РОСЖЕЛДОР

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Ростовский государственный университет путей сообщения»

(ФГБОУ ВО РГУПС)

Допустить к защите в ГЭК

Зав. кафедрой «ВТ и АСУ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Игнатьева

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Разработка приложения с веб-интерфейсом для создания и

**мониторинга задач рабочей группы**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к выпускной квалификационной (бакалаврской) работе

АВБ 12.02.16 ПЗ

Направление подготовки 09.03.01 «Информатика и вычислительная техника», профиль «Вычислительные машины, комплексы, системы и сети»

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.А. Нарышкин

Руководитель работы

д.п.н., профессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.И. Соколова

Нормоконтроль

ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Р. Осипова

Научный консультант

к.т.н., научный сотрудник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Кулькин

2022

РОСЖЕЛДОР

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ростовский государственный университет путей сообщения»

(ФГБОУ ВО РГУПС)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра «ВТ и АСУ»

УТВЕРЖДАЮ

Зав.кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В.Игнатьева

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

ЗАДАНИЕ

на выпускную квалификационную(бакалаврскую) работу

Студенту \_Нарышкину Роману Андреевичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Группа АВБ-4-032

**1. Тема работы:** Разработка приложения с веб-интерфейсом для создания и

мониторинга задач рабочей группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом по университету № 95/ос от 26.01.2022 г.

Срок сдачи студентом законченной работы «9» июня 2022 г.

**2. Исходные данные к работе** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Литература по программированию на языке Kotlin\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Литература по веб-разработке на фреймворке Angular\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Литература по проектированию информационных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке)**

1. Анализ предметной области разрабатываемого приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Актуальность разрабатываемого приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Анализ существующих приложений для создания и мониторинга задач\_\_\_\_\_\_

4. Техническое задание\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Выбор технических средств для разрабатываемого приложения\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Проектирование приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Построение диаграммы прецедентов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Создание главного сценария взаимодействия в разрабатываемом приложении

9. Построение диаграммы классов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. Построение диаграммы последовательности\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11. Расчет FP-метрик, LOC-метрик и COCOMO-метрик\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12. Проектирование базы данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13. Разработка веб-приложения для создания и мониторинга задач рабочей группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

14. Разработка веб-интерфейса приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15. Создание руководства пользователя приложением\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Перечень графического материала (с точным указанием названий слайдов презентации)**

1. Титульный лист\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Цели и задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Средства разработки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Функционал приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Диаграмма прецедентов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Схема базы данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Демонстрация интерфейса. Форма авторизации и регистрации\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Демонстрация интерфейса. Главная страница\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Демонстрация интерфейса. Задачи пользователя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. Демонстрация интерфейса. Группы пользователя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11. Демонстрация интерфейса. Администрируемые группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12. Демонстрация интерфейса. Панель управления группой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13. Демонстрация интерфейса. Панель управления группой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

14. Демонстрация интерфейса. Панель управления группой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15. Заключение\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « 8 » февраля 2022 г.

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы, фамилия)

Задание принял к исполнению « 8 » февраля 2022 г.

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы, фамилия)

**Реферат**

Бакалаврская работа содержит 120 листов пояснительной записки, включающей 48 рисунков, 6 таблиц, 29 источника и 2 приложения.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ИНФОРМАТИКА, UML-ДИАГРАММА, ВЕБ-ПРИЛОЖЕНИЕ, KOTLIN, SPRING, SPEING-BOOT, JAVA, INTELIJ IDEA, MAVEN, TYPESCRIPT, AGULAR, ANGULAR CLI, HTML, CSS, WEBSTORM, ИНФОРМАЦИОННАЯ СИСТЕМА, ПРЕЦЕДЕНТ, КЛАСС, БАЗА ДАННЫХ, СУБД, POSTRGESQL.

Объектом исследования является приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Цель работы – разработка приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

В выпускной квалификационной работе кратко описаны правила пользования приложением.

Приведена структура программы в виде списка модулей и характеристики каждого. Расчеты метрик позволили определить оценки по стоимости и затратам на разработку приложения с веб-интерфейсом для создания имониторинга задач рабочей группы. На основе диаграмм прецедентов, последовательности, взаимодействий и классов установлены отношения между [актёрами](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%82%D1%91%D1%80_(UML)) и [прецедентами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82_(UML)), спроектированы и созданы классы, интерфейсы и отношения между ними.

