Проекты и группы сообщества

Одной из особенностей проекта является то, что сервис не только предоставляет пользователю возможность работать со своими проектами или командами, созданными им же, но и взаимодействовать с другими. Для того, чтобы просмотреть все проекты, необходимо перейти по вкладке «Проекты сообщества» в меню сообщества (рисунок 4.50).

Страница с проектами сообщества представлена на рисунок 4.58 - 4.59.

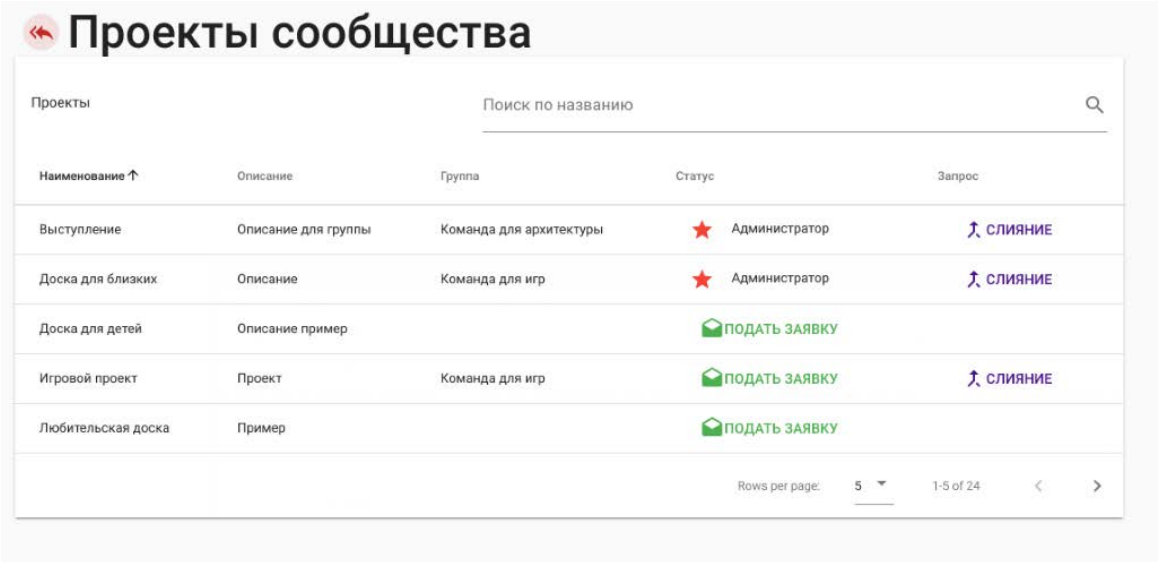

Рисунок 4.58 – Проекты сообщества

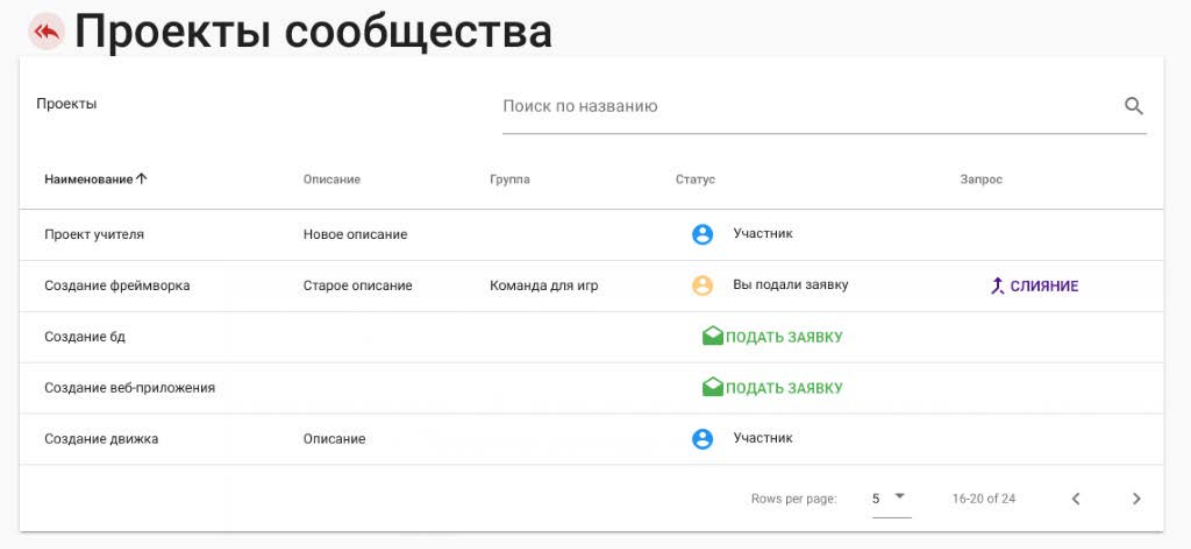

Рисунок 4.59 – Проекты сообщества

В строке с наименованием и описанием каждого проекта будет указан статус пользователя по отношению к данному проекту: «участник», «вы подали заявку», «администратор» и «Подать заявку». Если пользователь никаким образом не связан с данным проектом, то при нажатии на «Подать заявку», его статус сразу же меняется на «вы подали заявку». Если заявка была принята, то статус меняется на «участник». Также если пользователь является администратором, то соответствующий статус будет отображен напротив имени проекта.

Данную таблицу можно фильтровать по наименованию проекта или описанию или сортировать по столбцам (рисунок 4.60).

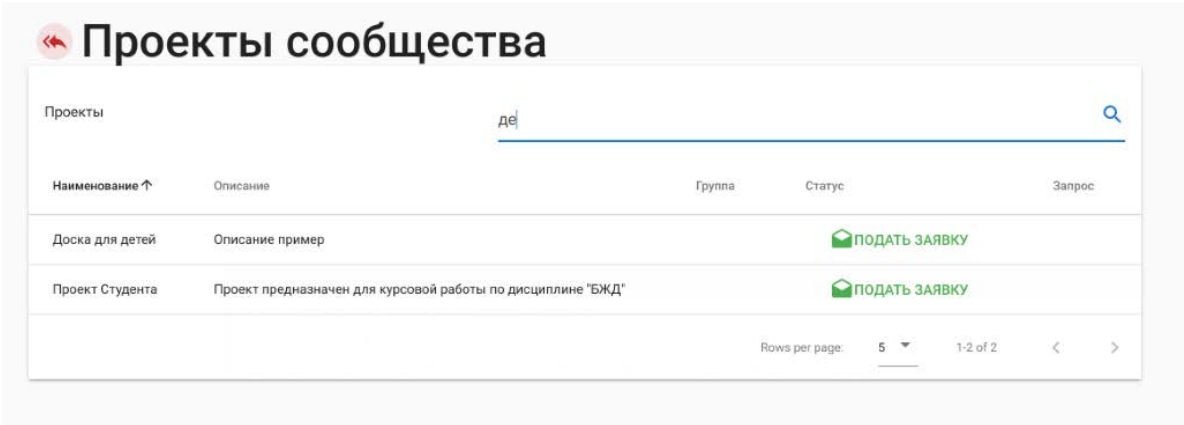

Рисунок 4.60 – Фильтр и сортировка по столбцу

Пользователь может просматривать информацию и о других группах. Перейти на страницу сообщества групп можно в меню сообщества (рисунок 4.50).

Пример страницы с группами сообщества представлено на рисунке 4.61.

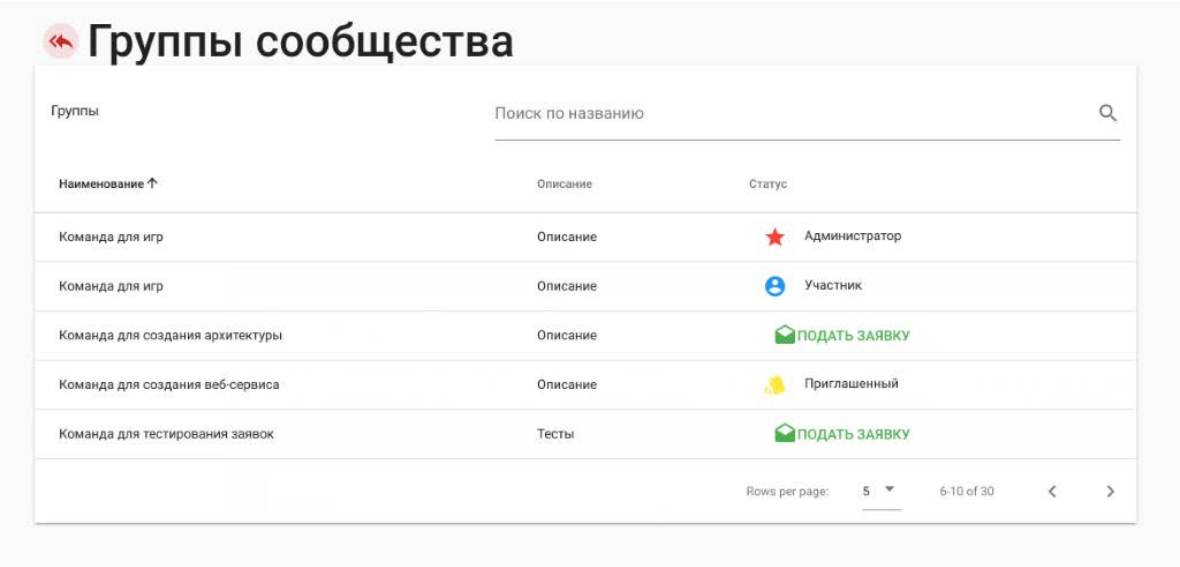

Рисунок 4.61 – Группы сообщества

С строке с наименованием и описанием группы располагается статус пользователя к данной группе. Статус может быть: «участник», «приглашенный», «Вы подали заявку», «Администратор» и «Подать заявку». Как и в случае с проектами, если пользователь никак не связан с группой, будет предложено подать заявку. В дальнейшем статус пользователя будет «Вы подали заявку». После принятия администратором группы заявки пользователь будет являться участником группы.

При создании группы пользователю предлагается выбор участников. Все приглашенные участники данной группы увидят свой статус также в этой таблице. Принятие приглашения осуществляется в личном кабинете пользователя.

Пользователь также может фильтровать таблицу и сортировать её. Пример фильтрации и сортировки представлено на рисунке 4.62.

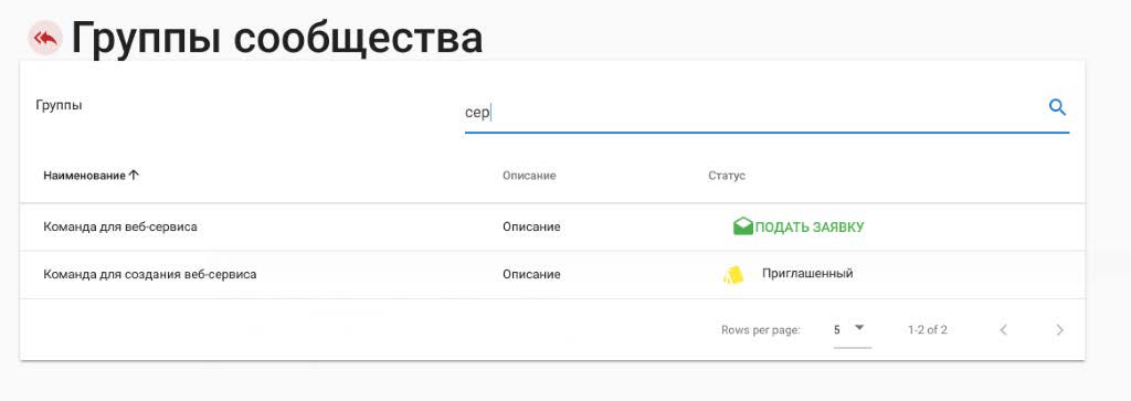

Рисунок 4.62 – Фильтрация и сортировка таблицы групп сообщества

5. ТЕСТИРОВАНИЕ

5.1. Тестирование форм ввода

Проект разрабатывался исключительно в рамках дипломного проектирования. Поэтому было проведено только альфа-тестирование продукта. Для нахождения багов и предотвращения нежелательных событий при работе программы были проведены модульные тесты.

1. Тестирование форм ввода создании проекта

На рисунках 5.1 – 5.3 представлены проверки при неправильно введенных для сервиса значениях.

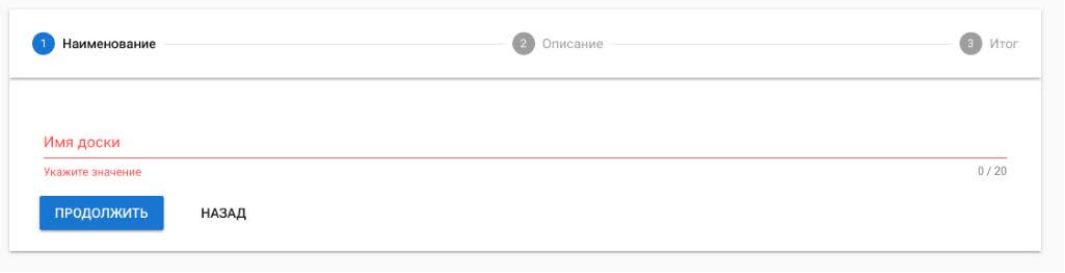

Рисунок 5.1 – Валидация формы ввода имени проекта

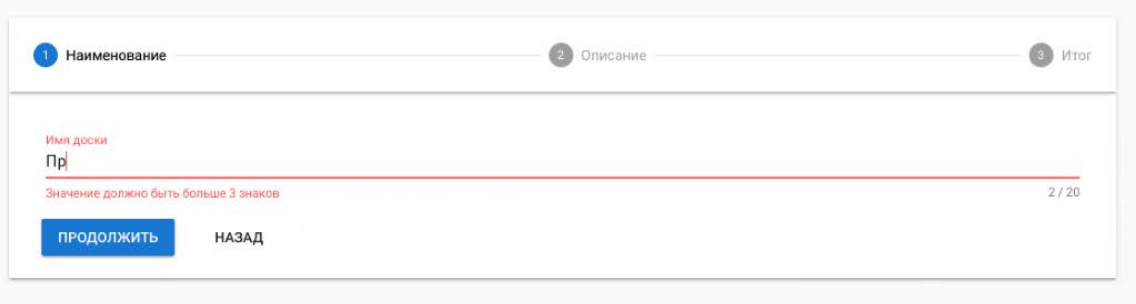

Рисунок 5.2 – Валидация формы ввода имени проекта

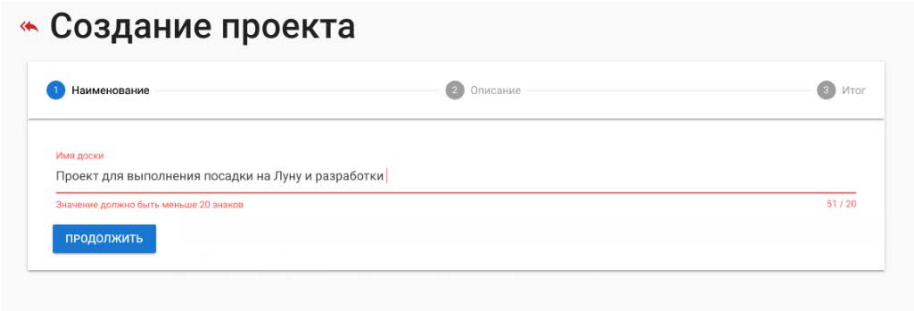

Рисунок 5.3 – Валидация формы ввода имени проекта

2. Тестирование форм ввода создания списка задач

На рисунках 5.4. – 5.6 представлены проверки при неправильно введенных для сервиса значений для списка задач.

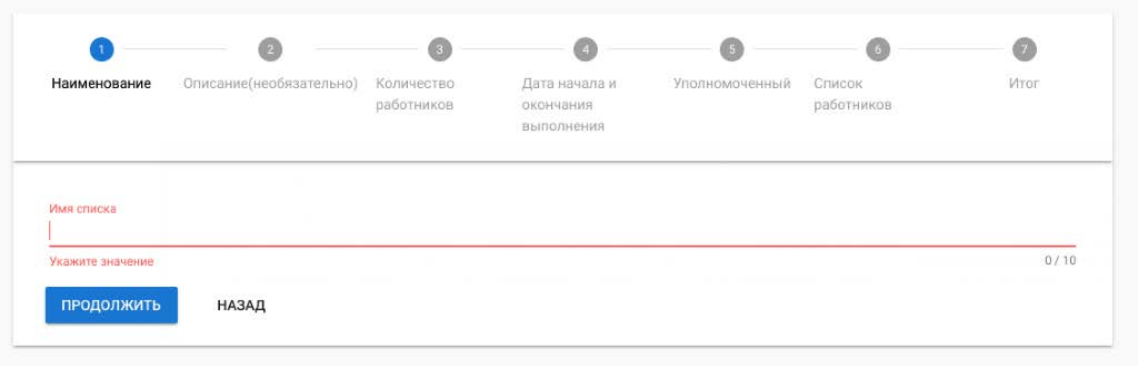

Рисунок 5.4 – Валидация формы ввода имени списка

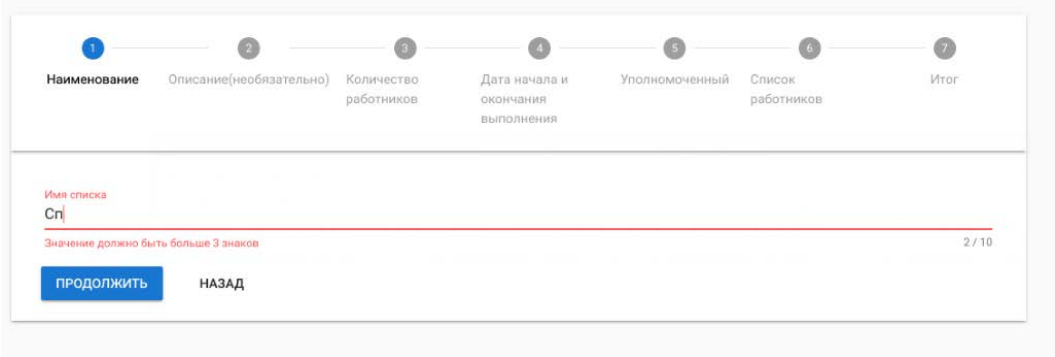

Рисунок 5.5 – Валидация формы ввода имени списка

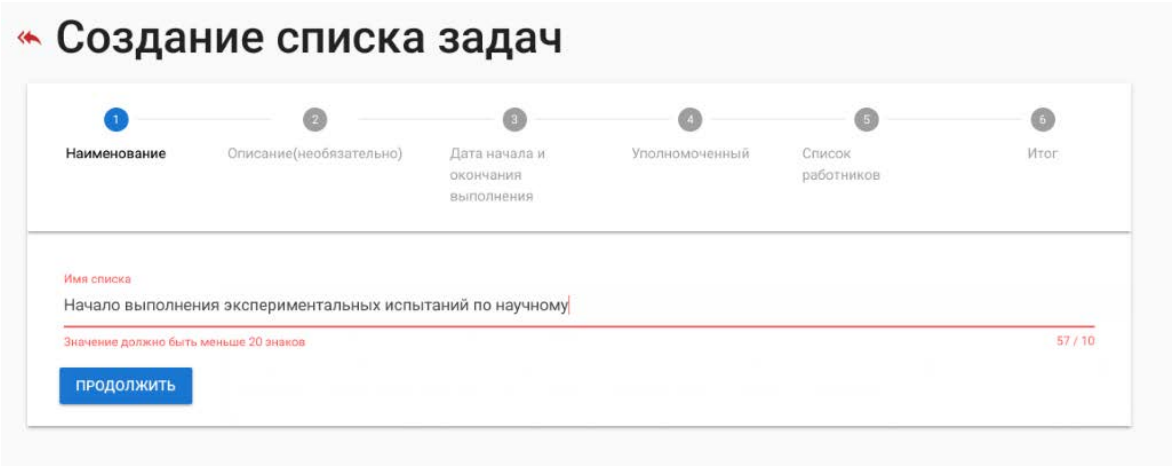

Рисунок 5.6 – Валидация формы ввода имени списка

Валидация формы ввода периода выполнения не предполагает даты раньше, чем текущая дата пользователя, а также форма не предоставляет возможности сделать дату окончания меньше, чем дата начала.

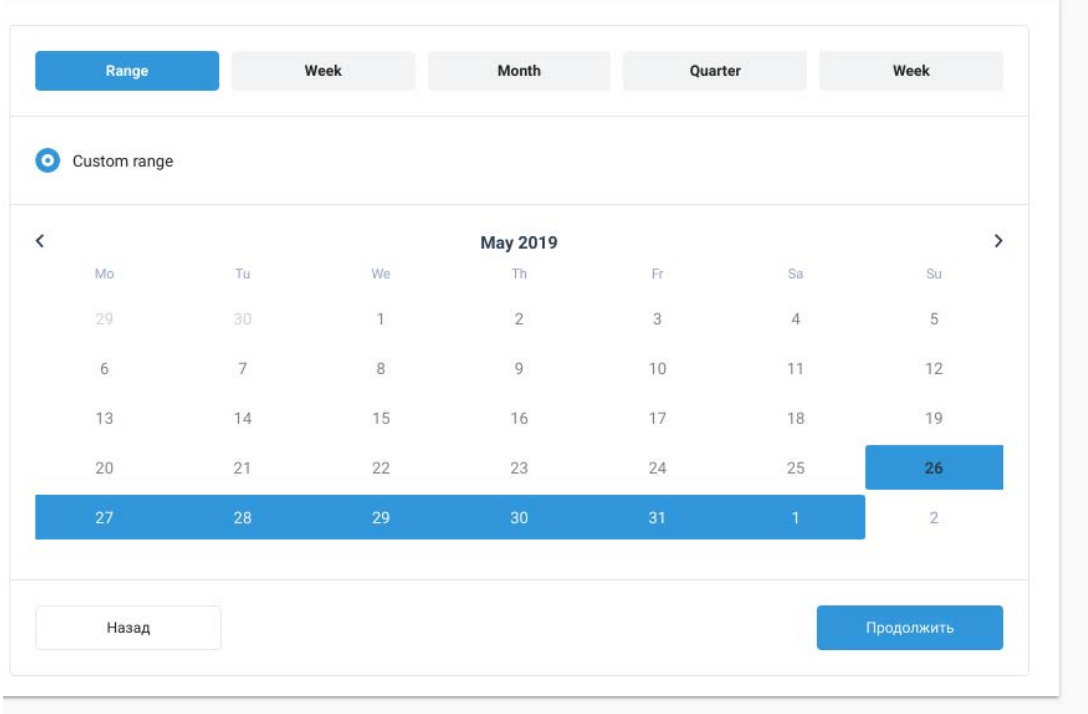

Рисунок 5.7 – Валидация формы ввода периода выполнения

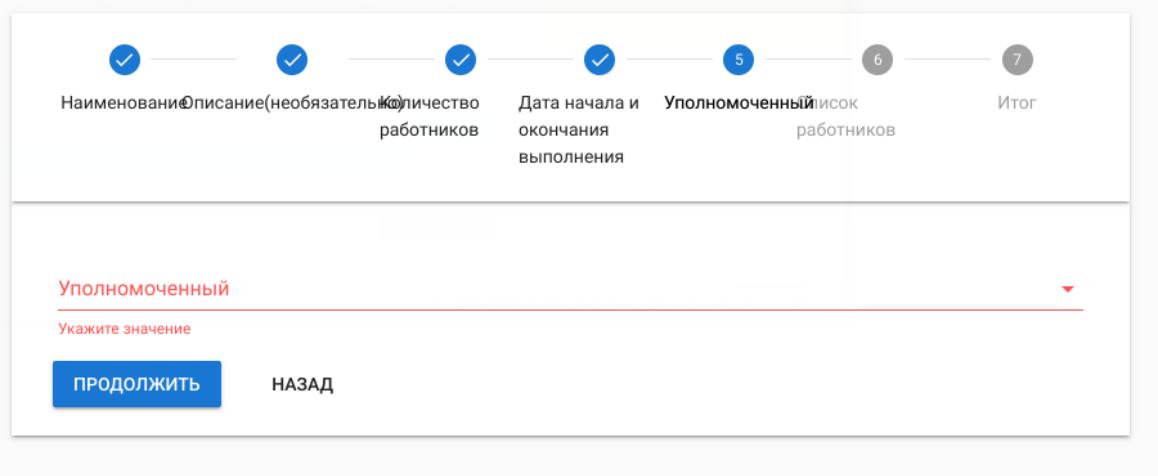

Рисунок 5.8 – Проверка формы выбора уполномоченного при пустом значении

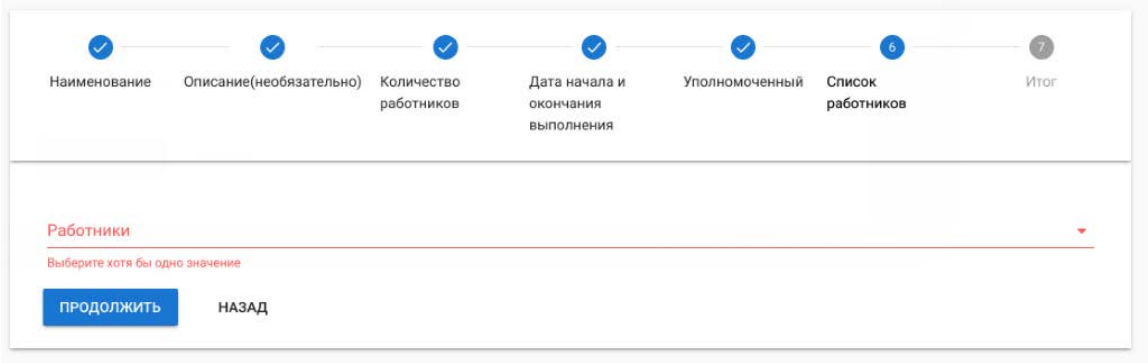

Рисунок 5.9 – Проверка формы выбора работников

3. Тестирование форм создания списка

На рисунках 5.10 – 5.15 представлены проверки заполнения форм создания задания пользователем.

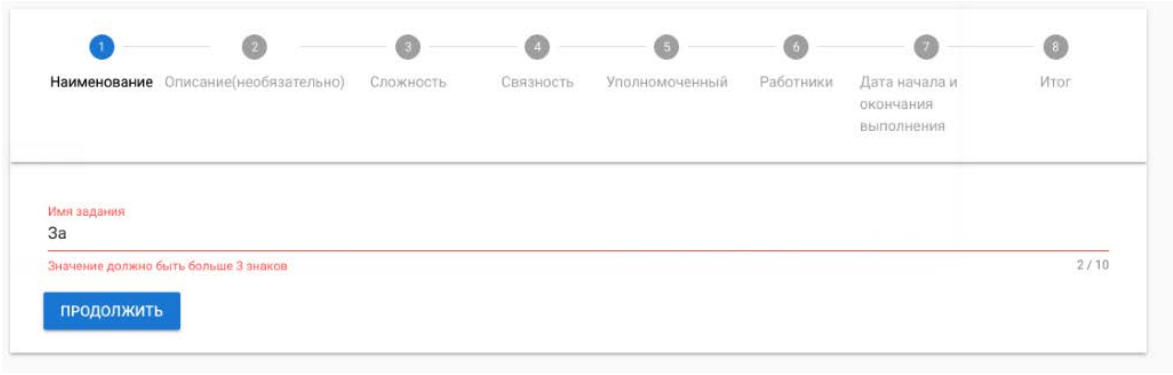

Рисунок 5.10 – Проверка формы наименования задания

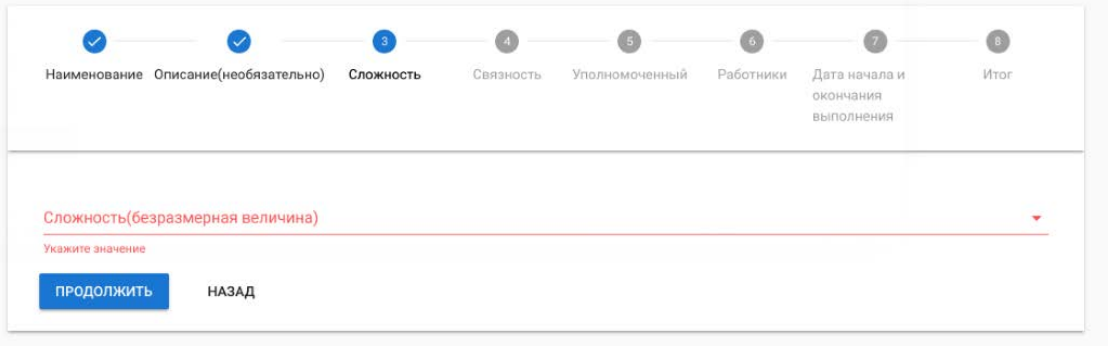

Рисунок 5.11 – Проверка формы выбора сложности задания

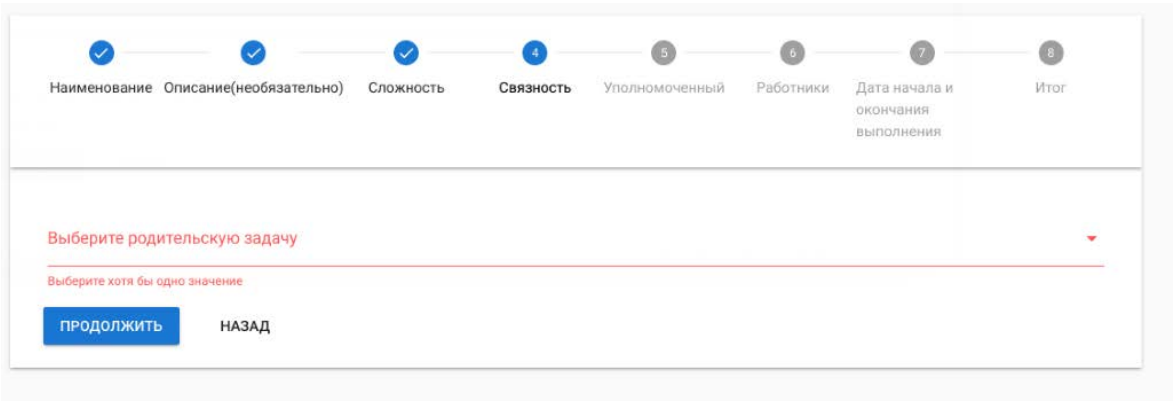

Рисунок 5.12 – Проверка формы выбора родительских задач

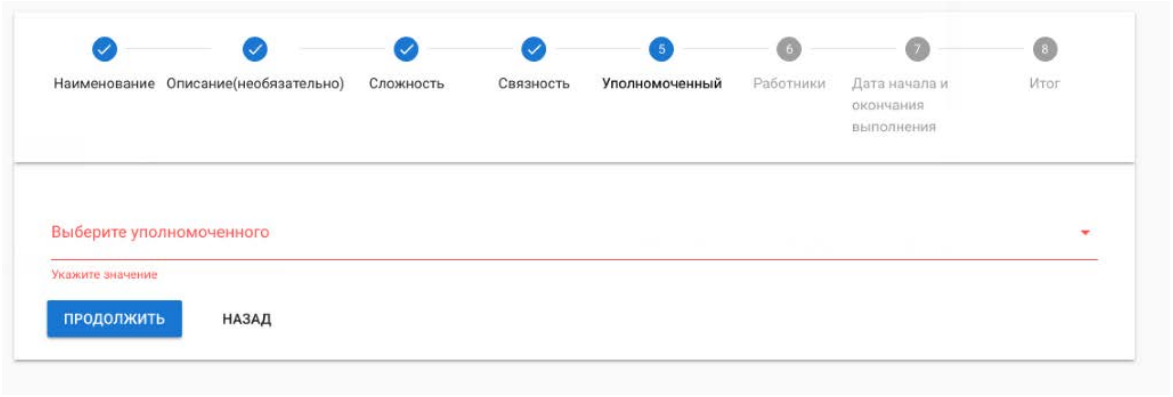

Рисунок 5.13 – Проверка формы выбора уполномоченного

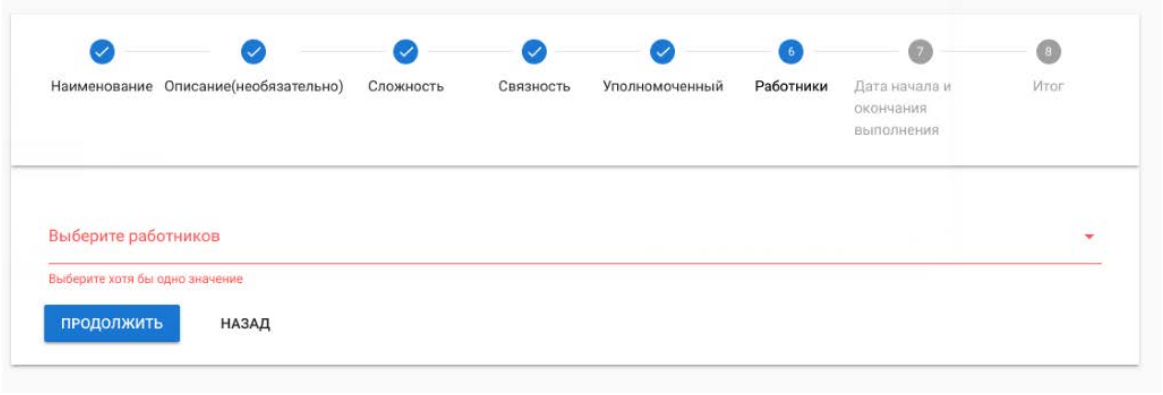

Рисунок 5.14 – Проверка формы выбора работников

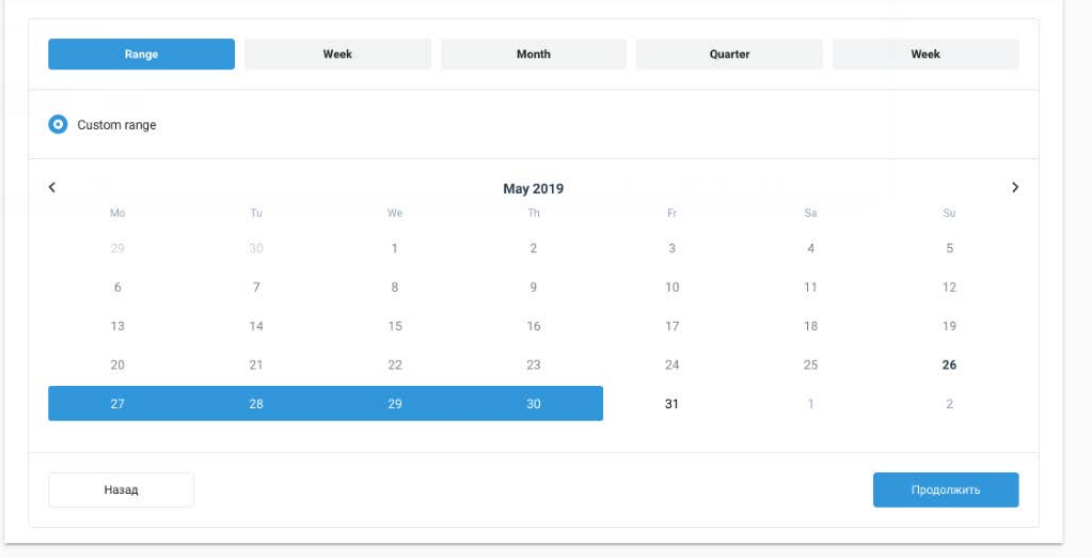

Рисунок 5.15 – Форма выбор периода выполнения

5.2. Тестирование прав пользователя

В приложении существует система прав как на уровне всего приложения, так и структурах проекта, списка задач или отдельно взятой задачи. Для выявления ошибок, связанных с неправильным отображением информации и выдачей неправильной возможности работы, были проведены тесты прав пользователя.

На следующем рисунке 5.16 представлено, как пользователь с правами “USER” в рамках всего приложения не может перейти в панель администратора.

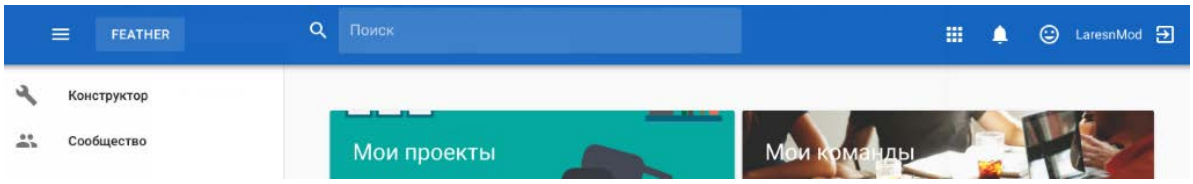

Рисунок 5.16 – Переход к панели администратора с правами “USER”

На рисунке 5.17 представлено, что пользователь с правами “ADMIN” имеет возможность перейти в панель администратора.

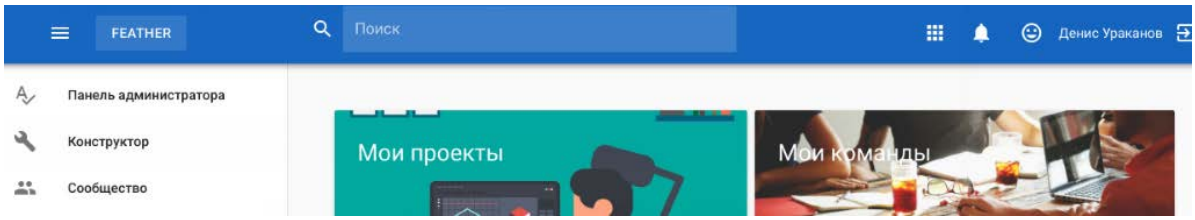

Рисунок 5.17 – Переход к панели администратора с правами “ADMIN”

Также пользователь, который имеет права “USER” в рамках проекта, может только просмотреть основную информацию о проекте, не совершая изменений в свойствах или работе с заявками к проекту (рисунок 5.18).

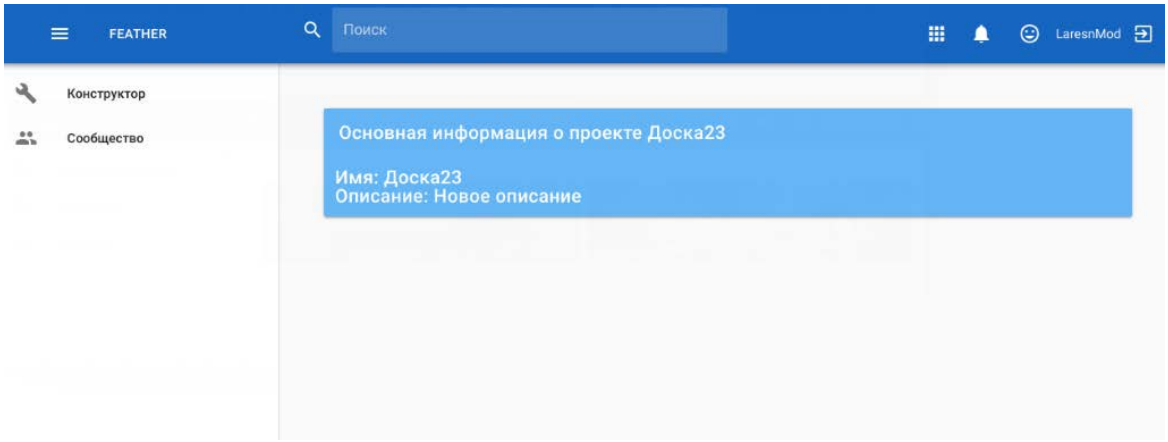

Рисунок 5.18 – Страница с изменением свойств проекта с правами “USER” для проекта

Пользователь, который имеет права “USER” в рамках списка задач, может только просмотреть основную информацию о списке (рисунок 5.19).

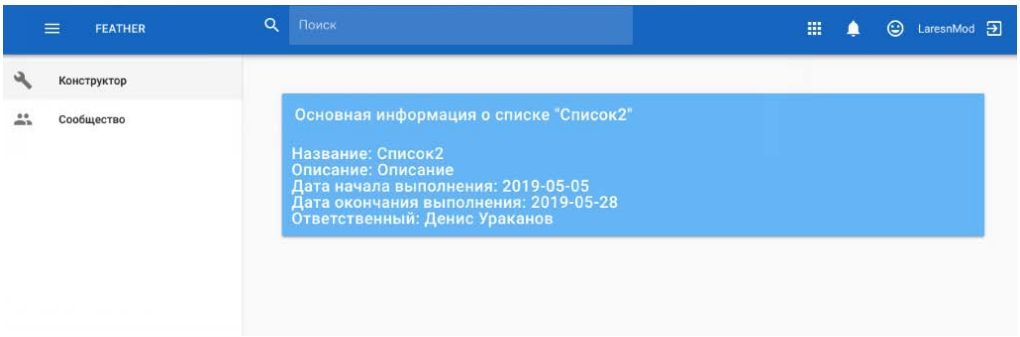

Рисунок 5.19 – Страница с изменением свойств списка с правами “USER” для списка

Также пользователь не в состоянии изменить свойства группы, если он не имеет прав “ADMIN” для данной группы (рисунок 5.20).

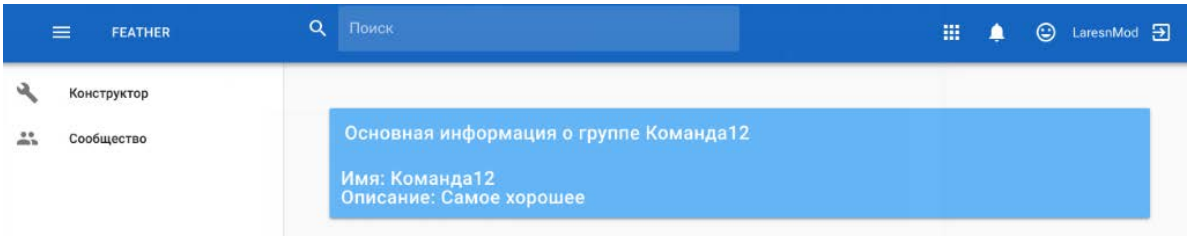

Рисунок 5.20 – Изменение свойств группы с правами “USER”

5.3. Тестирование системы приглашений и заявок

В приложении существует система приглашений и заявок. В рамках альфа-тестирования было проведено несколько испытаний данного функционала на примере группы.

Сначала первый пользователь создает группу и в меню выбора участников выбирает второго (рисунок 5.21).

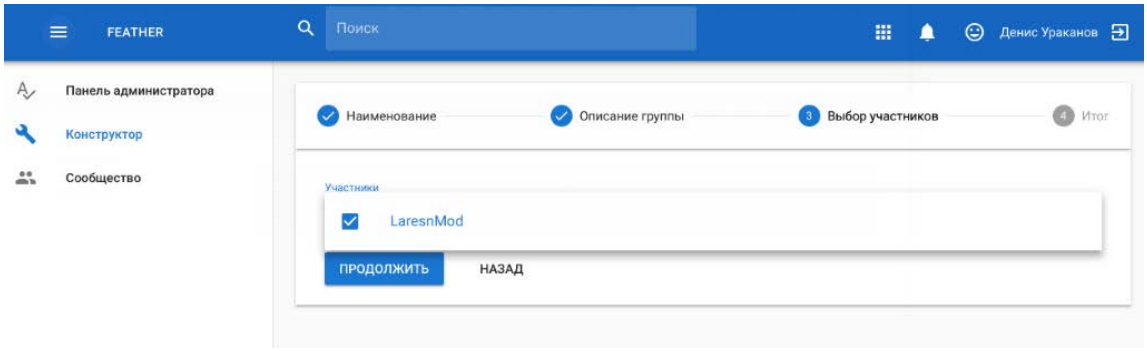

Рисунок 5.21 – Приглашения второго пользователя в группу

Второй пользователь при переходе в личный кабинет имеет возможность увидеть данное приглашение и согласиться или отклонить его (рисунок 5.22). Текущий список групп второго пользователя представлен на рисунке 5.23.