Рассмотрены современные среды разработки, их возможности и преимущества. Выбраны оптимальные для разрабатываемого программного средства инструменты.

В результате разработки было разработано приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы. Рассмотрены перспективы развития проекта, а также возможности внедрения в другие сферы деятельности.

**Содержание**

[Обозначения и сокращения 7](#_Toc102332448)

[Введение 8](#_Toc102332449)

[1 Анализ предметной области разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 11](#_Toc102332450)

[1.1 Актуальность разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 11](#_Toc102332451)

[1.2 Анализ существующих приложений для создания и мониторинга задач рабочей группы 12](#_Toc102332452)

[1.3 Техническое задание на разработку приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 17](#_Toc102332453)

[1.4 Выбор технических средств для разработки приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 18](#_Toc102332454)

[1.4.1 Выбор технических средств для серверной части приложения 18](#_Toc102332455)

[1.4.2 Выбор технических средств для разработки пользовательского интерфейса 21](#_Toc102332456)

[1.4.3 Выбор СУБД для приложения 21](#_Toc102332457)

[1.4.4 Описание среды разработки программного средства приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 22](#_Toc102332458)

[1.5 Вывод по первому разделу 26](#_Toc102332459)

[2 Проектирование приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 28](#_Toc102332460)

[2.1 Анализ требований разработки на основе построения диаграмм прецендентов приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 28](#_Toc102332461)

[2.2 Главный сценарий взаимодействия в разрабатываемом приложении 32](#_Toc102332462)

[2.3 Построение диаграммы классов приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 35](#_Toc102332463)

[2.4 Построение диаграммы последовательности приложения с веб-интерфейсом для мониторинга и постановки задач рабочей группы 39](#_Toc102332464)

[2.5 Анализ рисков при разработке приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 41](#_Toc102332465)

[2.5.1 Расчет FP-метрик разрабатываемого приложения 41](#_Toc102332466)

[2.5.2 Расчет LOC-метрик разрабатываемого приложения 45](#_Toc102332467)

[2.5.3 Расчет COCOMO-метрик разрабатываемого приложения 46](#_Toc102332468)

[2.6 Проектирование базы данных приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 48](#_Toc102332469)

[2.7 Вывод по второму разделу 50](#_Toc102332470)

[3 Программная реализация приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 52](#_Toc102332471)

[3.1 Веб-интерфейс приложения для создания и мониторинга задач рабочей группы 52](#_Toc102332472)

[3.2 Руководство пользователя приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы 59](#_Toc102332473)

[3.3 Выводы по третьему разделу 69](#_Toc102332474)

[Заключение 70](#_Toc102332475)

[Список используемой литературы 72](#_Toc102332476)

[Приложение А 75](#_Toc102332477)

[Приложение В 120](#_Toc102332478)

# Обозначения и сокращения

СУБД – система управления базами данных

ПО – программное обеспечение

БД – база данных

# Введение

Каждый работник любого предприятия выполняет определенную задачу, поставленную своим руководителем. Он сам решает, как разделить работу между своими подчиненными. Из-за высокой занятости сложно следить за тем, что и как должен делать работник, поэтому со временем люди начали решать эту проблему. Все начиналось с физического сохранения составленного плана выполнения работы сотруднику работодателем, однако, по естественным причинам, придерживаться такого метода было нерационально: в задачи вносились правки, работники могли, незаметно для руководителя, избавляться от своих обязанностей или добавлять их другим. Благодаря этим и многим другим факторам, приложение, которое позволяет следить за поставленными задачами, актуально в многих компаниях.

Целью выпускной квалификационной работы является разработка приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Произвести анализ предметной области разрабатываемого приложения с веб-интерфейсов для создания и мониторинга задач рабочей группы.
2. Спроектировать приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы.
3. Реализовать в программном виде приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Каждая из вышеперечисленных задач содержит в себе подзадачи. Анализ предметной области включает в себя следующие подзадачи:

1. Выделение актуальности разрабатываемого приложения.
2. Произвести анализ существующих приложений для создания и мониторинга рабочей группы.
3. Сформировать техническое задание на разработку приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
4. Выбрать среду разработки для создания приложения (выбрать среду разработки серверной части приложения, веб-интерфейса и СУБД).