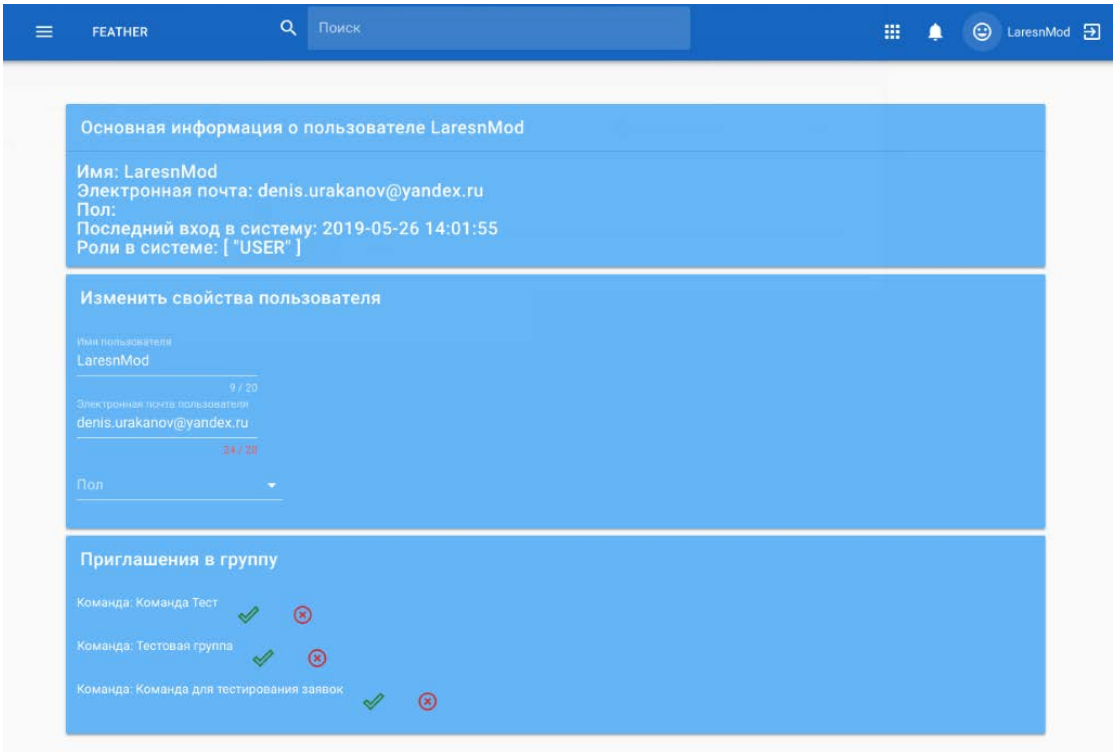

Рисунок 5.22 – Приглашения в группы второго пользователя

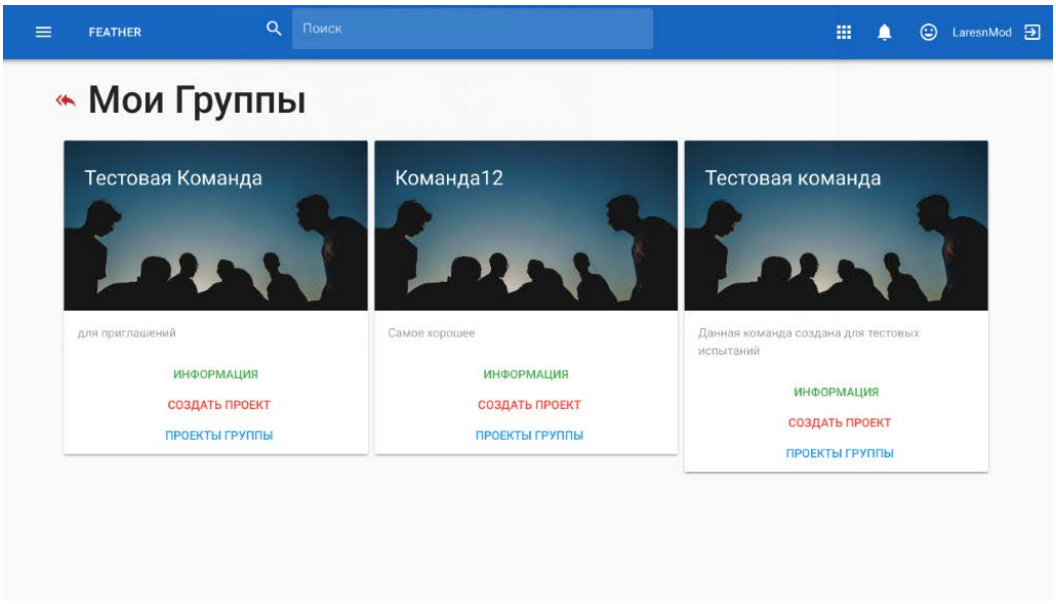

Рисунок 5.23 – Текущий список групп второго пользователя

Также второй пользователь может увидеть свой текущий статус по отношению к данной группе в списке групп сообщества (рисунок 5.24).

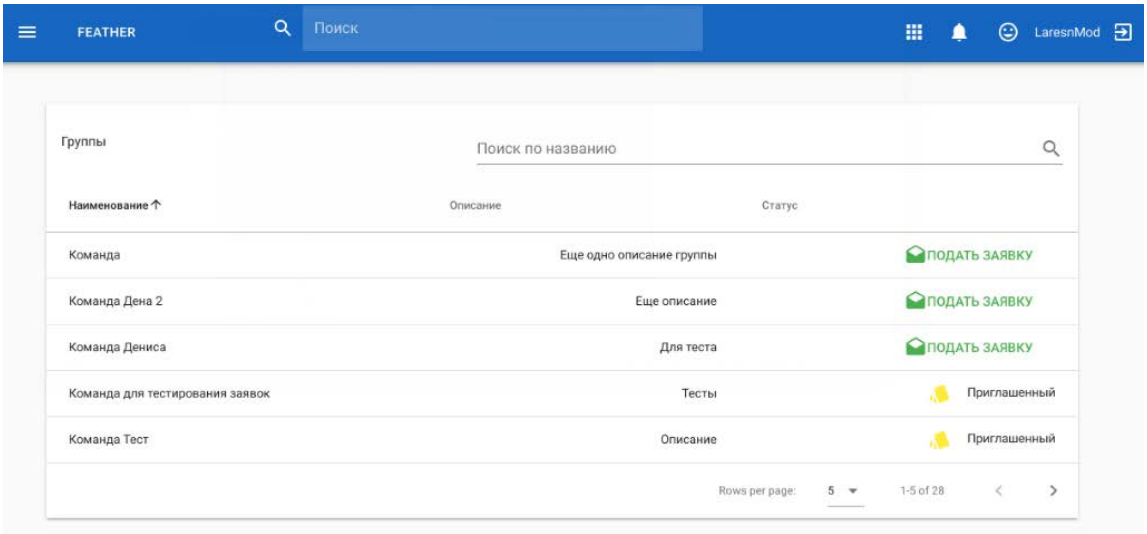

Рисунок 5.24 – Список групп сообщества

После принятия группы, созданной первым пользователем, второй имеет возможность увидеть данную команду в списке своих групп (рисунок 5.25).

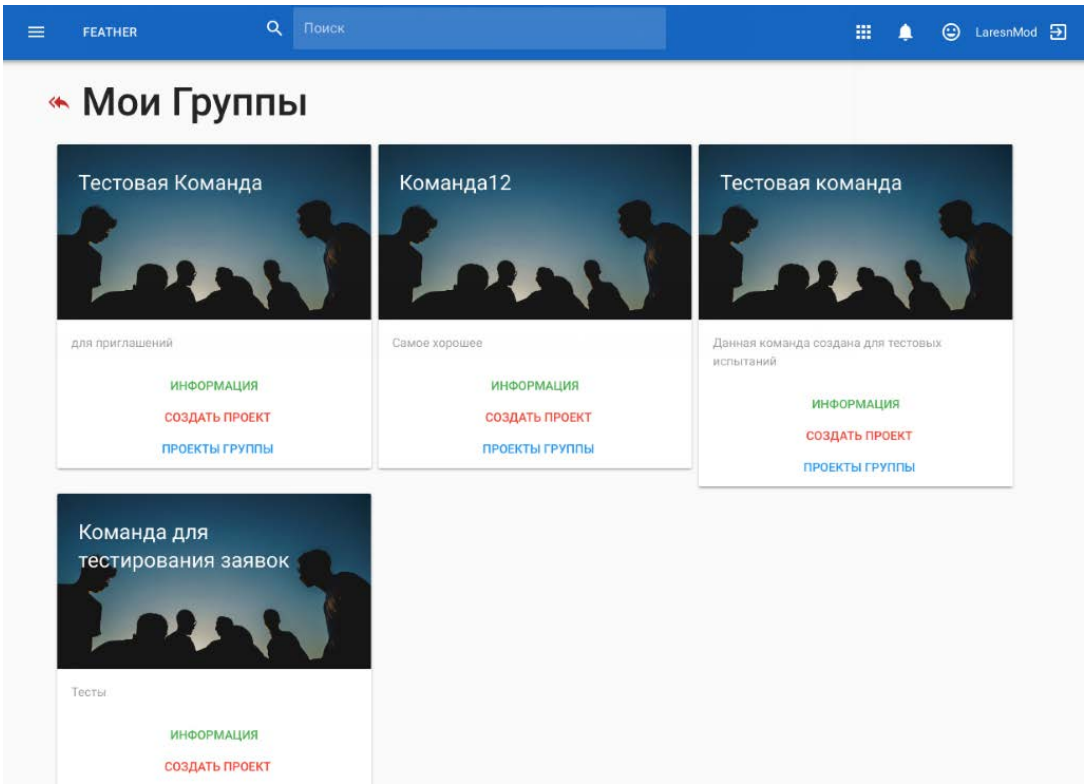

Рисунок 5.25 – Список групп пользователя после принятия приглашения

5.4. Тестирование оценки надежности выполнения задачи

Для тестирования оценки надежности выполнения задачи необходимо сначала отдельно просчитать данную оценку вручную. Затем сравнить с выдачей на сервисе.

Для наиболее интересного случая в тестировании будет два пользователя с выполненными несколькими заданиями в рамках проекта «Создание фреймворка».

Первый пользователь выполнил задание «Создание макета» сложности 4 за 6 дней, задание «Описание структуры» сложности 3 за 2 дня, задание «Создание опроса» сложности 5 за 2 дня. Данный пользователь на момент создания новой задачи имеет коэффициент ценности 2,4.

Второй пользователь выполнил задание «Создание БД» сложности 8 за 5 дней, задание «Выполнение оформления» сложности 9 за 10 дней, задание «Создание Вывода» сложности 3 за 5 дней. Второй пользователь имеет коэффициент ценности 1,2.

Средняя эффективность первого пользователя в данном списке задач (4/6 + 3/2 + 5/2)*2,4/3 = 3,73. Средняя эффективность второго пользователя в данном списке задач (8/5 + 9/10 + 3/5)*1,2/3 = 1,24. Средняя по двум пользователям равна 2,485.

На сервисе уже были подставлены необходимые параметры. В базе данных уже хранится информация по списку задач для тестирования (рисунок 5.26)

name	start_time	task_list_id	status	time_complete
Задача5	<null>	13	WAIT	<null>
Задача1	<null>	13	WAIT	<null>
Задача2	<null>	13	WAIT	<null>
Задача3	<null>	13	WAIT	<null>
Задача4	<null>	13	WAIT	<null>
Задание6	<null>	14	WAIT	<null>
Задание7	<null>	14	WAIT	<null>
Задание6	<null>	15	WAIT	<null>
Задание7	<null>	15	WAIT	<null>
Задание9	<null>	16	WAIT	<null>
Задание8	<null>	16	WAIT	<null>
Задание12	<null>	16	WAIT	<null>
Задание11	<null>	16	WAIT	<null>
Задание10	<null>	16	WAIT	<null>
Задание15	<null>	17	WAIT	<null>
Задание17	<null>	17	WAIT	<null>
Задание14	<null>	17	WAIT	<null>
Задание13	<null>	17	WAIT	<null>
Задание16	<null>	17	WAIT	<null>
Оформление	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-13 05:00:00.000000
Описание структуры	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-05 05:00:00.000000
Создание опроса	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-05 05:00:00.000000
Создание макета	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-06 05:00:00.000000
Создание БД	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-08 05:00:00.000000
Создание вывода	2019-06-03 05:00:00.000000	18	IS_DONE	2019-06-08 05:00:00.000000

Рисунок 5.26 – Задания для тестирования

Сам список представлен на рисунок 5.27.

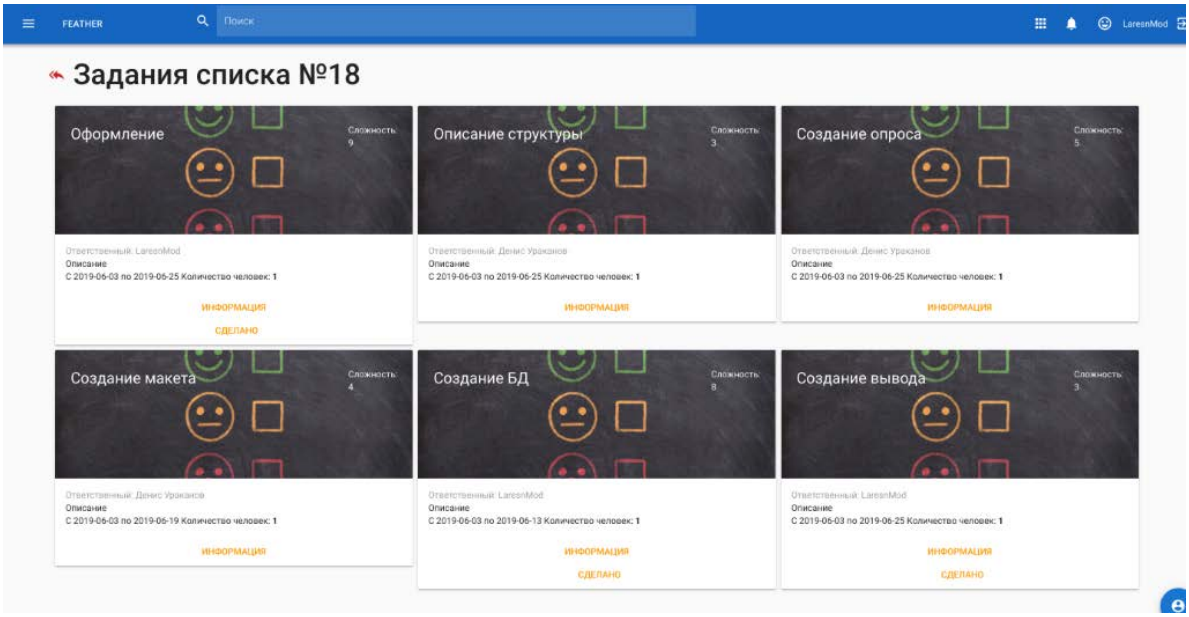

Рисунок 5.27 – Задания списка

Создадим новую задачу в этом списке. Выберем для неё сложность 8, а крайний день наступает через 2 дня. Сервис нам должен выдать число (2,485/(8/2)) *100% = 62,125%. На рисунках показаны шаги создания задачи.

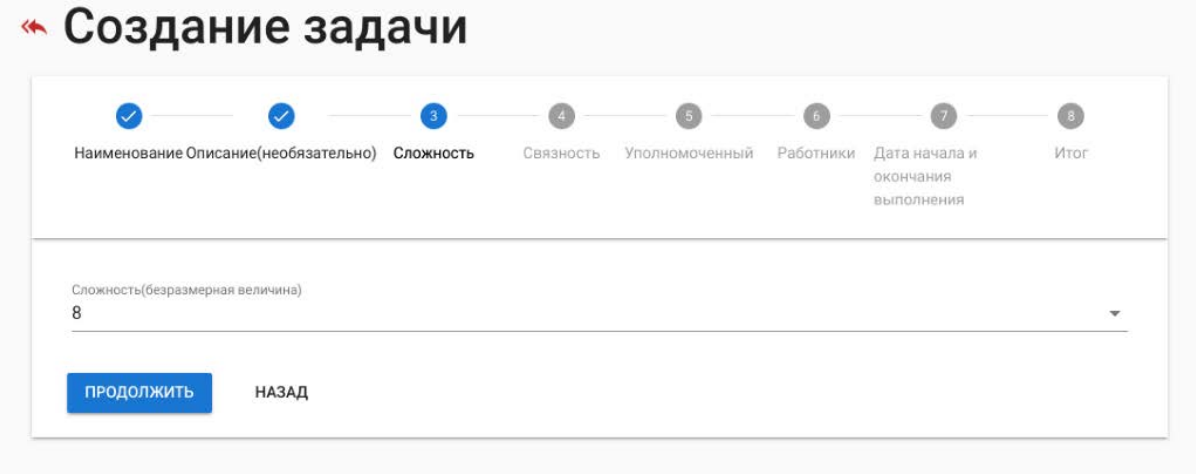

Рисунок 5.28 – Создание задачи

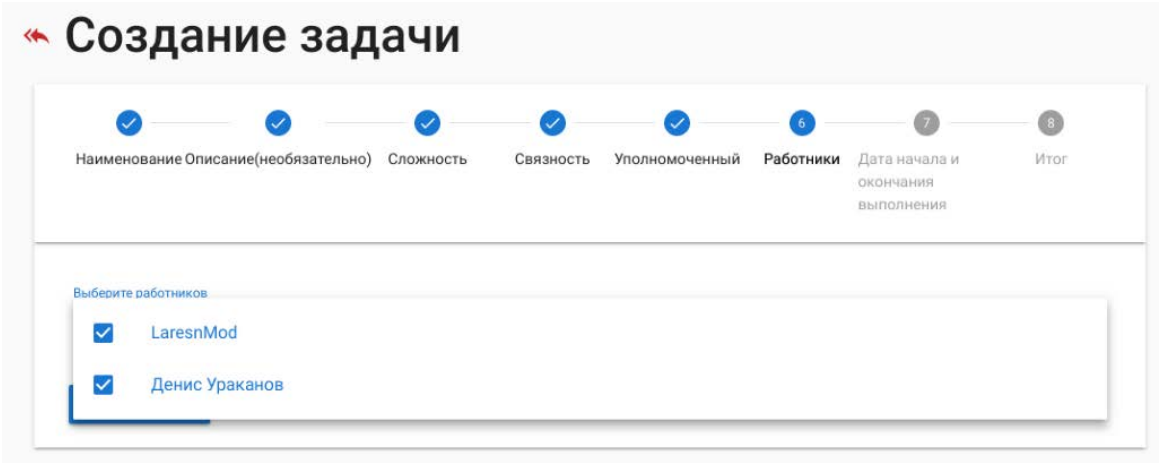

Рисунок 5.29 – Создание задачи

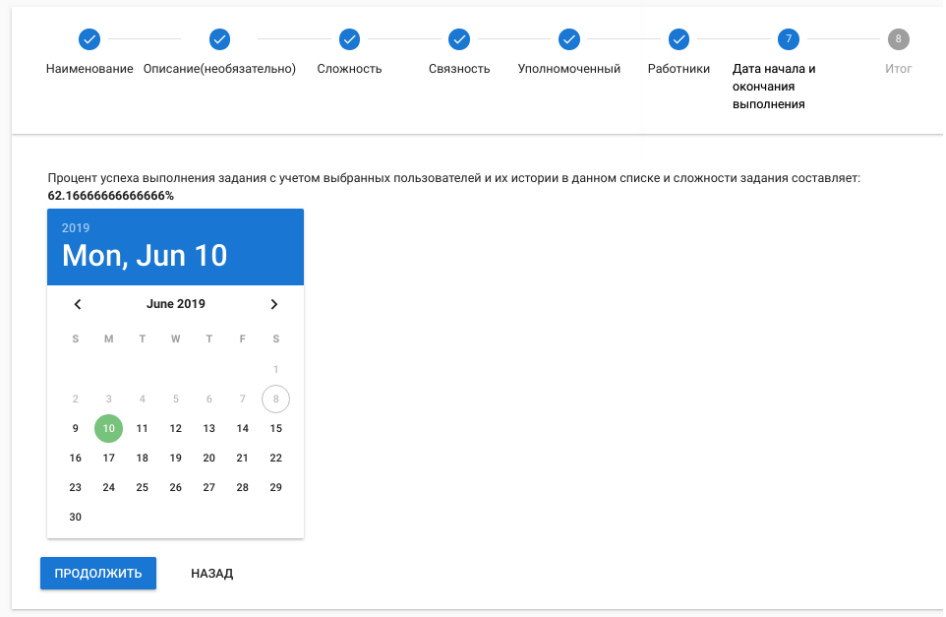

Рисунок 5.30 – Создание задачи

Как можно заметить на рисунке 5.30 погрешность результата на сервисе ~ 0,01%. Данное расхождение происходит из-за перевода вещественных чисел на серверной части. Но в рамках проекта и оценки выполнения задания данная погрешность имеет малое влияние.

5.5. Тестирование отображения списка проектов, списка задач и отдельных задач

Примеры страниц пользователя со связанными проектами, списками задач и отдельных задач представлены на рисунках 4.12, 4.13, 4.33.

5.6. Тестирование слияния проектов

Примеры с тестированием слияния проектов представлены в п.4.5.

5.7. Тестирование отображения графов задач

Примеры страниц с отображением графов задач представлены на рисунках 4.34, 4.35.

5.8. Тестирование отображения списка групп

Примеры страниц с отображением списка групп данного пользователя представлены на рисунке 4.43.

5.9. Тестирование отображения списка проектов и групп сообщества

Примеры страниц с отображением списка проектов и групп сообщества представлены на рисунках 4.47 и 4.50.

6. Заключение

В ходе дипломного проектирования было выполнено следующее:

1. Проведен анализ современного подхода к взаимодействию между членами команды, изучены способы гибкой работы небольших групп.
2. Проведен анализ предметной области и рынка родственных разработок.
3. Определены функциональные и нефункциональные требования к приложению.
4. Спроектирована архитектура веб-приложения.
5. Разработана структура базы данных.
6. Разработано и протестировано программное обеспечение, выполняющее поставленную задачу.

В веб-приложении были реализованы следующие функции:

1. Возможность организовывать работу с проектами, списками задач и отдельными задачами.
2. Возможность создавать команды и создавать проекты, связанные с данными командами.
3. Возможность отображения проекта или списка задач в виде графа.
4. Создание и отображение оптимального выполнения списка задач.
5. Сообщество, в котором пользователи могут приглашать или быть приглашенными к определенному проекту.
6. Возможность взаимодействия проектов.
7. Система модерирования приложения.

В ближайших планах стоит цель добавления сущностей коммерческих организаций и связанных с ней функционалом.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Проектирование баз данных и работа с ними Веб-приложений. Введение в БД, SQL Server, ADO.NET. – https://www.intuit.ru/studies/courses/611/467/lecture/28791%3Fpage%3D6. Дата обращения: 12.02.2019.
2. Базы данных - Урок 1. Понятие базы данных. – https://www.site-do.ru/db/db1.php. Дата обращения: 20.04.2019.
3. Что такое реляционная база данных? – https://aws.amazon.com/ru/relational-database/. Дата обращения: 02.03.2019.
4. Что такое NoSQL? – https://aws.amazon.com/ru/nosql/. Дата обращения: 04.03.2019.
5. Python. – http://wikireality.ru/wiki/Python. Дата обращения: 06.03.2019.
6. Язык программирования JavaScript: информация для начинающих. – https://www.internet-technologies.ru/articles/yazyk-programmirovaniya-javascript-informaciya-dlya-nachinayuschih.html. Дата обращения: 07.03.2019.
7. Преимущества и недостатки C#. – https://vk.com/@profcsharp-preimuschestva-i-nedostatki-c. Дата обращения: 08.03.2019.
8. Лучшие JavaScript (JS) Фреймворки и Библиотеки 2019. – https://merehead.com/ru/blog/top-javascript-frameworks-2019/. Дата обращения: 08.03.2019.
9. Single Page Applications — что это? – https://fokit.ru/single-page-applications-chto-eto/. Дата обращения: 08.03.2019.
10. Agile, scrum, kanban: в чем разница и для чего использовать? – https://rb.ru/story/agile-scrum-kanban/. Дата обращения: 14.03.2019.

11. Core Technologies. – https://docs.spring.io/spring/docs/5.2.0.BUILD-SNAPSHOT/spring-framework-reference/core.html#spring-core. Дата обращения: 25.03.2019.
12. Spring Boot Reference Guide. – https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/. Дата обращения: 25.03.2019.
13. Hibernate ORM Documentation – 5.4. – https://hibernate.org/orm/documentation/5.4/. Дата обращения: 25.03.2019.
14. Introduction. – https://vuejs.org/v2/guide/. Дата обращения: 09.04.2019.
15. Быстрый старт. – https://vuetifyjs.com/ru/getting-started/quick-start. Дата обращения: 09.04.2019.
16. Documentation. – https://www.postgresql.org/docs/. Дата обращения: 11.04.2019.
17. Java EE at a Glance. – https://www.oracle.com/technetwork/java/javaee/overview/index.html. Дата обращения: 15.04.2019.
18. Icons. – https://material.io/tools/icons/. Дата обращения: 14.05.2019.
19. Сазерленд, Дж. Scrum. Революционный метод управления проектами / Дж. Сазерленд; пер. с англ. М. Глинской – М.: Манн, Иванов и Фербер, 2017. – 272 с.
20. Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссилес – СПб.: Питер, 2001. – 368 с.
21. Мартин, Р. Принципы, паттерны и методики гибкой разработки на языке C# / Р. Мартин; пер. с англ. А. Слинкина. – СПб.: Символ-Плюс, 2011. – 768 с.

22. Хорстман, К. С. Java. Библиотека профессионала, том 1. Основы. / К. С. Хорстман, Г. Корнелл; пер. с англ. – М.: Издательский дом «Вильямс», 2014. – 864 с.
23. Фаулер, Мартин. Архитектура корпоративных программных приложений / М. Фаулер; пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 544 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД МОДЕЛИ USER

Листинг А.1 – Класс сущности пользователя

```
/**
 * Класс User.
 */
@Entity
@Table(name = "usr")
@Data
public class User implements Serializable {
    @Id
    private String id;
    private String name;
    private String userpic;
    private String email;
    private String gender;
    private String locale;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd
HH:mm:ss")
    private LocalDateTime lastVisit;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name = "us-
er_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;

    private Double valueCoeff;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonIgnore
    private Set<UserGroup> userGroups = new HashSet<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonIgnore
    private Set<UserDesk> userDesks = new HashSet<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonIgnore
    private Set<UserTasklist> userTasklists = new HashSet<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonIgnore
    private Set<UserTask> userTasks = new HashSet<>();

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public String getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
     * Возвращаемое значение:  id Определенный идентификатор
```

```
 *
 * Входной параметр: id the id
 */
public void setId(String id) {
    this.id = id;
}

/**
 * Функция получения name.
 *
 * @return определенное наименование
 */
public String getName() {
    return name;
}

/**
 * Передаваемые аргументы: name.
 * <p>
 * Возвращаемое значение:  name определенное наименование
 *
 * Входной параметр: name the name
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Функция получения userpic.
 *
 * @return Определенный пользовательpic
 */
public String getUserpic() {
    return userpic;
}

/**
 * Передаваемые аргументы: userpic.
 * <p>
 * Возвращаемое значение:  userpic Определенный пользовательpic
 *
 * Входной параметр: userpic Определенный пользовательpic
 */
public void setUserpic(String userpic) {
    this.userpic = userpic;
}

/**
 * Функция получения email.
 *
 * @return the email
 */
public String getEmail() {
    return email;
}

/**
 * Передаваемые аргументы: email.
 * <p>
 * Возвращаемое значение:  email the email
 *
 * Входной параметр: email the email
```

```
 */
public void setEmail(String email) {
    this.email = email;
}

/**
 * Функция получения gender.
 *
 * @return the gender
 */
public String getGender() {
    return gender;
}

/**
 * Передаваемые аргументы: gender.
 * <p>
 * Возвращаемое значение:  gender the gender
 *
 * Входной параметр: gender the gender
 */
public void setGender(String gender) {
    this.gender = gender;
}

/**
 * Функция получения locale.
 *
 * @return the locale
 */
public String getLocale() {
    return locale;
}

/**
 * Передаваемые аргументы: locale.
 * <p>
 * Возвращаемое значение:  locale the locale
 *
 * Входной параметр: locale the locale
 */
public void setLocale(String locale) {
    this.locale = locale;
}

/**
 * Функция получения last visit.
 *
 * @return the last visit
 */
public LocalDateTime getLastVisit() {
    return lastVisit;
}

/**
 * Передаваемые аргументы: last visit.
 * <p>
 * Возвращаемое значение:  lastVisit the last visit
 *
 * Входной параметр: lastVisit the last visit
 */
public void setLastVisit(LocalDateTime lastVisit) {
```

```
    this.lastVisit = lastVisit;
}

/**
 * Функция получения roles.
 *
 * @return Определенная ролs
 */
public Set<Role> getRoles() {
    return roles;
}

/**
 * Передаваемые аргументы: roles.
 * <p>
 * Возвращаемое значение:  roles Определенная ролs
 *
 * Входной параметр: roles Определенная ролs
 */
public void setRoles(Set<Role> roles) {
    this.roles = roles;
}

/**
 * Функция получения user groups.
 *
 * @return Определенный пользователь groups
 */
public Set<UserGroup> getUserGroups() {
    return userGroups;
}

/**
 * Передаваемые аргументы: user groups.
 * <p>
 * Возвращаемое значение:  userGroups Определенный пользователь groups
 *
 * Входной параметр: userGroups Определенный пользователь groups
 */
public void setUserGroups(Set<UserGroup> userGroups) {
    this.userGroups = userGroups;
}

/**
 * Функция получения user desks.
 *
 * @return Определенный пользователь desks
 */
public Set<UserDesk> getUserDesks() {
    return userDesks;
}

/**
 * Передаваемые аргументы: user desks.
 * <p>
 * Возвращаемое значение:  userDesks Определенный пользователь desks
 *
 * Входной параметр: userDesks Определенный пользователь desks
 */
public void setUserDesks(Set<UserDesk> userDesks) {
    this.userDesks = userDesks;
}
```

```
    /**
     * Функция получения user tasklists.
     *
     * @return Определенный пользователь tasklists
     */
    public Set<UserTasklist> getUserTasklists() {
        return userTasklists;
    }

    /**
     * Передаваемые аргументы: user tasklists.
     * <p>
     * Возвращаемое значение:  userTasklists Определенный пользователь tasklists
     *
     * Входной параметр: userTasklists Определенный пользователь tasklists
     */
    public void setUserTasklists(Set<UserTasklist> userTasklists) {
        this.userTasklists = userTasklists;
    }

    /**
     * Функция получения user tasks.
     *
     * @return Определенный пользователь tasks
     */
    public Set<UserTask> getUserTasks() {
        return userTasks;
    }

    /**
     * Передаваемые аргументы: user tasks.
     * <p>
     * Возвращаемое значение:  userTasks Определенный пользователь tasks
     *
     * Входной параметр: userTasks Определенный пользователь tasks
     */
    public void setUserTasks(Set<UserTask> userTasks) {
        this.userTasks = userTasks;
    }

    /**
     * Функция получения value coeff.
     *
     * @return the value coeff
     */
    public Double getValueCoeff() {
        return valueCoeff;
    }

    /**
     * Передаваемые аргументы: value coeff.
     * <p>
     * Возвращаемое значение:  valueCoeff the value coeff
     *
     * Входной параметр: valueCoeff the value coeff
     */
    public void setValueCoeff(Double valueCoeff) {
        this.valueCoeff = valueCoeff;
    }
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД МОДЕЛИ GROUP

Листинг Б.1 – Класс сущности группы

```
/**
 * Класс Group.
 * <p>
 * Возвращаемое значение:
 */
@Entity
@Table(name = "grp")
public class Group implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    private String name;

    private String description;

    @OneToMany(mappedBy = "group", fetch = FetchType.EAGER)
    @JsonIgnore
    private Set<UserGroup> userGroups = new HashSet<>();

    @OneToMany(mappedBy = "group", fetch = FetchType.EAGER)
    @JsonIgnore
    private Set<GroupDesk> groupDesks = new HashSet<>();

    /**
     * Конструктор. Создание объекта класса Group.
     */
    public Group() {
    }

    /**
     * Конструктор. Создание объекта класса Group.
     * <p>
     * Возвращаемое значение:  name        определенное наименование
     * Возвращаемое значение:  description the description
     * Возвращаемое значение:  userGroups  Определенный пользователь groups
     * Возвращаемое значение:  groupDesks  Определенная группа desks
     *
     * Входной параметр: name        the name
     * Входной параметр: description the description
     * Входной параметр: userGroups  Определенный пользователь groups
     * Входной параметр: groupDesks  Определенная группа desks
     */
    public Group(String name, String description, Set<UserGroup> userGroups,
Set<GroupDesk> groupDesks) {
        this.name = name;
        this.description = description;
        this.userGroups = userGroups;
        this.groupDesks = groupDesks;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
```

```
 */
public Long getId() {
    return id;
}

/**
 * Передаваемые аргументы: id.
 * <p>
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Функция получения name.
 *
 * @return определенное наименование
 */
public String getName() {
    return name;
}

/**
 * Передаваемые аргументы: name.
 * <p>
 * Возвращаемое значение:  name определенное наименование
 *
 * Входной параметр: name the name
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Функция получения user groups.
 *
 * @return Определенный пользователь groups
 */
public Set<UserGroup> getUserGroups() {
    return userGroups;
}

/**
 * Передаваемые аргументы: user groups.
 * <p>
 * Возвращаемое значение:  userGroups Определенный пользователь groups
 *
 * Входной параметр: userGroups Определенный пользователь groups
 */
public void setUserGroups(Set<UserGroup> userGroups) {
    this.userGroups = userGroups;
}
```

```
    /**
     * Функция получения description.
     *
     * @return the description
     */
    public String getDescription() {
        return description;
    }

    /**
     * Передаваемые аргументы: description.
     * <p>
     * Возвращаемое значение:  description the description
     *
     * Входной параметр: description the description
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * Функция получения group desks.
     *
     * @return Определенная группа desks
     */
    public Set<GroupDesk> getGroupDesks() {
        return groupDesks;
    }

    /**
     * Передаваемые аргументы: group desks.
     * <p>
     * Возвращаемое значение:  groupDesks Определенная группа desks
     *
     * Входной параметр: groupDesks Определенная группа desks
     */
    public void setGroupDesks(Set<GroupDesk> groupDesks) {
        this.groupDesks = groupDesks;
    }
}
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД МОДЕЛИ USERGROUP

Листинг В.1 – Класс сущности связи пользователя и группы

```
/**
 * Класс User group.
 */
@Entity
@Table(name = "usr_grp")
public class UserGroup implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "group_id")
    private Group group;

    private String role;

    /**
     * Конструктор. Создание объекта класса User group.
     */
    public UserGroup() {
    }

    /**
     * Конструктор. Создание объекта класса User group.
     * <p>
     * Возвращаемое значение:  user  Определенный пользователь
     * Возвращаемое значение:  group Определенная группа
     * Возвращаемое значение:  role  Определенная роль
     *
     * Входной параметр: user  Определенный пользователь
     * Входной параметр: group Определенная группа
     * Входной параметр: role  Определенная роль
     */
    public UserGroup(User user, Group group, String role) {
        this.user = user;
        this.group = group;
        this.role = role;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
```

```
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}
/**
 * Функция получения user.
 *
 * @return Определенный пользователь
 */
public User getUser() {
    return user;
}
/**
 * Передаваемые аргументы: user.
 * <p>
 * Возвращаемое значение:  user Определенный пользователь
 *
 * Входной параметр: user Определенный пользователь
 */
public void setUser(User user) {
    this.user = user;
}
/**
 * Функция получения group.
 *
 * @return Определенная группа
 */
public Group getGroup() {
    return group;
}
/**
 * Передаваемые аргументы: group.
 * <p>
 * Возвращаемое значение:  group Определенная группа
 *
 * Входной параметр: group Определенная группа
 */
public void setGroup(Group group) {this.group = group;}

/**
 * Функция получения role.
 *
 * @return Определенная роль
 */
public String getRole() {return role;}

/**
 * Передаваемые аргументы: role.
 * <p>
 * Возвращаемое значение:  role Определенная роль
 *
 * Входной параметр: role Определенная роль
 */
public void setRole(String role) {this.role = role; }}
```

ПРИЛОЖЕНИЕ Г
ИСХОДНЫЙ КОД МОДЕЛИ DESK

Листинг Г.1 – Класс сущности проекта

```
/**
 * Класс Desk. Доски.
 */
@Entity
@Table(name = "desk")
public class Desk implements Serializable {
    private static final long serialVersionUID = -3360543705272372261L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    private String name;
    private String description;

    @OneToMany(mappedBy = "desk", cascade = CascadeType.ALL, fetch = Fet-
chType.LAZY)
    @JsonIgnore
    private List<TaskList> taskLists;

    @OneToMany(mappedBy = "desk")
    @JsonIgnore
    private Set<UserDesk> userDesks = new HashSet<>();

    @OneToMany(mappedBy = "desk")
    @JsonIgnore
    private Set<GroupDesk> groupDesks = new HashSet<>();

    @OneToMany(mappedBy = "desk")
    @JsonIgnore
    private Set<DeskRequestMerge> deskRequestMerges = new HashSet<>();

    @ManyToMany
    @JoinTable(name = "deskrelation",
               joinColumns = @JoinColumn(name="id_deskMaster", referencedCo-
lumnName = "id"),
               inverseJoinColumns = @JoinColumn(name = "id_deskSlave", referen-
cedColumnName = "id"))
    private List<Desk> deskRelation = new ArrayList<>();

    /**
     * Создание объекта Desk.
     */
    public Desk() {
    }

    /**
     * Создание объекта Desk.
     * <p>
     * Возвращаемое значение:  name             определенное наименование, имя
     * Возвращаемое значение:  description      the description, описание
     * Возвращаемое значение:  taskLists        определенный список задач, спи-
сок задач
     * Возвращаемое значение:  userDesks        Определенный пользователь
desks, связь с пользователями
```

```
     * Возвращаемое значение:  groupDesks        Определенная группа desks,
связь с группами
     * Возвращаемое значение:  deskRequestMerges Определенный проект request
merges, список заявок на слияние
     * Возвращаемое значение:  deskRelation      Определенный проект relation,
связи с другими досками
     *
     * Входной параметр: name              the name
     * Входной параметр: description       the description
     * Входной параметр: taskLists         the task lists
     * Входной параметр: userDesks         Определенный пользователь desks
     * Входной параметр: groupDesks        Определенная группа desks
     * Входной параметр: deskRequestMerges Определенный проект request merges
     * Входной параметр: deskRelation      Определенный проект relation
     */
    public Desk(String name, String description, List<TaskList> taskLists,
Set<UserDesk> userDesks, Set<GroupDesk> groupDesks, Set<DeskRequestMerge> de-
skRequestMerges, List<Desk> deskRelation) {
        this.name = name;
        this.description = description;
        this.taskLists = taskLists;
        this.userDesks = userDesks;
        this.groupDesks = groupDesks;
        this.deskRequestMerges = deskRequestMerges;
        this.deskRelation = deskRelation;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
     * Возвращаемое значение:  id Определенный идентификатор
     *
     * Входной параметр: id the id
     */
    public void setId(Long id) {
        this.id = id;
    }

    /**
     * Функция получения name.
     *
     * @return определенное наименование
     */
    public String getName() {
        return name;
    }

    /**
     * Передаваемые аргументы: name.
     * <p>
     * Возвращаемое значение:  name определенное наименование
     *
     * Входной параметр: name the name
```

```
 */
public void setName(String name) {
    this.name = name;
}
/**
 * Функция получения task lists.
 *
 * @return определенный список задачs
 */
public List<TaskList> getTaskLists() {
    return taskLists;
}
/**
 * Передаваемые аргументы: task lists.
 * <p>
 * Возвращаемое значение:  taskLists определенный список задачs
 *
 * Входной параметр: taskLists the task lists
 */
public void setTaskLists(List<TaskList> taskLists) {
    this.taskLists = taskLists;
}
/**
 * Функция получения description.
 *
 * @return the description
 */
public String getDescription() {
    return description;
}
/**
 * Передаваемые аргументы: description.
 * <p>
 * Возвращаемое значение:  description the description
 *
 * Входной параметр: description the description
 */
public void setDescription(String description) {
    this.description = description;
}
/**
 * Функция получения user desks.
 *
 * @return Определенный пользователь desks
 */
public Set<UserDesk> getUserDesks() {
    return userDesks;
}
/**
 * Передаваемые аргументы: user desks.
 * <p>
 * Возвращаемое значение:  userDesks Определенный пользователь desks
 *
 * Входной параметр: userDesks Определенный пользователь desks
 */
public void setUserDesks(Set<UserDesk> userDesks) {
    this.userDesks = userDesks;
}

/**
 * Функция получения serial version uid.
 *
```

```
     * @return the serial version uid
     */
    public static long getSerialVersionUID() {
        return serialVersionUID;
    }
    /**
     * Функция получения group desks.
     *
     * @return Определенная группа desks
     */
    public Set<GroupDesk> getGroupDesks() {
        return groupDesks;
    }
    /**
     * Передаваемые аргументы: group desks.
     * <p>
     * Возвращаемое значение:  groupDesks Определенная группа desks
     *
     * Входной параметр: groupDesks Определенная группа desks
     */
    public void setGroupDesks(Set<GroupDesk> groupDesks) {
        this.groupDesks = groupDesks;
    }
    /**
     * Функция получения desk relation.
     *
     * @return Определенный проект relation
     */
    public List<Desk> getDeskRelation() {
        return deskRelation;
    }
    /**
     * Передаваемые аргументы: desk relation.
     * <p>
     * Возвращаемое значение:  deskRelation Определенный проект relation
     *
     * Входной параметр: deskRelation Определенный проект relation
     */
    public void setDeskRelation(List<Desk> deskRelation) {
        this.deskRelation = deskRelation;
    }
    /**
     * Функция получения desk request merges.
     *
     * @return Определенный проект request merges
     */
    public Set<DeskRequestMerge> getDeskRequestMerges() {
        return deskRequestMerges;
    }
    /**
     * Передаваемые аргументы: desk request merges.
     * <p>
     * Возвращаемое значение:  deskRequestMerges Определенный проект request
merges
     *
     * Входной параметр: deskRequestMerges Определенный проект request merges
     */
    public void setDeskRequestMerges(Set<DeskRequestMerge> deskRequestMerges) {
        this.deskRequestMerges = deskRequestMerges; }}
```