Проектирование приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы включает в себя следующие подзадачи:

1. Произвести анализ требований разработки на основе построения диаграмм прецедентов приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
2. Определить главный сценарий взаимодействия.
3. Построить диаграмму классов приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
4. Построить диаграмму последовательности для приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
5. Проанализировать риски при разработке приложения с веб-интерфейсом для создания и мониторинга рабочей группы (рассчитать FP-метрики, LOC-метрики, COCOMO-метрики).
6. Спроектировать базу данных приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Программная реализация приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы включает в себя следующие подзадачи:

1. Описать среду разработки программного средства приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
2. Создать веб-интерфейс приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
3. Сформировать руководство пользователя приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Путь к достижению цели выпускной квалификационной работы (ВКР) вызвал необходимость использования трудов национальных и зарубежных разработчиков, вложивших непосредственный вклад в техническую литературу. Например, «Чистый код» [24] и «Чистая архитектура» [22] Роберта Мартина, «Гид Java-разработчика: проектно-ориентированный подход» Рауль-Габриэля Урма и Ричарда Уорбертона [23] и т. д.

В первой главе проведен анализ предметной области, который состоит из рассмотрения актуальности разработки, аналитического обзора приложения и постановки задачи для создания приложения в виде технического задания - с перечислением общих сведений, назначения системы, цели её создания системы и требований к ней.

Во второй главе спроектировано приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы. Для этого: проведен анализ требований разработки на основе диаграммы прецедентов, написан главный сценарий взаимодействия, построены диаграммы классов и последовательности, спроектирована БД для приложения. Кроме того, проанализированы риски при разработке приложения ВКР, а именно рассчитаны FP-, LOC- и COCOMO-метрики.

В третьей главе демонстрируется программная реализация разработанного приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы. В этой главе писана среда разработки программного средства приложения, описан веб-интерфейс и сформировано руководство пользователя разработанного приложения.

Дипломный проект состоит из введения, трех разделов, заключения и двух приложений.

# 1 Анализ предметной области разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы

## **1.1 Актуальность разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

За все время развития информационно-вычислительных систем разработчики стремились к созданию такого продукта, который заменит либо однообразную рутинную работу, как, например, заполнение каких-либо письменных форм, либо автоматизирует эту работу до такого состояния, что только воспоминание о её первоначальном состоянии будет подчеркивать удобство созданного продукта. Из этого следует, что создание неактуального приложения практически невозможно, т. к. любое программное ПО для удобства или автоматизации работы. Информационный прогресс не оставил в стороне и задачи структуризации задач для работников.

Абсолютно любое предприятие тем или иным способом распределяют задачи между своими работниками опираясь на ряд причин. Организация задач может выражаться как в устном плане, так и в письменном, но так или иначе эти способы для большинства организаций уже устарели по ряду причин:

* резкий рост кадров;
* высокая нагрузка руководителя;
* недобросовестные действия сотрудников и т.д.

Актуальность создания, а тем более использования приложения для мониторинга и создания задач аргументируется желанием работодателя следить за ходом работы подчиненных и структуризацией поставленных задач. Кроме того, актуальность темы заключается и в практической ценности продукта, иначе не существовало бы программных средств для выполнения схожих задач.

Очевидным примером является использование подобного рода приложений в команде разработчиков программных продуктов. Каждый разработчик может просматривать свои задачи на специальной панели и редактировать статус проекта в зависимости от проделанной работы.

Таким образом, мы рассмотрели актуальность разработки приложения для создания и мониторинга задач, которое помогает рабочей группе в администрировании и управлении своими проектами в ходе рабочего процесса.

## **1.2 Анализ существующих приложений для создания и мониторинга задач рабочей группы**

На текущий момент существует ряд программных средств для мониторинга и постановки задач. К их числу относятся: Jira, Redmine, Trello, Basecamp и Wrike [1]. Рассмотрим каждое ПО более подробно.

Jira – это коммерческий программный инструмент, созданный компанией Atlassian, с целью управления проектами. Известность Jira приобрела в IT-компаниях, где используется преимущественно для формирования перечня задач, наблюдения общего прогресса команды разработчиков и быстрого решения проблем, возникающих в ходе разработки продукта. Эта система использует Agile-методику разработки, которая подразумевает деление больших задач на маленькие подзадачи [2]. Скриншот программы Jira представлен на рисунке ниже (Рисунок 1.1).

Преимущества:

* богатый функционал;
* имеется мобильная версия;
* приложение построено по принципам канбан- и скрам- досок (методики планирования задач).

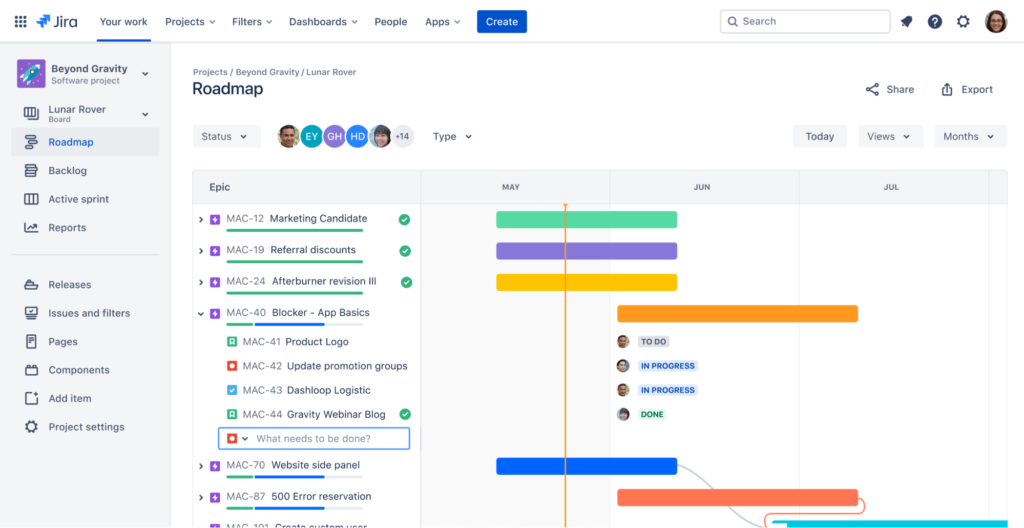


Рисунок 1.1 – Скриншот ПО Jira

Недостатки:

* сложный пользовательский интерфейс;
* высокая цена.

Redmine – это серверное веб-приложение для управления проектами [3]. Оно написано на языке программирования Ruby и основано на веб-фреймворке Ruby on Rails. Скриншот программы Redmine представлен на рисунке ниже (Рисунок 1.2).