ПРИЛОЖЕНИЕ Д
ИСХОДНЫЙ КОД МОДЕЛИ TASKLIST

Листинг Д.1 – Класс сущности списка задач

```
/**
 * Класс Task list.
 */
@Entity
@Table(name = "task_list")
public class TaskList implements Serializable {
    private static final long serialVersionUID = 8763771067078165195L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    private String name;
    private String discription;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date start_time;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date end_time;

    private Integer userNumber;

    @OneToMany(mappedBy = "taskList", cascade = CascadeType.ALL, fetch = Fet-
chType.LAZY)
    @JsonIgnore
    private List<Task> tasks;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "desk_id")
    @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class,
property = "id")
    @JsonIdentityReference(alwaysAsId = true)
    private Desk desk;

    @OneToMany(mappedBy = "taskList")
    @JsonIgnore
    private Set<UserTasklist> userTasklists = new HashSet<>();

    /**
     * Создание объекта Task list.
     */
    public TaskList() {
    }

    /**
     * Создание объекта Task list.
     * <p>
     * Возвращаемое значение:  name          определенное наименование, имя спи-
ска
     * Возвращаемое значение:  discription   определенное описание, описание
списка
     * Возвращаемое значение:  start_time    определенное начало выполнения, на-
чало выполнения
     * Возвращаемое значение:  end_time      определенное окончание выполнения,
окончание выполнения
     * Возвращаемое значение:  userNumber    Определенный пользователь number,
число участников
```

```
     * Возвращаемое значение:  tasks         Определенное заданиеs, связанные
задачи
     * Возвращаемое значение:  desk          Определенный проект, связанная дос-
ка
     * Возвращаемое значение:  userTasklists Определенный пользователь
tasklists, связанные пользователи
     *
     * Входной параметр: name          the name
     * Входной параметр: discription   the discription
     * Входной параметр: start_time    the start time
     * Входной параметр: end_time      the end time
     * Входной параметр: userNumber    Определенный пользователь number
     * Входной параметр: tasks         the tasks
     * Входной параметр: desk          Определенный проект
     * Входной параметр: userTasklists Определенный пользователь tasklists
     */
    public TaskList(String name, String discription, Date start_time, Date
end_time, Integer userNumber, List<Task> tasks, Desk desk, Set<UserTasklist>
userTasklists) {
        this.name = name;
        this.discription = discription;
        this.start_time = start_time;
        this.end_time = end_time;
        this.userNumber = userNumber;
        this.tasks = tasks;
        this.desk = desk;
        this.userTasklists = userTasklists;
    }

    @Override
    public String toString() {
        return "TaskList{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", discription='" + discription + '\'' +
                ", start_time=" + start_time +
                ", end_time=" + end_time +
                ", tasks=" + tasks +
                ", desk=" + desk +
                ", userTasklists=" + userTasklists +
                '}';
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
     * Возвращаемое значение:  id Определенный идентификатор
     *
     * Входной параметр: id the id
     */
    public void setId(Long id) {
        this.id = id;
    }
```

```
/**
 * Функция получения name.
 *
 * @return определенное наименование
 */
public String getName() {
    return name;
}

/**
 * Передаваемые аргументы: name.
 * <p>
 * Возвращаемое значение:  name определенное наименование
 *
 * Входной параметр: name the name
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Функция получения discription.
 *
 * @return определенное описание
 */
public String getDiscription() {
    return discription;
}

/**
 * Передаваемые аргументы: discription.
 * <p>
 * Возвращаемое значение:  discription определенное описание
 *
 * Входной параметр: discription the discription
 */
public void setDiscription(String discription) {
    this.discription = discription;
}

/**
 * Функция получения start time.
 *
 * @return определенное начало выполнения
 */
public Date getStart_time() {
    return start_time;
}

/**
 * Передаваемые аргументы: start time.
 * <p>
 * Возвращаемое значение:  start_time определенное начало выполнения
 *
 * Входной параметр: start_time the start time
 */
```

```
public void setStart_time(Date start_time) {
    this.start_time = start_time;
}

/**
 * Функция получения end time.
 *
 * @return определенное окончание выполнения
 */


public Date getEnd_time() {
    return end_time;
}

/**
 * Передаваемые аргументы: end time.
 * <p>
 * Возвращаемое значение:  end_time определенное окончание выполнения
 *
 * Входной параметр: end_time the end time
 */
public void setEnd_time(Date end_time) {
    this.end_time = end_time;
}

/**
 * Функция получения tasks.
 *
 * @return Определенное заданиеs
 */
public List<Task> getTasks() {
    return tasks;
}

/**
 * Передаваемые аргументы: tasks.
 * <p>
 * Возвращаемое значение:  tasks Определенное заданиеs
 *
 * Входной параметр: tasks the tasks
 */
public void setTasks(List<Task> tasks) {
    this.tasks = tasks;
}

/**
 * Функция получения desk.
 *
 * @return Определенный проект
 */
public Desk getDesk() {
    return desk;
}

/**
 * Передаваемые аргументы: desk.
 * <p>
 * Возвращаемое значение:  desk Определенный проект
 *
 * Входной параметр: desk Определенный проект
 */
```

```
    public void setDesk(Desk desk) {
        this.desk = desk;
    }

    /**
     * Функция получения user tasklists.
     *
     * @return Определенный пользователь tasklists
     */
    public Set<UserTasklist> getUserTasklists() {
        return userTasklists;
    }

    /**
     * Передаваемые аргументы: user tasklists.
     * <p>
     * Возвращаемое значение:  userTasklists Определенный пользователь tasklists
     *
     * Входной параметр: userTasklists Определенный пользователь tasklists
     */
    public void setUserTasklists(Set<UserTasklist> userTasklists) {
        this.userTasklists = userTasklists;
    }

    /**
     * Функция получения user number.
     *
     * @return Определенный пользователь number
     */
    public Integer getUserNumber() {
        return userNumber;
    }

    /**
     * Передаваемые аргументы: user number.
     * <p>
     * Возвращаемое значение:  userNumber Определенный пользователь number
     *
     * Входной параметр: userNumber Определенный пользователь number
     */
    public void setUserNumber(Integer userNumber) {
        this.userNumber = userNumber;
    }

}
```

ПРИЛОЖЕНИЕ Е
ИСХОДНЫЙ КОД МОДЕЛИ TASK

Листинг Е.1 – Класс сущности задачи

```
/**
 * Класс Task. Задачи.
 */
@Entity
@Table(name = "task")
public class Task implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    private String name;
    private String discription;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date start_time;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date end_time;

    private Integer complexity;
    private Integer headerTaskList;

    @Transient
    private Integer stage;
    @Transient
    private Integer userComplexity;

    private String status;
    private Date time_complete;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "taskList_id")
    @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class,
property = "id")
    @JsonIdentityReference(alwaysAsId = true)
    private TaskList taskList;

    @OneToMany(mappedBy = "task", cascade = CascadeType.ALL)
    @JsonIgnore
    private Set<UserTask> userTasks = new HashSet<>();

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable( name = "taskrelation",
            joinColumns = @JoinColumn(name = "id_task",referencedColumnName =
"id"),
            inverseJoinColumns = @JoinColumn(name =
"id_child",referencedColumnName = "id"))
    private List<Task> taskRelation = new ArrayList<Task>();

    /**
     * Создание объекта Task.
     */
    public Task() {
    }
    /**
     * Создание объекта Task.
     * <p>
```

```
     * Возвращаемое значение:  name           определенное наименование, имя
     * Возвращаемое значение:  discription    определенное описание, описание
     * Возвращаемое значение:  start_time     определенное начало выполнения,
начало выполнения
     * Возвращаемое значение:  end_time       определенное окончание выполнения,
окончание выполнения
     * Возвращаемое значение:  complexity     the complexity, сложность
     * Возвращаемое значение:  headerTaskList the header task list, флаг первого
задания в списке
     * Возвращаемое значение:  stage          the stage, стадия выполнения
     * Возвращаемое значение:  userComplexity Определенный пользователь complex-
ity, сложность пользователя
     * Возвращаемое значение:  status         the status, статус
     * Возвращаемое значение:  time_complete  the time complete, дата окончания
выполнения
     * Возвращаемое значение:  taskList       определенный список задач, связан-
ный список задач
     * Возвращаемое значение:  userTasks      Определенный пользователь tasks,
связанные пользователи
     * Возвращаемое значение:  taskRelation   связь с заданиями, связанные про-
чие задачи
     *
     * Входной параметр: name           the name
     * Входной параметр: discription    the discription
     * Входной параметр: start_time     the start time
     * Входной параметр: end_time       the end time
     * Входной параметр: complexity     the complexity
     * Входной параметр: headerTaskList the header task list
     * Входной параметр: stage          the stage
     * Входной параметр: userComplexity Определенный пользователь complexity
     * Входной параметр: status         the status
     * Входной параметр: time_complete  the time complete
     * Входной параметр: taskList       the task list
     * Входной параметр: userTasks      Определенный пользователь tasks
     * Входной параметр: taskRelation   the task relation
     */
    public Task(String name, String discription, Date start_time, Date end_time,
Integer complexity, Integer headerTaskList, Integer stage, Integer userComplexi-
ty, String status, Date time_complete, TaskList taskList, Set<UserTask> user-
Tasks, List<Task> taskRelation) {
        this.name = name;
        this.discription = discription;
        this.start_time = start_time;
        this.end_time = end_time;
        this.complexity = complexity;
        this.headerTaskList = headerTaskList;
        this.stage = stage;
        this.userComplexity = userComplexity;
        this.status = status;
        this.time_complete = time_complete;
        this.taskList = taskList;
        this.userTasks = userTasks;
        this.taskRelation = taskRelation;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
```

```
}

/**
 * Передаваемые аргументы: id.
 * <p>
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Функция получения name.
 *
 * @return определенное наименование
 */
public String getName() {
    return name;
}

/**
 * Передаваемые аргументы: name.
 * <p>
 * Возвращаемое значение:  name определенное наименование
 *
 * Входной параметр: name the name
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Функция получения discription.
 *
 * @return определенное описание
 */
public String getDiscription() {
    return discription;
}

/**
 * Передаваемые аргументы: discription.
 * <p>
 * Возвращаемое значение:  discription определенное описание
 *
 * Входной параметр: discription the discription
 */
public void setDiscription(String discription) {
    this.discription = discription;
}

/**
 * Функция получения start time.
 *
 * @return определенное начало выполнения
 */
public Date getStart_time() {
    return start_time;
}
/**
```

```
 * Передаваемые аргументы: start time.
 * <p>
 * Возвращаемое значение:  start_time определенное начало выполнения
 *
 * Входной параметр: start_time the start time
 */
public void setStart_time(Date start_time) {
    this.start_time = start_time;
}
/**
 * Функция получения end time.
 *
 * @return определенное окончание выполнения
 */
public Date getEnd_time() {
    return end_time;
}
/**
 * Передаваемые аргументы: end time.
 * <p>
 * Возвращаемое значение:  end_time определенное окончание выполнения
 *
 * Входной параметр: end_time the end time
 */
public void setEnd_time(Date end_time) {
    this.end_time = end_time;
}
/**
 * Функция получения task list.
 *
 * @return определенный список задач
 */
public TaskList getTaskList() {
    return taskList;
}
/**
 * Передаваемые аргументы: task list.
 * <p>
 * Возвращаемое значение:  taskList определенный список задач
 *
 * Входной параметр: taskList the task list
 */
public void setTaskList(TaskList taskList) {
    this.taskList = taskList;
}
/**
 * Функция получения user tasks.
 *
 * @return Определенный пользователь tasks
 */
public Set<UserTask> getUserTasks() {
    return userTasks;
}
/**
 * Передаваемые аргументы: user tasks.
 * <p>
 * Возвращаемое значение:  userTasks Определенный пользователь tasks
 *
 * Входной параметр: userTasks Определенный пользователь tasks
 */
public void setUserTasks(Set<UserTask> userTasks) {
    this.userTasks = userTasks;
```

```
}

/**
 * Функция получения task relation.
 *
 * @return связь с заданиями
 */
public List<Task> getTaskRelation() {
    return taskRelation;
}
/**
 * Передаваемые аргументы: task relation.
 * <p>
 * Возвращаемое значение:  taskRelation связь с заданиями
 *
 * Входной параметр: taskRelation the task relation
 */
public void setTaskRelation(List<Task> taskRelation) {
    this.taskRelation = taskRelation;
}
/**
 * Функция получения complexity.
 *
 * @return the complexity
 */
public Integer getComplexity() {
    return complexity;
}
/**
 * Передаваемые аргументы: complexity.
 * <p>
 * Возвращаемое значение:  complexity the complexity
 *
 * Входной параметр: complexity the complexity
 */
public void setComplexity(Integer complexity) {
    this.complexity = complexity;
}
/**
 * Функция получения header task list.
 *
 * @return the header task list
 */
public Integer getHeaderTaskList() {
    return headerTaskList;
}
/**
 * Передаваемые аргументы: header task list.
 * <p>
 * Возвращаемое значение:  headerTaskList the header task list
 *
 * Входной параметр: headerTaskList the header task list
 */
public void setHeaderTaskList(Integer headerTaskList) {
    this.headerTaskList = headerTaskList;
}
/**
 * Функция получения stage.
 *
 * @return the stage
 */
public Integer getStage() {
```

```
        return stage;}
    /**
     * Передаваемые аргументы: stage.
     * <p>
     * Возвращаемое значение:  stage the stage
     *
     * Входной параметр: stage the stage
     */
    public void setStage(Integer stage) {this.stage = stage;}
    /**
     * Функция получения user complexity.
     *
     * @return Определенный пользователь complexity
     */
    public Integer getUserComplexity() {return userComplexity;}
    /**
     * Передаваемые аргументы: user complexity.
     * <p>
     * Возвращаемое значение:  userComplexity Определенный пользователь complex-
ity
     *
     * Входной параметр: userComplexity Определенный пользователь complexity
     */
    public void setUserComplexity(Integer userComplexity) {this.userComplexity =
userComplexity;}
    /**
     * Функция получения status.
     *
     * @return the status
     */
    public String getStatus() {
        return status;
    }
    /**
     * Передаваемые аргументы: status.
     * <p>
     * Возвращаемое значение:  status the status
     *
     * Входной параметр: status the status
     */
    public void setStatus(String status) {
        this.status = status;
    }
    /**
     * Функция получения time complete.
     *
     * @return the time complete
     */
    public Date getTime_complete() {
        return time_complete;
    }
    /**
     * Передаваемые аргументы: time complete.
     * <p>
     * Возвращаемое значение:  time_complete the time complete
     *
     * Входной параметр: time_complete the time complete
     */
    public void setTime_complete(Date time_complete) {
        this.time_complete = time_complete;
    }
}
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД МОДЕЛИ DESKREQUESTMERGE

Листинг Ж.1 – Класс сущностии заявки на слияние проектов

```
/**
 * Класс Desk request merge. Форма подачи заявки на слияние проектов.
 */
@Entity
@Table(name = "deskRequestMerge")
public class DeskRequestMerge implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column
    private Long id;

    @ManyToOne
    @JoinColumn(name = "desk_id")
    private Desk desk;

    private String deskReqIds;

    private String role;

    /**
     * Создание объекта Desk request merge.
     */
    public DeskRequestMerge() {
    }

    /**
     * Создание объекта Desk request merge.
     * <p>
     * Возвращаемое значение:  desk       Определенный проект, доска, на которую
подается заявка на слияние
     * Возвращаемое значение:  deskReqIds Определенный проект req ids, доски,
которые подают заявку на слияние
     * Возвращаемое значение:  role       Определенная роль, будущая роль доски,
на которую подается заявка: как часть общего проекта, или полноценное слияние
     *
     * Входной параметр: desk       Определенный проект
     * Входной параметр: deskReqIds Определенный проект req ids
     * Входной параметр: role       Определенная роль
     */
    public DeskRequestMerge(Desk desk, String deskReqIds, String role) {
        this.desk = desk;
        this.deskReqIds = deskReqIds;
        this.role = role;
    }
    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }
    /**
     * Передаваемые аргументы: id.
     * <p>
     * Возвращаемое значение:  id Определенный идентификатор
```

```
     *
     * Входной параметр: id the id
     */
    public void setId(Long id) {
        this.id = id;
    }
    /**
     * Функция получения desk.
     *
     * @return Определенный проект
     */
    public Desk getDesk() {
        return desk;
    }
    /**
     * Передаваемые аргументы: desk.
     * <p>
     * Возвращаемое значение:  desk Определенный проект
     *
     * Входной параметр: desk Определенный проект
     */
    public void setDesk(Desk desk) {
        this.desk = desk;
    }
    /**
     * Функция получения desk req ids.
     *
     * @return Определенный проект req ids
     */
    public String getDeskReqIds() {
        return deskReqIds;
    }
    /**
     * Передаваемые аргументы: desk req ids.
     * <p>
     * Возвращаемое значение:  deskReqIds Определенный проект req ids
     *
     * Входной параметр: deskReqIds Определенный проект req ids
     */
    public void setDeskReqIds(String deskReqIds) {
        this.deskReqIds = deskReqIds;
    }
    /**
     * Функция получения role.
     *
     * @return Определенная роль
     */
    public String getRole() {
        return role;
    }
    /**
     * Передаваемые аргументы: role.
     * <p>
     * Возвращаемое значение:  role Определенная роль
     *
     * Входной параметр: role Определенная роль
     */
    public void setRole(String role) {
        this.role = role;
    }
}
```

ПРИЛОЖЕНИЕ К
ИСХОДНЫЙ КОД МОДЕЛИ GROUPDESK

Листинг К.1 – Класс сущности связи между группой и доской

```
/**
 * Класс Group desk.Связь группы и доски.
 */
@Entity
@Table(name = "group_desk")
public class GroupDesk implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column
    private Long id;

    @ManyToOne
    @JoinColumn(name = "group_id")
    private Group group;

    @ManyToOne
    @JoinColumn(name = "desk_id")
    private Desk desk;

    private String role;

    /**
     * Создание объекта Group desk.
     */
    public GroupDesk() {
    }

    /**
     * Создание объекта Group desk.
     * <p>
     * Возвращаемое значение:  group Определенная группа, группа
     * Возвращаемое значение:  desk  Определенный проект, доска
     * Возвращаемое значение:  role  Определенная роль, роль
     *
     * Входной параметр: group Определенная группа
     * Входной параметр: desk  Определенный проект
     * Входной параметр: role  Определенная роль
     */
    public GroupDesk(Group group, Desk desk, String role) {
        this.group = group;
        this.desk = desk;
        this.role = role;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
```

```
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Функция получения group.
 *
 * @return Определенная группа
 */
public Group getGroup() {
    return group;
}

/**
 * Передаваемые аргументы: group.
 * <p>
 * Возвращаемое значение:  group Определенная группа
 *
 * Входной параметр: group Определенная группа
 */
public void setGroup(Group group) {
    this.group = group;
}
/**
 * Функция получения desk.
 *
 * @return Определенный проект
 */
public Desk getDesk() {
    return desk;
}
/**
 * Передаваемые аргументы: desk.
 * <p>
 * Возвращаемое значение:  desk Определенный проект
 *
 * Входной параметр: desk Определенный проект
 */
public void setDesk(Desk desk) {
    this.desk = desk;
}
/**
 * Функция получения role.
 *
 * @return Определенная роль
 */
public String getRole() {
    return role;
}
/**
 * Передаваемые аргументы: role.
 * <p>
 * Возвращаемое значение:  role Определенная роль
 *
 * Входной параметр: role Определенная роль
 */
public void setRole(String role) {this.role = role;}}
```

ПРИЛОЖЕНИЕ Л
ИСХОДНЫЙ КОД МОДЕЛИ USERDESK

Листинг Л.1 – Класс сущности связи пользователя и проекта

```
/**
 * Класс User desk.
 */
@Entity
@Table(name = "user_desk")
public class UserDesk implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "desk_id")
    private Desk desk;

    private String role;

    /**
     * Конструктор. Создание объекта класса User desk.
     */
    public UserDesk() {
    }

    /**
     * Конструктор. Создание объекта класса User desk.
     * <p>
     * Возвращаемое значение:  user Определенный пользователь
     * Возвращаемое значение:  desk Определенный проект
     * Возвращаемое значение:  role Определенная роль
     *
     * Входной параметр: user Определенный пользователь
     * Входной параметр: desk Определенный проект
     * Входной параметр: role Определенная роль
     */
    public UserDesk(User user, Desk desk, String role) {
        this.user = user;
        this.desk = desk;
        this.role = role;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
```

```
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Функция получения user.
 *
 * @return Определенный пользователь
 */
public User getUser() {
    return user;
}

/**
 * Передаваемые аргументы: user.
 * <p>
 * Возвращаемое значение:  user Определенный пользователь
 *
 * Входной параметр: user Определенный пользователь
 */
public void setUser(User user) {
    this.user = user;
}
/**
 * Функция получения desk.
 *
 * @return Определенный проект
 */
public Desk getDesk() {
    return desk;
}
/**
 * Передаваемые аргументы: desk.
 * <p>
 * Возвращаемое значение:  desk Определенный проект
 *
 * Входной параметр: desk Определенный проект
 */
public void setDesk(Desk desk) {
    this.desk = desk;
}
/**
 * Функция получения role.
 *
 * @return Определенная роль
 */
public String getRole() {
    return role;
}
/**
 * Передаваемые аргументы: role.
 * <p>
 * Возвращаемое значение:  role Определенная роль
 *
 * Входной параметр: role Определенная роль
 */
public void setRole(String role) {this.role = role;}}
```

ПРИЛОЖЕНИЕ М
ИСХОДНЫЙ КОД МОДЕЛИ USERTASK

Листинг М.1 – Класс сущности связи между пользователем и задачей

```
/**
 * Класс User task.
 */
@Entity
@Table(name = "user_task")
public class UserTask implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "task_id")
    private Task task;

    private String role;

    /**
     * Конструктор. Создание объекта класса User task.
     */
    public UserTask() {
    }

    /**
     * Конструктор. Создание объекта класса User task.
     * <p>
     * Возвращаемое значение:  user Определенный пользователь
     * Возвращаемое значение:  task Определенное задание
     * Возвращаемое значение:  role Определенная роль
     *
     * Входной параметр: user Определенный пользователь
     * Входной параметр: task the task
     * Входной параметр: role Определенная роль
     */
    public UserTask(User user, Task task, String role) {
        this.user = user;
        this.task = task;
        this.role = role;
    }

    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }

    /**
     * Передаваемые аргументы: id.
     * <p>
```

```
 * Возвращаемое значение:  id Определенный идентификатор
 *
 * Входной параметр: id the id
 */
public void setId(Long id) {
    this.id = id;
}
/**
 * Функция получения user.
 *
 * @return Определенный пользователь
 */
public User getUser() {
    return user;
}
/**
 * Передаваемые аргументы: user.
 * <p>
 * Возвращаемое значение:  user Определенный пользователь
 *
 * Входной параметр: user Определенный пользователь
 */
public void setUser(User user) {
    this.user = user;
}
/**
 * Функция получения task.
 *
 * @return Определенное задание
 */
public Task getTask() {
    return task;
}
/**
 * Передаваемые аргументы: task.
 * <p>
 * Возвращаемое значение:  task Определенное задание
 *
 * Входной параметр: task the task
 */
public void setTask(Task task) {
    this.task = task;
}
/**
 * Функция получения role.
 *
 * @return Определенная роль
 */
public String getRole() {
    return role;
}
/**
 * Передаваемые аргументы: role.
 * <p>
 * Возвращаемое значение:  role Определенная роль
 *
 * Входной параметр: role Определенная роль
 */
public void setRole(String role) {this.role = role;}}
```

ПРИЛОЖЕНИЕ Н

ИСХОДНЫЙ КОД МОДЕЛИ USERTASKLIST

Листинг Н.1 – Класс сущности связи между пользователем и списком

```
/**
 * Класс User tasklist.
 */
@Entity
@Table(name = "user_tasklist")
public class UserTasklist implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(columnDefinition = "serial")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "tasklist_id")
    private TaskList taskList;

    private String role;

    /**
     * Конструктор. Создание объекта класса User tasklist.
     */
    public UserTasklist() {
    }
    /**
     * Конструктор. Создание объекта класса User tasklist.
     * <p>
     * Возвращаемое значение:  user     Определенный пользователь
     * Возвращаемое значение:  taskList определенный список задач
     * Возвращаемое значение:  role     Определенная роль
     *
     * Входной параметр: user     Определенный пользователь
     * Входной параметр: taskList the task list
     * Входной параметр: role     Определенная роль
     */
    public UserTasklist(User user, TaskList taskList, String role) {
        this.user = user;
        this.taskList = taskList;
        this.role = role;
    }
    /**
     * Функция получения id.
     *
     * @return Определенный идентификатор
     */
    public Long getId() {
        return id;
    }
    /**
     * Передаваемые аргументы: id.
     * <p>
     * Возвращаемое значение:  id Определенный идентификатор
```

```
     *
     * Входной параметр: id the id
     */
    public void setId(Long id) {
        this.id = id;
    }
    /**
     * Функция получения user.
     *
     * @return Определенный пользователь
     */
    public User getUser() {
        return user;
    }
    /**
     * Передаваемые аргументы: user.
     * <p>
     * Возвращаемое значение:  user Определенный пользователь
     *
     * Входной параметр: user Определенный пользователь
     */
    public void setUser(User user) {
        this.user = user;
    }
    /**
     * Функция получения task list.
     *
     * @return определенный список задач
     */
    public TaskList getTaskList() {
        return taskList;
    }
    /**
     * Передаваемые аргументы: task list.
     * <p>
     * Возвращаемое значение:  taskList определенный список задач
     *
     * Входной параметр: taskList the task list
     */
    public void setTaskList(TaskList taskList) {
        this.taskList = taskList;
    }
    /**
     * Функция получения role.
     *
     * @return Определенная роль
     */
    public String getRole() {
        return role;
    }
    /**
     * Передаваемые аргументы: role.
     * <p>
     * Возвращаемое значение:  role Определенная роль
     *
     * Входной параметр: role Определенная роль
     */
    public void setRole(String role) {
        this.role = role;
    }
}
```

ПРИЛОЖЕНИЕ П
ИСХОДНЫЙ КОД ПЕРЕЧИСЛЕНИЯ ROLE

Листинг П.1 – Класс перечисления ролей в системе

```
/**
 * Перечисление ролей Role.
 */
public enum Role implements GrantedAuthority, Serializable {
    /**
     * User role. Обычная роль.
     */
    USER,
    /**
     * Admin role. Роль администратора.
     */
    ADMIN,
    /**
     * Guest role. Роль гостя.
     */
    GUEST,
    /**
     * User req role. Роль пользователя, подавшего заявку на полноценное пользо-
вание сервисом.
     */
    USER_REQ;

    @Override
    public String getAuthority() {
        return name();
    }
}
```

ПРИЛОЖЕНИЕ Р

ИСХОДНЫЙ КОД КОНФИГУРАЦИИ WEBSECURITYCONFIG

Листинг Р.1 – Класс конфигурации авторизации

```
/**
 * Класс Web security config для конфигурации авторизации.
 */
@Configuration
@EnableWebSecurity
@EnableOAuth2Sso
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .antMatcher("/**")
                .authorizeRequests()
                .antMatchers("/**","/rest/**", "/login**", "/js/**",
"/error**").permitAll()
                .anyRequest().authenticated()
                .and().logout().logoutSuccessUrl("/").permitAll()
                .and()
                .csrf().disable();
    }

    /**
     * Principal extractor principal extractor.
     * <p>
     * Возвращаемое значение:  userRepo экземлпляр репозиятория пользователя
     *
     * Входной параметр: userRepo Определенный пользователь repo
     * @return the principal extractor
     */
    @Bean
    public PrincipalExtractor principalExtractor(UserRepo userRepo) {
        return map -> {
            String id = (String) map.get("sub");

            User user = userRepo.findById(id).orElseGet(() -> {
                User newUser = new User();

                newUser.setId(id);
                newUser.setName((String) map.get("name"));
                newUser.setEmail((String) map.get("email"));
                newUser.setGender((String) map.get("gender"));
                newUser.setLocale((String) map.get("locale"));
                newUser.setUserpic((String) map.get("picture"));
                newUser.setRoles(Collections.singleton(Role.GUEST));
                newUser.setValueCoeff(0d);
                return newUser;
            });

            user.setLastVisit(LocalDateTime.now());
            user = userRepo.save(user);
            return user;
        };
    }
}
```

ПРИЛОЖЕНИЕ С
ИСХОДНЫЙ КОД РЕПОЗИТОРИЕВ СУЩНОСТЕЙ ПОЛЬЗОВАТЕЛЯ И ГРУППЫ

Листинг С.1 – Класс репозитория сущности пользователя

```
/**
 * The interface User repo.
 */
public interface UserRepo extends JpaRepository<User, String>{
    /**
     * Нахождение user через id user.
     *
     * Входной параметр: id the id
     * @return Определенный пользователь
     */
    User findUserById(String id);

    /**
     * Нахождение через name user.
     *
     * Входной параметр: username Определенный пользовательname
     * @return Определенный пользователь
     */
    User findByName(String username);

    /**
     * Нахождение всех через id user.
     *
     * Входной параметр: id the id
     * @return Определенный пользователь
     */
    User findAllById(String id);
}



/**
 * The interface Group repo.
 */
public interface GroupRepo extends JpaRepository<Group,Long> {
    /**
     * Нахождение group через id group.
     *
     * Входной параметр: id the id
     * @return Определенная группа
     */
    Group findGroupById(Long id);
}
```

```
/**
 * The interface User group repo.
 */
public interface UserGroupRepo extends JpaRepository<UserGroup,Long> {
    /**
     * Нахождение user group через id user group.
     *
     * Входной параметр: id the id
     * @return Определенный пользователь group
     */
    UserGroup findUserGroupById(Long id);

    /**
     * Нахождение user group через user and group user group.
     *
     * Входной параметр: user  Определенный пользователь
     * Входной параметр: group Определенная группа
     * @return Определенный пользователь group
     */
    UserGroup findUserGroupByUserAndGroup(User user, Group group);

    /**
     * Нахождение user groups через group list.
     *
     * Входной параметр: group Определенная группа
     * @return the list
     */
    List<UserGroup> findUserGroupsByGroup(Group group);

    /**
     * Нахождение user groups через user list.
     *
     * Входной параметр: user Определенный пользователь
     * @return the list
     */
    List<UserGroup> findUserGroupsByUser(User user);
}
```

ПРИЛОЖЕНИЕ Т
ИСХОДНЫЙ КОД РЕПОЗИТОРИЕВ СУЩНОСТЕЙ ЗАДАЧ, СПИСКОВ И ПРОЕКТОВ

Листинг Т.1 - Класс репозитория сущности проекта

```
/**
 * Интерфейс Desk repo.
 */
public interface DeskRepo extends JpaRepository<Desk,Long> {
    /**
     * Нахождение desk через id desk.
     *
     * Входной параметр: id the id
     * @return Определенный проект
     */
    Desk findDeskById(Long id);
}
```

Листинг Т.2 – Класс репозитория сущности задания

```
/**
 * Интерфейс Task repo.
 */
public interface TaskRepo extends JpaRepository<Task, Long> {
    /**
     * Нахождение task через id task.
     *
     * Входной параметр: id the id
     * @return the task
     */
    Task findTaskById(Long id);

    /**
     * Нахождение task через task list list.
     *
     * Входной параметр: taskList the task list
     * @return the list
     */
```

```
        List<Task> findTaskByTaskList(TaskList taskList);
    /**
     * Нахождение task через header task list task.
     *
     * Входной параметр: headerTaskList the header task list
     * @return the task
     */
    Task findTaskByHeaderTaskList(Integer headerTaskList);
}
```

Листинг Т.3 – Класс репозитория сущности спискса задач

```
/**
 * Интерфейс Task list repo.
 */
public interface TaskListRepo extends JpaRepository<TaskList,Long> {
    /**
     * Нахождение task list через id task list.
     *
     * Входной параметр: id the id
     * @return the task list
     */
    TaskList findTaskListById(Long id);

    /**
     * Нахождение task list через desk list.
     *
     * Входной параметр: desk Определенный проект
     * @return the list
     */
    List<TaskList> findTaskListByDesk(Desk desk);
}
```

ПРИЛОЖЕНИЕ У
ИСХОДНЫЙ КОД РЕПОЗИТОРИЕВ СВЯЗИ МЕЖДУ СУЩНОСТЯМИ

Листинг У.1 – Класс репозиятория сущности заявки на слияние проектов

```
/**
 * The interface Desk request merge repo.
 */
public interface DeskRequestMergeRepo extends JpaReposito-
ry<DeskRequestMerge,Long> {
    /**
     * Нахождение desk request merge через desk desk request merge.
     *
     * Входной параметр: desk Определенный проект
     * @return Определенный проект request merge
     */
    DeskRequestMerge findDeskRequestMergeByDesk(Desk desk);

    /**
     * Нахождение desk request merges через role desk request merge.
     *
     * Входной параметр: role Определенная роль
     * @return Определенный проект request merge
     */
    DeskRequestMerge findDeskRequestMergesByRole(String role);
}
```

Листинг У.2 – Класс репозитория сущности связи пользователя и группы

```
/**
 * The interface Group desk repo.
 */
public interface GroupDeskRepo extends JpaRepository<GroupDesk, Long> {
    /**
     * Нахождение group desk через group and desk group desk.
     *
     * Входной параметр: group Определенная группа
     * Входной параметр: desk  Определенный проект
     * @return Определенная группа desk
     */
    GroupDesk findGroupDeskByGroupAndDesk(Group group, Desk desk);

    /**
     * Нахождение group desk через group list.
     *
     * Входной параметр: group Определенная группа
     * @return the list
     */
    List<GroupDesk> findGroupDeskByGroup(Group group);

    /**
     * Нахождение group desk через desk list.
     *
     * Входной параметр: desk Определенный проект
     * @return the list
     */
    List<GroupDesk> findGroupDeskByDesk(Desk desk);
}
```

Листинг У.3 – Класс репозитория сущности связи пользователя и проекта

```
/**
 * The interface User desk repo.
 */
public interface UserDeskRepo extends JpaRepository<UserDesk, Long> {
    /**
     * Нахождение user desk через user and desk user desk.
     *
     * Входной параметр: user Определенный пользователь
     * Входной параметр: desk Определенный проект
     * @return Определенный пользователь desk
     */
    UserDesk findUserDeskByUserAndDesk(User user, Desk desk);

    /**
     * Нахождение user desks через user list.
     *
     * Входной параметр: user Определенный пользователь
     * @return the list
     */
    List<UserDesk> findUserDesksByUser(User user);

    /**
     * Нахождение user desks через desk list.
     *
     * Входной параметр: desk Определенный проект
     * @return the list
     */
    List<UserDesk> findUserDesksByDesk(Desk desk);
}
```

Листинг У.4 – Класс репозитория сущности связи пользотвателя и задачи

```
/**
 * The interface User task repo.
 */
public interface UserTaskRepo extends JpaRepository<UserTask, Long> {
    /**
     * Нахождение user task через user and task user task.
     *
     * Входной параметр: user Определенный пользователь
     * Входной параметр: task the task
     * @return Определенный пользователь task
     */
    UserTask findUserTaskByUserAndTask(User user, Task task);

    /**
     * Нахождение user task через user list.
     *
     * Входной параметр: user Определенный пользователь
     * @return the list
     */
    List<UserTask> findUserTaskByUser(User user);

    /**
     * Нахождение user task через task list.
     *
     * Входной параметр: task the task
     * @return the list
     */
    List<UserTask> findUserTaskByTask(Task task);
}
```

Листинг У.5 – Класс репозитория сущности связи пользователя и списка

```
/**
 * The interface User tasklist repo.
 */
public interface UserTasklistRepo extends JpaRepository<UserTasklist,Long> {
    /**
     * Нахождение user tasklist через user and task list user tasklist.
     *
     * Входной параметр: user     Определенный пользователь
     * Входной параметр: taskList the task list
     * @return Определенный пользователь tasklist
     */
    UserTasklist findUserTasklistByUserAndTaskList(User user, TaskList task-
List);

    /**
     * Нахождение user tasklist через user list.
     *
     * Входной параметр: user Определенный пользователь
     * @return the list
     */
    List<UserTasklist> findUserTasklistByUser(User user);

    /**
     * Нахождение user tasklist через task list list.
     *
     * Входной параметр: taskList the task list
     * @return the list
     */
    List<UserTasklist> findUserTasklistByTaskList(TaskList taskList);
}
```

ПРИЛОЖЕНИЕ Ф
ИСХОДНЫЙ КОД КЛАССА ОШИБКИ

Листинг Ф.1 – Класс описания ошибки в работе сервиса

```
/**
 * Класс Not found exception.
 */
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException {
}
```

ПРИЛОЖЕНИЕ Х
ИСХОДНЫЙ КОД ГЛАВНОГО КОНТРОЛЛЕРА

Листинг Х.1 – Класс главного контроллера сервиса

```
/**
 * Класс Main controller.
 */
@Controller
@RequestMapping("/")
public class MainController {
    private final MessageRepo messageRepo;

    @Value("${spring.profiles.active}")
    private String profile;
    @Autowired
    private UserRepo userRepo;
    //test
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private DeskService deskService;

    /**
     * Получение экземпляра Main controller.
     * <p>
     * Возвращаемое значение:  messageRepo эклемпляр message repo
     *
     * Входной параметр: messageRepo the message repo
     */
    @Autowired
    public MainController(MessageRepo messageRepo) {
        this.messageRepo = messageRepo;
    }

    /**
     * Возвращение главной страницы.
     * <p>
     * Возвращаемое значение:  model модель
     * Возвращаемое значение:  user  объект пользователя
     *
     * Входной параметр: model the model
     * Входной параметр: user  Определенный пользователь
     * @return строка string
     */
    @GetMapping
    public String main(Model model, @AuthenticationPrincipal User user) {
        HashMap<Object, Object> data = new HashMap<>();

        data.put("profile", user);
        data.put("messages", messageRepo.findAll());

        model.addAttribute("frontendData", data);
        model.addAttribute("isDevMode","dev".equals(profile));

        return "index";
    }
```

ПРИЛОЖЕНИЕ Ц
ИСХОДНЫЙ КОД СЕРВИСА DESKSERVICE

Листинг Ц.1 – Класс сервиса с сущностью проекта

```
/**
 * Класс Desk service.
 */
@Service
public class DeskService {
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private TaskListRepo taskListRepo;
    @Autowired
    private GroupDeskRepo groupDeskRepo;
    @Autowired
    private UserDeskRepo userDeskRepo;
    @Autowired
    private EffectiveWalkService effectiveWalkService;
    /**
     * Слияние проектов.
     *
     * Входной параметр: deskIds Идентификаторы всех сливаемых проектов
     * Входной параметр: newDesk Объект результата слияния
     */
    public void mergeProjects(List<Long> deskIds, Desk newDesk){
       for (Long deskId:
             deskIds) {
            Desk currentDesk = deskRepo.findDeskById(deskId);
            currentDesk.getDeskRelation().clear();
            deskRepo.save(currentDesk);
            taskListRe-
po.findTaskListByDesk(deskRepo.findDeskById(deskId)).stream().forEach(taskList -
> {
                taskList.setDesk(newDesk);
                taskListRepo.save(taskList);
            });
            groupDeskRepo.findGroupDeskByDesk(currentDesk).forEach(groupDesk ->
{
                GroupDesk groupDeskDB = groupDeskRe-
po.findGroupDeskByGroupAndDesk(groupDesk.getGroup(),newDesk);
                if(groupDeskDB == null){
                    GroupDesk newGroupDesk = new Group-
Desk(groupDesk.getGroup(),newDesk,groupDesk.getRole());
                    groupDeskRepo.save(newGroupDesk);
                }
                groupDeskRepo.delete(groupDesk);
            });
            userDeskRepo.findUserDesksByDesk(currentDesk).forEach(userDesk -> {
                UserDesk userDeskDB = userDeskRe-
po.findUserDeskByUserAndDesk(userDesk.getUser(),newDesk);
                if (userDeskDB == null){
                    UserDesk newUserDesk = new User-
Desk(userDesk.getUser(),newDesk,userDesk.getRole());
                    userDeskRepo.save(newUserDesk);
                } else{
                    if (userDeskDB.getRole().equals("USER") && user-
Desk.getRole().equals("ADMIN")){
```

```
                        userDeskDB.setRole(userDesk.getRole());
                    }
                }
                userDeskRepo.delete(userDesk);
            });
            deskRepo.delete(currentDesk);
        }
    }
    /**
     * Удаление desk.
     *
     * Входной параметр: deskId Определенный проект id
     */
    public void deleteDesk(Long deskId){
        Desk deskToDelete = deskRepo.findDeskById(deskId);
        Desk firstDesk = deskRepo.findDeskById(1L);
        deskToDelete.getDeskRelation().clear();
        deskRepo.save(deskToDelete);
        taskListRe-
po.findTaskListByDesk(deskRepo.findDeskById(deskId)).stream().forEach(taskList -
> {
            taskList.setDesk(firstDesk);
            taskListRepo.save(taskList);
        });
        groupDeskRepo.findGroupDeskByDesk(deskToDelete).forEach(groupDesk -> {
            groupDeskRepo.delete(groupDesk);
        });
        userDeskRepo.findUserDesksByDesk(deskToDelete).forEach(userDesk -> {
            userDeskRepo.delete(userDesk);
        });
        deskRepo.delete(deskToDelete);
    }
    /**
     * Получение all effective task of desk list.
     *
     * Входной параметр: deskId Определенный проект id
     * @return the list
     */
    public List<Task> getAllEffectiveTaskOfDesk(@PathVariable Long deskId){
        Desk deskDB = deskRepo.findDeskById(deskId);
        List<Task> allTaskOfDesk = new ArrayList<>();

        List<TaskList> taskLists = taskListRepo.findTaskListByDesk(deskDB);
        taskLists.forEach(taskList -> {
            allTaskOfDesk.addAll(taskRepo.findTaskByTaskList(taskList));
        });

        Task taskFhantom = taskRepo.findTaskByHeaderTaskList(2);
        List<Task> header1Tasks = allTaskOfDesk.stream().filter(task ->
task.getHeaderTaskList() != null && task.getHeaderTaskList() ==
1).collect(Collectors.toList());
        taskFhantom.setTaskRelation(header1Tasks);
        allTaskOfDesk.add(taskFhantom);
        effectiveWalkService.BFS(taskFhantom,10);
        return allTaskOfDesk;
    }
}
```

ПРИЛОЖЕНИЕ Ш

ИСХОДНЫЙ КОД СЕРВИСА EFFECTIVE WALK SERVICE

Листинг Ш.1 – Класс сервиса оптимального выполнения задач

```
/**
 * Класс Effective walk service.
 */
@Service
public class EffectiveWalkService {
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private TaskListRepo taskListRepo;

    /**
     * Создание графа.
     *
     * Входной параметр: taskListId the task list id
     */
    public void createGraph(Long taskListId){
        TaskList taskList = taskListRepo.findTaskListById(taskListId);
        List<Task> tasks = taskRepo.findTaskByTaskList(taskList);

        Task headerTask = null;
        for (Task task: tasks) {
            if (task.getHeaderTaskList() == 1){
                headerTask = task;
                break;
            }
        }

        List<TaskNode> taskNodes = new ArrayList<>();
        //getSumOfChildAndCreateNodes(headerTask.getTaskRelation());
    }

    /**
     * Реализация алгоритма обхода в ширину.
     *
     * Входной параметр: headerTask the header task
     * Входной параметр: userNum    Определенный пользователь num
     */
    public void BFS(Task headerTask, int userNum){
        int stage = 0,currentStage = 0;
        ArrayDeque<Task> taskQueue = new ArrayDeque<>();
        List<Task> stageBuffer = new ArrayList<>();
        List<Task> allTasks = getAllTaskOf-
List(headerTask.getTaskList().getId());
        headerTask.setStage(stage);
        taskQueue.addLast(headerTask);

        while (!taskQueue.isEmpty()) {
            Task currentTask = taskQueue.pop();
            stageBuffer.add(currentTask); //???
            if (currentTask.getStage() == currentStage){
                currentStage = currentStage + 1;
            }


            for (Task childTask:
                 currentTask.getTaskRelation()) {
```

```
                if (checkDouble(childTask,taskQueue) || isRelatedWithAnoth-
er(childTask,currentTask,getAllTaskOfList(childTask.getTaskList().getId())))
continue;
                childTask.setStage(currentStage);
                taskQueue.addLast(childTask);
            }

            if(!taskQueue.isEmpty() && taskQueue.getFirst().getStage() > cur-
rentTask.getStage() || taskQueue.isEmpty()){
                processNodes(stageBuffer,userNum);
                stageBuffer.clear();
            }
        }
    }

    private boolean checkDouble(Task taskCheck,ArrayDeque<Task> taskQueue){
        for (Task task:
             taskQueue) {
            if (task.getId() == taskCheck.getId()) return true;
        }
        return false;
    }
    private void processNodes(List<Task> buffer, int userNum){
        float sumComplexity = 0;
        for (Task task:
             buffer) {
            sumComplexity += task.getComplexity();
        }

        for (Task task:
             buffer) {
            task.setUserComplexity(Math.round(task.getComplexity()/sumComplexity
* userNum));
        }
    }

    private List<Task> getAllTaskOfList(Long taskListId){
        TaskList taskList = taskListRepo.findTaskListById(taskListId);
        List<Task> allTasks = taskRepo.findTaskByTaskList(taskList);
        return allTasks;
    }
    private boolean isRelatedWithAnother(Task childTask, Task task, List<Task>
taskList){
        for (Task anotherTask: taskList) {
            if (anotherTask.getId() == task.getId()) continue;
            for (Task test:
                 anotherTask.getTaskRelation()) {
                if (test.getId() == childTask.getId() && (anotherTask.getStage()
== null || anotherTask.getStage() > task.getStage())) return true;
            }
        }
        return false;
    }

}
```

ПРИЛОЖЕНИЕ Щ

ИСХОДНЫЙ КОД СЕРВИСА USER RELIABILITY SERVICE

Листинг Щ.1 – Класс сервиса вычисления оценки реальности выполнения задачи

```
/**
 * Класс User reliability service.
 */
@Service
public class UserReliabilityService {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private TaskListRepo taskListRepo;
    @Autowired
    private UserTaskRepo userTaskRepo;


    /**
     * Получение надежности выполнения задачи.
     *
     * Входной параметр: users          Определенный пользователь
     * Входной параметр: taskListId     the task list id
     * Входной параметр: taskComplexity the task complexity
     * Входной параметр: daysForDone    the days for done
     * @return the double
     */
    public double getProcentageForUsers(List<User> users, Long taskListId, int
taskComplexity, int daysForDone){
        double sumReliability = getUserListReliability(users, taskListId);

        double userProcentage = ((sumReliability * daysForDone) /
(Double.valueOf(taskComplexity))) * 100;

        return userProcentage;
    }

    /**
     * Получение user list reliability double.
     *
     * Входной параметр: users      Определенный пользователь
     * Входной параметр: taskListId the task list id
     * @return the double
     */
    public double getUserListReliability(List<User> users, Long taskListId){
        double sumReliability = 0;

        if (users.size() > 0) {
            for (User user :
                    users) {
                double userReliability = getUserReliability(user.getId(), tas-
kListId);
                if (userReliability == -1) continue;
                sumReliability += userReliability;
            }

            sumReliability /= users.size();
```

```
        }
        return sumReliability;
    }

    /**
     * Получение надежность определенного пользователя.
     *
     * Входной параметр: userId     Определенный пользователь id
     * Входной параметр: taskListId the task list id
     * @return the double
     */
    public double getUserReliability(String userId, Long taskListId){
        User user = userRepo.findUserById(userId);
        TaskList taskList = taskListRepo.findTaskListById(taskListId);

        List<Task> tasksFromList = taskRepo.findTaskByTaskList(taskList);
        if (tasksFromList.size() < 1) return -1;

        List<UserTask> userTasks = userTaskRepo.findUserTaskByUser(user);
        if (userTasks.size() < 1) return -1;

        List<Task> tasksFromUser = new ArrayList<>();
        for (UserTask userTask: userTasks) {
            tasksFromUser.add(userTask.getTask());
        }

        //ищем пересечение
        if (!tasksFromList.retainAll(tasksFromUser)) return -1;

        double userTaskRelySum = 0;
        for (Task task:
             tasksFromList) {
            if (task.getStatus() == null || !task.getStatus().equals("IS_DONE"))
continue; //TODO просмотреть вариант с теми, которые еще не доделаны

            int coeff = task.getComplexity();

            Period period = Pe-
riod.between(convertToLocalDate(task.getStart_time()),convertToLocalDate(task.ge
tTime_complete()));
            int diff = period.getDays();

            if (diff < 1) diff = 1;
            userTaskRelySum += coeff / Double.valueOf(diff) * us-
er.getValueCoeff();
        }

        userTaskRelySum =  userTaskRelySum/
Double.valueOf(tasksFromList.size());

        return userTaskRelySum;
    }

    private LocalDate convertToLocalDate(Date dateToConvert) {
        return dateToConvert.toInstant()
                .atZone(ZoneId.systemDefault())
                .toLocalDate();
    }
}
```

ПРИЛОЖЕНИЕ Э

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА USER

Листинг Э.1 – Класс контроллера сущности пользователя

```
/**
 * Класс User rest controller.
 */
@RestController
@RequestMapping("rest/user")
public class UserRestController {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private GroupRepo groupRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<User> findAll(){
        return userRepo.findAll();
    }
    /**
     * Нахождение через id user.
     *
     * Входной параметр: id the id
     * @return Определенный пользователь
     */
    @GetMapping("{id}")
    public User findById(@PathVariable String id){
        return userRepo.findAllById(id);
    }
    /**
     * Сохранение user user.
     *
     * Входной параметр: user Определенный пользователь
     * @return Определенный пользователь
     */
    @PostMapping
    public User saveUser(@RequestBody User user){
        return userRepo.save(user);
    }
    /**
     * Создание role user.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: role   Определенная роль
     * @return Определенный пользователь
     */
    @GetMapping("{userId}/role/{role}")
    public User setRole(@PathVariable String userId, @PathVariable String role){
        User user = userRepo.findUserById(userId);
        user.getRoles().clear();
        user.getRoles().add(Role.valueOf(role));
        return userRepo.save(user);
    }

}
```

ПРИЛОЖЕНИЕ Ю
ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА GROUP

Листинг Ю.1 – Класс контроллера сущности группы

```
/**
 * Класс Group rest controller.
 */
@RestController
@RequestMapping("rest/group")
public class GroupRestController {
    @Autowired
    private GroupRepo groupRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<Group> findAll(){
        return groupRepo.findAll();
    }

    /**
     * Нахождение через id group.
     *
     * Входной параметр: id the id
     * @return Определенная группа
     */
    @GetMapping("{id}")
    public Group findById(@PathVariable Long id){
        return groupRepo.findGroupById(id);
    }

    /**
     * Добавление group.
     *
     * Входной параметр: group Определенная группа
     * @return Определенная группа
     */
    @PostMapping
    public Group add(@RequestBody Group group){
        groupRepo.save(group);
        return group;
    }

    /**
     * Удаление.
     *
     * Входной параметр: id the id
     */
    @DeleteMapping("{id}")
    public void delete(@PathVariable Long id){
        groupRepo.deleteById(id);
    }

}
```

ПРИЛОЖЕНИЕ Я

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА USERGROUP

Листинг Я.1 – Класс контроллера сущности связи пользователя и группы

```
/**
 * Класс User group rest controller.
 */
@RestController
@RequestMapping("rest/user-group")
public class UserGroupRestController {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private GroupRepo groupRepo;

    @Autowired
    private UserGroupRepo userGroupRepo;

    /**
     * Получение all user groups list.
     *
     * @return the list
     */
    @GetMapping
    public List<UserGroup> getAllUserGroups(){
        return userGroupRepo.findAll();
    }

    /**
     * Получение user group user group.
     *
     * Входной параметр: userId  Определенный пользователь id
     * Входной параметр: groupId Определенная группа id
     * @return Определенный пользователь group
     */
    @GetMapping("get")
    public UserGroup getUserGroup(@RequestParam String userId, @RequestParam
Long groupId){
        User user = userRepo.findAllById(userId);
        Group group = groupRepo.findGroupById(groupId);
        return userGroupRepo.findUserGroupByUserAndGroup(user,group);
    }

    /**
     * Получение через user id list.
     *
     * Входной параметр: userId Определенный пользователь id
     * @return the list
     */
    @GetMapping("user/{userId}")
    public List<UserGroup> getByUserId(@PathVariable String userId){
        User user = userRepo.findAllById(userId);
        return userGroupRepo.findUserGroupsByUser(user);
    }

    /**
     * Получение через group id list.
     *
     * Входной параметр: groupId Определенная группа id
     * @return the list
```

```
     */
    @GetMapping("group/{groupId}")
    public List<UserGroup> getByGroupId(@PathVariable Long groupId){
        Group group = groupRepo.findGroupById(groupId);
        return userGroupRepo.findUserGroupsByGroup(group);
    }

    /**
     * Создание relation.
     *
     * Входной параметр: userId  Определенный пользователь id
     * Входной параметр: groupId Определенная группа id
     * Входной параметр: role    Определенная роль
     */
    @GetMapping("set")
    public void setRelation(@RequestParam String userId, @RequestParam Long
groupId, @RequestParam String role){
        User user = userRepo.findAllById(userId);
        Group group = groupRepo.findGroupById(groupId);
        UserGroup userGroupDB = userGroupRe-
po.findUserGroupByUserAndGroup(user,group);
        if (userGroupDB == null){
            UserGroup userGroup = new UserGroup(user,group,role);
            userGroupRepo.save(userGroup);
        }else{
            userGroupDB.setRole(role);
            userGroupRepo.save(userGroupDB);
        }
    }

    /**
     * Удаление relation.
     *
     * Входной параметр: userId  Определенный пользователь id
     * Входной параметр: groupId Определенная группа id
     */
    @DeleteMapping("delete")
    public void deleteRelation(@RequestParam String userId, @RequestParam Long
groupId){
        User user = userRepo.findAllById(userId);
        Group group = groupRepo.findGroupById(groupId);
        UserGroup userGroup = userGroupRe-
po.findUserGroupByUserAndGroup(user,group);
        userGroupRepo.delete(userGroup);
    }

}
```

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА DESK

Листинг А.1 – Класс контроллера сущности проекта

```
/**
 * Класс Desk rest controller.
 */
@RestController
@RequestMapping("rest/desk")
public class DeskRestController {
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private TaskListRepo taskListRepo;
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private DeskService deskService;
    /**
     * Нахождение всех desks list.
     *
     * @return the list
     */
    @GetMapping
    public List<Desk> findAllDesks(){
        return deskRepo.findAll();}
    /**
     * Нахождение desk через id desk.
     *
     * Входной параметр: id the id
     * @return Определенный проект
     */
    @GetMapping("{id}")
    public Desk findDeskById(@PathVariable Long id){
        return deskRepo.findDeskById(id);}
    /**
     * Добавление desk desk.
     *
     * Входной параметр: desk Определенный проект
     * @return Определенный проект
     */
    @PostMapping
    public Desk addDesk(@RequestBody Desk desk){
        deskRepo.save(desk);
        return desk;}
    /**
     * Удаление.
     *
     * Входной параметр: id the id
     */
    @DeleteMapping("{id}")
    public void delete(@PathVariable Long id){
        deskRepo.deleteById(id);}
    /**
     * Получение all tasks of desk list.
     *
     * Входной параметр: deskId Определенный проект id
     * @return the list
     */
    @GetMapping("{deskId}/allTasks")
```

```
    @ResponseBody
    public List<Task> getAllTasksOfDesk(@PathVariable Long deskId){
        List<Task> allTaskOfDesk = new ArrayList<>();
        Desk deskDB = findDeskById(deskId);
        List<TaskList> taskListsOfDesk = taskListRe-
po.findTaskListByDesk(deskDB);
        for (TaskList tasklist: taskListsOfDesk) {
            List<Task> tasksOfTaskList = taskRepo.findTaskByTaskList(tasklist);
            allTaskOfDesk.addAll(tasksOfTaskList);
        }
        Task header = null;
        for (Task task:
             allTaskOfDesk) {
            if (task.getHeaderTaskList() != null) header = task;
        }
        //effectiveWalkService.BFS(header,10);
        return allTaskOfDesk;
    }
    /**
     * Merge projects.
     *
     * Входной параметр: newDesk the new desk
     * Входной параметр: deskIds Определенный проект ids
     */
    @PostMapping("merge")
    public void mergeProjects(@RequestBody Desk newDesk, @RequestParam
List<Long> deskIds){
        Desk deskDB = deskRepo.save(newDesk);
        deskService.mergeProjects(deskIds, deskDB);
    }
    /**
     * Установка desk rel.
     *
     * Входной параметр: masterDeskId the master desk id
     * Входной параметр: slaveDeskId  the slave desk id
     */
    @GetMapping("setRel")
    public void setDeskRel(@RequestParam(name = "masterDeskId") Long masterDesk-
Id,@RequestParam(name = "slaveDeskId")Long slaveDeskId){
        Desk masterDesk = deskRepo.findDeskById(masterDeskId);
        Desk slaveDesk = deskRepo.findDeskById(slaveDeskId);
        masterDesk.getDeskRelation().add(slaveDesk);
        deskRepo.save(masterDesk);
    }
    /**
     * Full delete desk.
     *
     * Входной параметр: deskId Определенный проект id
     */
    @GetMapping("fullDelete/{deskId}")
    public void fullDeleteDesk(@PathVariable Londesk-
Id){deskService.deleteDesk(deskId);}
    /**
     * Получение all effective taskof desk list.
     *
     * Входной параметр: deskId Определенный проект id
     * @return the list
     */
    @GetMapping("{deskId}/allEffectiveTasks")
    @ResponseBody
    public List<Task> getAllEffectiveTaskofDesk(@PathVariable Long deskId){
        return deskService.getAllEffectiveTaskOfDesk(deskId);}}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА TASK

Листинг В.1 – Класс контроллера сущности задачи

```
/**
 * Класс Task rest controller.
 */
@RestController
@RequestMapping("rest/task")
public class TaskRestController {
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private TaskListRepo taskListRepo;

    /**
     * Нахождение всех tasks list.
     *
     * @return the list
     */
    @GetMapping
    public List<Task> findAllTasks(){
        return taskRepo.findAll();
    }

    /**
     * Нахождение task bi id task.
     *
     * Входной параметр: id the id
     * @return the task
     */
    @GetMapping("{id}")
    public Task findTaskBiId(@PathVariable Long id){
        return taskRepo.findTaskById(id);
    }

    /**
     * Нахождение task через task list list.
     *
     * Входной параметр: tasklistid the tasklistid
     * @return the list
     */
    @GetMapping("tasklist/{tasklistid}")
    public List<Task> findTaskByTaskList(@PathVariable Long tasklistid){
        TaskList taskList = taskListRepo.findTaskListById(tasklistid);
        return taskRepo.findTaskByTaskList(taskList);
    }

    /**
     * Добавление task task.
     *
     * Входной параметр: task the task
     * @return the task
     */
    @PostMapping
    public Task addTask(@RequestBody Task task){
        taskRepo.save(task);
        return task;
    }
```

```
    /**
     * Удаление.
     *
     * Входной параметр: id the id
     */
    @DeleteMapping("{id}")
    public void delete(@PathVariable Long id){
        taskRepo.deleteById(id);
    }

    /**
     * Получение parent task list list.
     *
     * Входной параметр: id the id
     * @return the list
     */
    @GetMapping("{id}/parent")
    public List<Task> getParentTaskList(@PathVariable Long id){
        return findTaskBiId(id).getTaskRelation();
    }

    /**
     * Установка task rel.
     *
     * Входной параметр: taskId  the task id
     * Входной параметр: childId the child id
     */
    @GetMapping("setRel")
    public void setTaskRel(@RequestParam(name = "taskId") Long task-
Id,@RequestParam(name = "childId") Long childId){
        Task taskDB = taskRepo.findTaskById(taskId);
        Task taskChild = taskRepo.findTaskById(childId);

        taskDB.getTaskRelation().add(taskChild);
        System.out.println(taskDB);
        System.out.println(taskChild);
        taskRepo.save(taskDB);
    }
}
```

ПРИЛОЖЕНИЕ С

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА TASKLIST

Листинг С.1 – Класс контроллера сущности списка задач

```
/**
 * Класс Task list rest controller.
 */
@RestController
@RequestMapping("rest/tasklist")
public class TaskListRestController {
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private TaskListRepo taskListRepo;
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private EffectiveWalkService effectiveWalkService;

    /**
     * Нахождение всех task lists list.
     *
     * @return the list
     */
    @GetMapping
    public List<TaskList> findAllTaskLists(){
        return taskListRepo.findAll();
    }

    /**
     * Нахождение task list bi id task list.
     *
     * Входной параметр: id the id
     * @return the task list
     */
    @GetMapping("{id}")
    public TaskList findTaskListBiId(@PathVariable Long id){
        return taskListRepo.findTaskListById(id);
    }

    /**
     * Нахождение task list через desk id list.
     *
     * Входной параметр: deskid Определенный проектid
     * @return the list
     */
    @GetMapping("desk/{deskid}")
    public List<TaskList> findTaskListByDeskId(@PathVariable Long deskid){
        Desk desk = deskRepo.findDeskById(deskid);
        return taskListRepo.findTaskListByDesk(desk);
    }

    /**
     * Добавление task list task list.
     *
     * Входной параметр: task the task
     * @return the task list
     */
    @PostMapping
    public TaskList addTaskList(@RequestBody TaskList task){
```

```
        System.out.println(task);
        taskListRepo.save(task);
        return task;

    }

    /**
     * Удаление.
     *
     * Входной параметр: id the id
     */
    @DeleteMapping("{id}")
    public void delete(@PathVariable Long id){
        taskListRepo.deleteById(id);
    }

    /**
     * Получение all tasks of list list.
     *
     * Входной параметр: tasklistId the tasklist id
     * @return the list
     */
    @GetMapping("{tasklistId}/getAllTasksOfList")
    @ResponseBody
    public List<Task> getAllTasksOfList(@PathVariable Long tasklistId){
        TaskList taskListDB = findTaskListBiId(tasklistId);
        List<Task> allTaskOfList = taskRepo.findTaskByTaskList(taskListDB);
        return allTaskOfList;
    }

    /**
     * Получение all tasks of desk 2 list.
     *
     * Входной параметр: tasklistId the tasklist id
     * @return the list
     */
    @GetMapping("{tasklistId}/allEffectiveTasks")
    @ResponseBody
    public List<Task> getAllTasksOfDesk2(@PathVariable Long tasklistId){
        TaskList taskListDB = findTaskListBiId(tasklistId);
        List<Task> allTaskOfList = taskRepo.findTaskByTaskList(taskListDB);

        Task header = null;
        for (Task task:
                allTaskOfList) {
            if (task.getHeaderTaskList() != null) header = task;
        }

        effectiveWalkService.BFS(header,10);
        return allTaskOfList;
    }
}
```

ПРИЛОЖЕНИЕ D

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА GROUPDESK

Листинг D.1 – Класс контроллера сущности связи группы и проекта

```
/**
 * Класс Group desk controller.
 */
@RestController
@RequestMapping("rest/group-desk")
public class GroupDeskController {
    @Autowired
    private GroupRepo groupRepo;
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private GroupDeskRepo groupDeskRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<GroupDesk> findAll(){
        return groupDeskRepo.findAll();
    }

    /**
     * Нахождение всех desks через group list.
     *
     * Входной параметр: groupId Определенная группа id
     * @return the list
     */
    @GetMapping("group/{groupId}")
    public List<GroupDesk> findAllDesksByGroup(@PathVariable Long groupId){
        Group group = groupRepo.findGroupById(groupId);
        return groupDeskRepo.findGroupDeskByGroup(group);
    }

    /**
     * Нахождение всех groups через desk list.
     *
     * Входной параметр: deskId Определенный проект id
     * @return the list
     */
    @GetMapping("desk/{deskId}")
    public List<GroupDesk> findAllGroupsByDesk(@PathVariable Long deskId){
        Desk desk = deskRepo.findDeskById(deskId);
        return groupDeskRepo.findGroupDeskByDesk(desk);
    }
```

```
    /**
     * Создание relation.
     *
     * Входной параметр: groupId Определенная группа id
     * Входной параметр: deskId  Определенный проект id
     * Входной параметр: role    Определенная роль
     */
    @GetMapping("set")

    public void setRelation(@RequestParam Long groupId, @RequestParam Long desk-
Id, @RequestParam String role){
        Group group = groupRepo.findGroupById(groupId);
        Desk desk = deskRepo.findDeskById(deskId);
        GroupDesk groupDeskDB = groupDeskRe-
po.findGroupDeskByGroupAndDesk(group,desk);
        if (groupDeskDB == null){
            GroupDesk groupDesk = new GroupDesk(group,desk,role);
            groupDeskRepo.save(groupDesk);
        }else {
            groupDeskDB.setRole(role);
            groupDeskRepo.save(groupDeskDB);
        }
    }

    /**
     * Удаление relation.
     *
     * Входной параметр: groupId Определенная группа id
     * Входной параметр: deskId  Определенный проект id
     */
    @GetMapping("delete")
    public void deleteRelation(@RequestParam Long groupId, @RequestParam Long
deskId){
        Group group = groupRepo.findGroupById(groupId);
        Desk desk = deskRepo.findDeskById(deskId);
        groupDeskRe-
po.delete(groupDeskRepo.findGroupDeskByGroupAndDesk(group,desk));
    }

}
```

ПРИЛОЖЕНИЕ F

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА USERDESK

Листинг F.1 – Класс контроллера сущности связи пользователя и проекта

```
/**
 * Класс User desk controller.
 */
@RestController
@RequestMapping("rest/user-desk")
public class UserDeskController {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private DeskRepo deskRepo;
    @Autowired
    private UserDeskRepo userDeskRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<UserDesk> findAll(){
        return userDeskRepo.findAll();
    }

    /**
     * Нахождение user desk user desk.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: deskId Определенный проект id
     * @return Определенный пользователь desk
     */
    @GetMapping("get")
    public UserDesk findUserDesk(@RequestParam String userId, @RequestParam Long
deskId){
        User user = userRepo.findAllById(userId);
        Desk desk = deskRepo.findDeskById(deskId);
        return userDeskRepo.findUserDeskByUserAndDesk(user, desk);
    }

    /**
     * Нахождение всех users через desk list.
     *
     * Входной параметр: deskId Определенный проект id
     * @return the list
     */
    @GetMapping("desk/{deskId}")
    public List<UserDesk> findAllUsersByDesk(@PathVariable Long deskId){
        Desk desk = deskRepo.findDeskById(deskId);
        return userDeskRepo.findUserDesksByDesk(desk);
    }

    /**
     * Нахождение всех desks через user list.
     *
     * Входной параметр: userId Определенный пользователь id
     * @return the list
     */
```

```
    @GetMapping("user/{userId}")
    public List<UserDesk> findAllDesksByUser(@PathVariable String userId){
        User user = userRepo.findAllById(userId);
        return userDeskRepo.findUserDesksByUser(user);
    }

    /**
     * Создание relation.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: deskId Определенный проект id
     * Входной параметр: role   Определенная роль
     */
    @GetMapping("set")
    public void setRelation(@RequestParam String userId, @RequestParam Long
deskId, @RequestParam String role){
        User user = userRepo.findAllById(userId);
        Desk desk = deskRepo.findDeskById(deskId);
        UserDesk userDeskDB = userDeskRepo.findUserDeskByUserAndDesk(user,desk);
        if (userDeskDB == null){
            UserDesk userDesk = new UserDesk(user,desk,role);
            userDeskRepo.save(userDesk);
        }else {
            userDeskDB.setRole(role);
            userDeskRepo.save(userDeskDB);
        }
    }

    /**
     * Удаление relation.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: deskId Определенный проект id
     */
    @GetMapping("delete")
    public void deleteRelation(@RequestParam String userId, @RequestParam Long
deskId){
        User user = userRepo.findAllById(userId);
        Desk desk = deskRepo.findDeskById(deskId);
        userDeskRepo.delete(userDeskRepo.findUserDeskByUserAndDesk(user,desk));
    }
}
```

ПРИЛОЖЕНИЕ G

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА USERTASK

Листинг G.1 – Класс контроллера сущности связи пользователя и задачи

```
/**
 * Класс User task controller.
 */
@RestController
@RequestMapping("rest/user-task")
public class UserTaskController {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private TaskRepo taskRepo;
    @Autowired
    private UserTaskRepo userTaskRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<UserTask> findAll(){
        return userTaskRepo.findAll();
    }

    /**
     * Нахождение user task user task.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: taskId the task id
     * @return Определенный пользователь task
     */
    @GetMapping("get")
    public UserTask findUserTask(@RequestParam String userId, @RequestParam Long
taskId){
        User user = userRepo.findAllById(userId);
        Task task = taskRepo.findTaskById(taskId);
        return userTaskRepo.findUserTaskByUserAndTask(user,task);
    }

    /**
     * Нахождение всех users через task list.
     *
     * Входной параметр: taskId the task id
     * @return the list
     */
    @GetMapping("task/{taskId}")
    public List<UserTask> findAllUsersByTask(@PathVariable Long taskId){
        Task task = taskRepo.findTaskById(taskId);
        return userTaskRepo.findUserTaskByTask(task);
    }

    /**
     * Нахождение всех tasks через user list.
     *
     * Входной параметр: userId Определенный пользователь id
     * @return the list
     */
```

```
    @GetMapping("user/{userId}")
    public List<UserTask> findAllTasksByUser(@PathVariable String userId){
        User user = userRepo.findAllById(userId);
        return userTaskRepo.findUserTaskByUser(user);
    }

    /**
     * Создание relation.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: taskId the task id
     * Входной параметр: role   Определенная роль
     */
    @GetMapping("set")
    public void setRelation(@RequestParam String userId, @RequestParam Long
taskId, @RequestParam String role){
        User user = userRepo.findAllById(userId);
        Task task = taskRepo.findTaskById(taskId);
        UserTask userTaskDB = userTaskRepo.findUserTaskByUserAndTask(user,task);
        if (userTaskDB == null){
            UserTask userTask = new UserTask(user,task,role);
            userTaskRepo.save(userTask);
        }else {
                userTaskDB.setRole(role);
                userTaskRepo.save(userTaskDB);

        }
    }

    /**
     * Удаление relation.
     *
     * Входной параметр: userId Определенный пользователь id
     * Входной параметр: taskId the task id
     */
    @DeleteMapping("delete")
    public void deleteRelation(@RequestParam String userId, @RequestParam Long
taskId){
        User user = userRepo.findAllById(userId);
        Task task = taskRepo.findTaskById(taskId);
        userTaskRepo.delete(userTaskRepo.findUserTaskByUserAndTask(user,task));
    }
}
```

ПРИЛОЖЕНИЕ Н

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА USERTASKLIST

Листинг Н.1 – Класс контроллера сущности связи пользователя и списка

```
/**
 * Класс User task list controller.
 */
@RestController
@RequestMapping("rest/user-tasklist")
public class UserTaskListController {
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private TaskListRepo taskListRepo;
    @Autowired
    private UserTasklistRepo userTasklistRepo;

    /**
     * Нахождение всех list.
     *
     * @return the list
     */
    @GetMapping
    public List<UserTasklist> findAll(){
        return userTasklistRepo.findAll();
    }

    /**
     * Нахождение user task list user tasklist.
     *
     * Входной параметр: userId     Определенный пользователь id
     * Входной параметр: tasklistId the tasklist id
     * @return Определенный пользователь tasklist
     */
    @GetMapping("get")
    public UserTasklist findUserTaskList(@RequestParam String userId,
@RequestParam Long tasklistId){
        User user = userRepo.findAllById(userId);
        TaskList taskList = taskListRepo.findTaskListById(tasklistId);
        return userTasklistRe-
po.findUserTasklistByUserAndTaskList(user,taskList);
    }

    /**
     * Нахождение всех users через tasklist list.
     *
     * Входной параметр: tasklistId the tasklist id
     * @return the list
     */
    @GetMapping("tasklist/{tasklistId}")
    public List<UserTasklist> findAllUsersByTasklist(@PathVariable Long tasklis-
tId){
        TaskList taskList = taskListRepo.findTaskListById(tasklistId);
        return userTasklistRepo.findUserTasklistByTaskList(taskList);
    }

    /**
     * Нахождение всех tasklists через user list.
     *
     * Входной параметр: userId Определенный пользователь id
```

```
     * @return the list
     */
    @GetMapping("user/{userId}")
    public List<UserTasklist> findAllTasklistsByUser(@PathVariable String use-
rId){
        User user = userRepo.findAllById(userId);
        return userTasklistRepo.findUserTasklistByUser(user);
    }

    /**
     * Создание relation.
     *
     * Входной параметр: userId     Определенный пользователь id
     * Входной параметр: tasklistId the tasklist id
     * Входной параметр: role       Определенная роль
     */
    @GetMapping("set")
    public void setRelation(@RequestParam String userId, @RequestParam Long tas-
klistId, @RequestParam String role){
        User user = userRepo.findAllById(userId);
        TaskList taskList = taskListRepo.findTaskListById(tasklistId);
        UserTasklist userTasklistDB = userTasklistRe-
po.findUserTasklistByUserAndTaskList(user,taskList);
        if (userTasklistDB == null){
            UserTasklist userTasklist = new UserTasklist(user,taskList,role);
            userTasklistRepo.save(userTasklist);
        }else {
                userTasklistDB.setRole(role);
                userTasklistRepo.save(userTasklistDB);

        }
    }

    /**
     * Удаление relation.
     *
     * Входной параметр: userId     Определенный пользователь id
     * Входной параметр: tasklistId the tasklist id
     */
    @DeleteMapping("delete")
    public void deleteRelation(@RequestParam String userId, @RequestParam Long
tasklistId){
        User user = userRepo.findAllById(userId);
        TaskList taskList = taskListRepo.findTaskListById(tasklistId);
        userTasklistRe-
po.delete(userTasklistRepo.findUserTasklistByUserAndTaskList(user,taskList));
    }
}
```

ПРИЛОЖЕНИЕ J

ИСХОДНЫЙ КОД REST-КОНТРОЛЛЕРА UTILITYREST

Листинг J.1 – Класс контроллера для выполнения сервисных функций

```
/**
 * Класс Utility rest.
 */
@RestController
@RequestMapping("rest/utility")
public class UtilityRest {
    @Autowired
    private UserReliabilityService userReliabilityService;
    @Autowired
    private UserRepo userRepo;

    /**
     * Calc reliability users double.
     *
     * Входной параметр: userIds        Определенный пользователь ids
     * Входной параметр: taskListId     the task list id
     * Входной параметр: taskComplexity the task complexity
     * Входной параметр: daysForDone    the days for done
     * @return the double
     */
    @GetMapping("reliability")
    public double calcReliabilityUsers(@RequestParam List<String> userIds, Long
taskListId, int taskComplexity, int daysForDone){
        List<User> userList = new ArrayList<>();
        for (String id:
             userIds) {
            userList.add(userRepo.findAllById(id));
        }

        return userReliabilitySer-
vice.getProcentageForUsers(userList,taskListId,taskComplexity,daysForDone);
    }

}
```

ПРИЛОЖЕНИЕ К
ИСХОДНЫЙ КОД ТОЧКИ ВХОДА В ПРОГРАММУ

Листинг К.1 – Класс точки входа в приложение

```
/**
 * Класс Sarafan application.
 */
@SpringBootApplication
public class SarafanApplication {

	/**
	 * Входная точка в работу приложения.
	 *
	 * Входной параметр: args the input arguments
	 */
	public static void main(String[] args) {
		SpringApplication.run(SarafanApplication.class, args);
	}
}
```

ПРИЛОЖЕНИЕ L

ИСХОДНЫЙ КОД ПРИМЕРА VUE ФАЙЛА TASKLISTEDIT

Листинг L.1 – Пример Vue файла TaskListEdit

```
<template xmlns:v-slot="http://www.w3.org/1999/XSL/Transform">
    <div id="app">
        <v-container grid-list-md>
            <v-layout>
                <v-tooltip bottom>
                    <template v-slot:activator="{ on }">
                        <v-btn flat icon color="red darken-3" :to="'/' +
$route.params.deskId" v-on="on">
                            <v-icon>reply_all</v-icon>
                        </v-btn>
                    </template>
                    <span>В меню списков</span>
                </v-tooltip>
                <span class="display-2 font-weight-medium">Основная информация о
списке</span>
            </v-layout>
            <v-layout v-layout align-space-between justify-start column fill-
height>
                <v-flex>
                    <v-card dark color="blue lighten-2">
                        <v-card-title primary class="title">Основная информация
о списке "{{current_list.name}}" </v-card-title>
                        <v-card-text>
                            <v-layout align-start justify-start column fill-
height>
                                <span class="title">Название: {{cur-
rent_list.name}}</span>
                                <span class="title">Описание: {{cur-
rent_list.discription}}</span>
                                <span class="title">Дата начала выполнения:
{{current_list.start_time}}</span>
                                <span class="title">Дата окончания выполнения:
{{current_list.end_time}}</span>
                                <span class="title">Ответственный: {{assignedUs-
er.name}}</span>
                            </v-layout>
                        </v-card-text>
                    </v-card>
                </v-flex>
                <v-flex>
                    <v-card dark color="blue lighten-2" v-
if="currentUserTaskList.role.includes('ASGN')">
                        <v-card-title primary class="title">Изменить свойства
списка</v-card-title>
                        <v-card-text>
                            <v-layout align-start justify-start column fill-
height>
```

```
                                <v-text-field
                                        v-model="edited_current_list.name"
                                        :rules="nameRules"
                                        :counter="20"
                                        label="Наименование"
                                        required
                                ></v-text-field>
                                <v-text-field
                                        v-
model="edited_current_list.discription"
                                        :rules="nameRules"
                                        :counter="20"
                                        label="Описание"
                                        required
                                ></v-text-field>
                                <v-menu
                                        ref="menu"
                                        v-model="menu"
                                        :close-on-content-click="false"
                                        :nudge-right="40"
                                        :return-value.sync="date"
                                        lazy
                                        transition="scale-transition"
                                        offset-y
                                        full-width
                                        min-width="290px"
                                >
                                    <template v-slot:activator="{ on }">
                                        <v-text-field
                                                v-
model="edited_current_list.start_time"
                                                label="Выбор начала периода"
                                                prepend-icon="event"
                                                readonly
                                                v-on="on"
                                        ></v-text-field>
                                    </template>
                                    <v-date-picker v-
model="edited_current_list.start_time" no-title scrollable>
                                        <v-spacer></v-spacer>
                                        <v-btn flat color="primary" @click="menu
= false">Отмена</v-btn>
                                        <v-btn flat color="primary"
@click="$refs.menu.save(date)">OK</v-btn>
                                    </v-date-picker>
                                </v-menu>
                                <v-menu
                                        ref="menu"
                                        v-model="menu"
                                        :close-on-content-click="false"
                                        :nudge-right="40"
                                        :return-value.sync="date"
                                        lazy
                                        transition="scale-transition"
                                        offset-y
```

```
                                        full-width
                                        min-width="290px"
                                    >
                                        <template v-slot:activator="{ on }">
                                        <v-text-field
                                                v-
model="edited_current_list.end_time"
                                                label="Выбор окончания периода"
                                                prepend-icon="event"
                                                readonly
                                                v-on="on"
                                        ></v-text-field>
                                        </template>
                                        <v-date-picker v-
model="edited_current_list.end_time" no-title scrollable>
                                        <v-spacer></v-spacer>
                                        <v-btn flat color="primary" @click="menu
= false">Отмена</v-btn>
                                        <v-btn flat color="primary"
@click="$refs.menu.save(date)">OK</v-btn>
                                    </v-date-picker>
                                </v-menu>
                                <v-select
                                        :items="all_desk_user"
                                        item-text="name"
                                        item-value="id"
                                        label="Уполномоченный"
                                        @change="changeUserAssign"
                                ></v-select>
                                <div v-if="isEdited">
                                    <v-btn small color="primary"
@click="acceptValue">Принять</v-btn>
                                </div>
                            </v-layout>
                        </v-card-text>
                    </v-card>
                </v-flex>
            </v-layout>
        </v-container>
    </div>
</template>

<script>
    import axios from 'axios';
    import _ from 'lodash';

    export default {
        data: () => ({
            errors: [],
            profile: frontendData.profile,
            currentUserTaskList: null,
            current_list: null,
            edited_current_list: null,
            nameRules: [
                v => v.length >= 2 || 'Имя должно быть больше 2 символов'
```

```
            ],
            isEdited: null,
            assignedUser: null,
            all_desk_user: [],
            selectedAsgnUser: null
        }),

        watch:{
            edited_current_list: {
                immediate: true,
                handler(newVal,oldVal){
                    if(oldVal != null) {
                        if (!this.isEdited) this.isEdited = true;
                    }
                },
                deep: true
            }
        },
        created() {
            axios.get('http://localhost:9000/rest/tasklist/' +
this.$route.params.id)
                .then(response => {
                    this.current_list = response.data;
                    this.edited_current_list =
JSON.parse(JSON.stringify(this.current_list));
                    this.edited_current_list.desk
={id:this.edited_current_list.desk};
                    this.isEdited = false;
                })
                .catch(e => {
                    this.errors.push(e)
                });

            axios.get('http://localhost:9000/rest/user-tasklist/tasklist/' +
this.$route.params.id)
                .then(response => {
                    response.data.forEach((obj) => {
                        if (obj.role.includes('ASGN')){
                            console.log('1');
                            console.log(obj.user.name);
                            this.assignedUser = obj.user;
                        }
                    })
                })
                .catch(e => {
                    this.errors.push(e)
                });

            axios.get('http://localhost:9000/rest/user-desk/desk/' +
this.$route.params.deskId)
                .then(response => {
                    response.data.forEach((obj)=> {
                        this.all_desk_user.push(obj.user);
                    })
                })
```

```
                .catch(e => {
                    this.errors.push(e)
                });

            axios.get('http://localhost:9000/rest/user-tasklist/get?userId=' +
this.profile.id + '&tasklistId=' + this.$route.params.id)
                .then(response => {
                    this.currentUserTaskList = response.data;
                    console.log("HEERE " + response.data);
                });

        },
        methods:{
            acceptValue: function () {
                axios.post('http://localhost:9000/rest/tasklist',
this.edited_current_list)
                    .then(response => {
                        this.current_list = response.data;
                        axios.get('http://localhost:9000/rest/user-
tasklist/set?userId=' + this.assignedUser.id + '&tasklistId=' +
this.current_list.id + '&role=USER')
                            .then(response => {
                                axios.get('http://localhost:9000/rest/user-
tasklist/set?userId=' + this.selectedAsgnUser + '&tasklistId=' +
this.current_list.id + '&role=ASGN')
                                    .then(response => {

axios.get('http://localhost:9000/rest/user-tasklist/tasklist/' +
this.$route.params.id)
                                            .then(response => {
                                                response.data.forEach((obj) => {
                                                    if
(obj.role.includes('ASGN')){
                                                        this.assignedUser =
obj.user;
                                                    }
                                                })
                                            })
                                            .catch(e => {
                                                this.errors.push(e)
                                            });
                                    });
                            });
                        this.isEdited=false;
                    })
                    .catch(e => {
                        this.errors.push(e)
                    });
            },
            changeUserAssign: function (val) {
                this.selectedAsgnUser = val;
            }
        }
    }
</script><style></style>
```

ПРИЛОЖЕНИЕ М

ИСХОДНЫЙ КОД ПРИМЕРА VUE ФАЙЛА GRAPH

Листинг М.1 – Пример Vue файла Graph

```
<template>
    <v-app id="inspire">
        <v-container grid-list-md>
            <v-layout row wrap>
                <v-flex xs12>
                    <div id="mynetwork" ref="mynetwork" class="mynetwork"></div>
                </v-flex>
            </v-layout>
        </v-container>
    </v-app>
</template>

<script>
    import vis from 'vis'
    import axios from 'axios'
    // create an array with nodes

    export default {
        components: {vis},
        data: () => ({
            nodes:null,
            edges: null,
            container: null,
            data: null,
            options: null,
            network:null,
            allTaskListOfList: null

        }),
        mounted(){
            axios.get('http://localhost:9000/rest/tasklist/' +
this.$route.params.tasklistid + '/allEffectiveTasks')
                .then(response => {
                    this.allTaskListOfList = response.data;

                    var array = [];
                    this.allTaskListOfList.forEach(function (task) {
                        var group;
                        if(task.userComplexity == 10) group = 'six';
                        else {
                            switch (task.stage) {
                                case 0:
                                    group = 'one'
                                    break;
                                case 1:
                                    group = 'two'
                                    break;
                                case 2:
                                    group = 'three'
```

```
                                break;
                            case 3:
                                group = 'four'
                                break;
                            case 4:
                                group = 'five'
                                break;
                        }
                    }
                    array.push({id:task.id, label: task.name +' | 
Сложность:'
                            + task.complexity + ' Рекомендуемая нагрузка: '
+ task.userComplexity, group:group});
                });
                this.nodes = new vis.DataSet(array);

                // create an array with edges
                var array2 = [];
                this.allTaskListOfList.forEach(function (task) {
                    task.taskRelation.forEach(function (rel) {
                        array2.push({from: task.id, to: rel.id, arrows:
'to'})
                    });
                });
                this.edges = new vis.DataSet(array2);

                // create a network
                this.data = {
                    nodes: this.nodes,
                    edges: this.edges
                };
                this.options = {
                    nodes: {
                        shape: 'box',
                        margin: 10,
                        widthConstraint: {
                            maximum: 200
                        },
                        borderWidth: 2,
                        shadow: true,
                        font: {size:20, face:'courier', strokeWidth:3}
                    },
                    physics: {
                        enabled: false
                    },
                    edges: {
                      width: 2
                    },
                    groups: {
                        one: {
                            color: {background: 'red'},
                            shape: 'box'
                        },
                        two: {
                            color: {background: 'blue'},
```

```
                                shape: 'box'
                            },
                            three: {
                                color: {background: 'green'},
                                shape: 'box'
                            },
                            four: {
                                color: {background: 'orange'},
                                shape: 'box'
                            },
                            five: {
                                color: {background: 'gray'},
                                shape: 'box'
                            },
                            six: {
                                color: {background: 'rgb(255,65,17)'},
                                shape: 'star'
                            },
                        }
                    };
                    this.container = document.getElementById('mynetwork');
                    this.network = new vis.Network(this.container, this.data,
this.options);
                })
                .catch(e => {
                    this.errors.push(e)
                });

        },

        created() {

        }
    }
</script>

<style>
    #mynetwork {
        height: 100vh;
    }
</style>
```