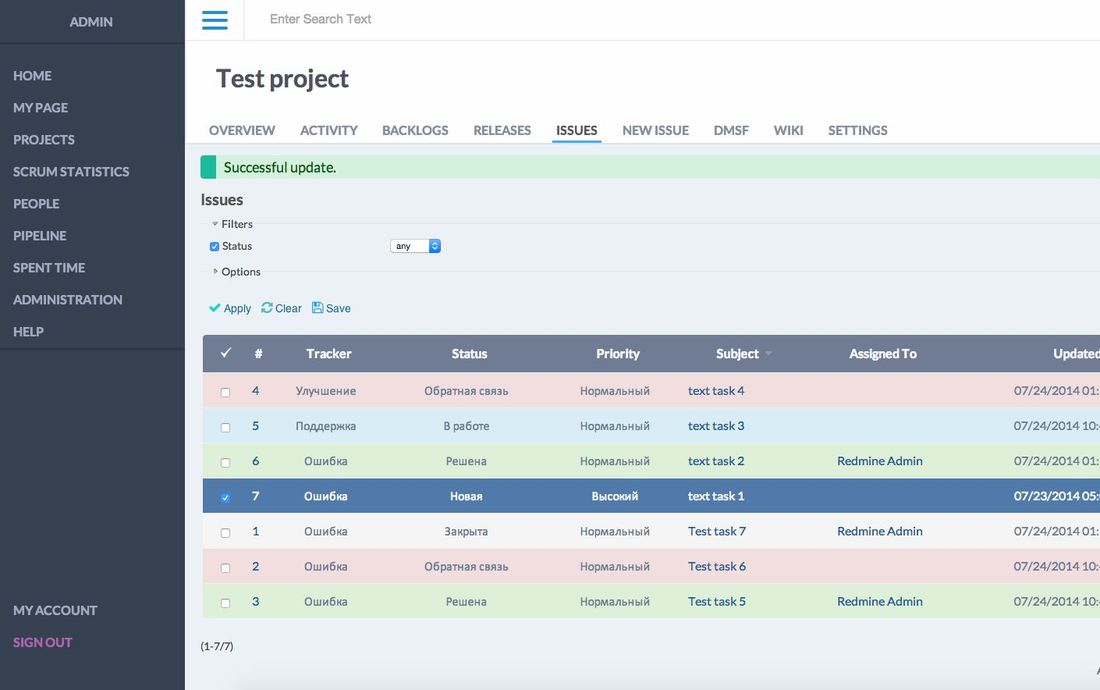


Рисунок 1.2 – Скриншот ПО Redmine

Преимущества:

* открытый исходный код;
* бесплатное использование;
* простой функционал;
* интеграция с системами контроля версий и поддержка СУБД.

Недостатки:

* отсутствие мобильной версии;
* упрощенный функционал.

Trello – это облачная программа для управления проектами малых групп, разработанная Fog Creek Software [18]. Внешний вид напоминает поле с заметками, на котором можно легко и удобно перемещать задачи, а также организовывать их в столбцы. Приложение удобно в повседневной жизни. Скриншот программы Trello представлен ниже   
(Рисунок 1.3).

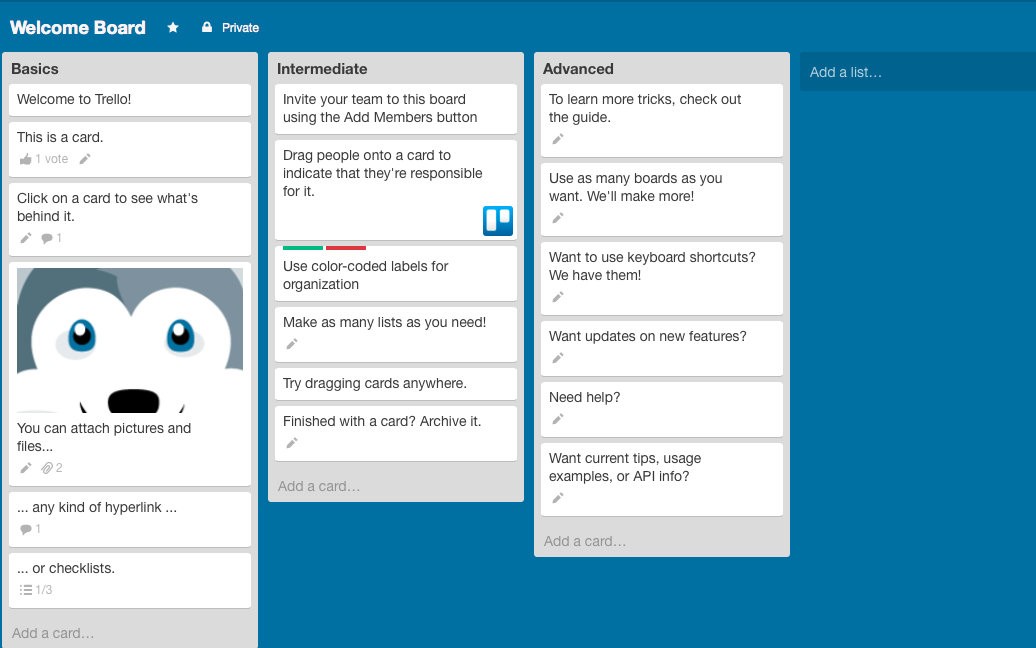


Рисунок 1.3 – Скриншот ПО Trello

Преимущества:

* простой интерфейс;
* условно бесплатный сервис;
* имеется мобильная версия.

Недостатки:

* неудобно работать со сложными проектами;
* ограниченный функционал, который нужно самому дополнительно расширять.

Basecamp – это онлайн-сервис для управления проектами, позволяющий командам организовать свою работу [19]. Этот сервис объединяет внутри себя инструменты для коммуникации, мониторинга задач, передачи файлов и т.д. В процессе создания и использования Basecamp был создан фреймворк Ruby on Rails. Скриншот программы Basecamp представлен на рисунке ниже (Рисунок 1.4).

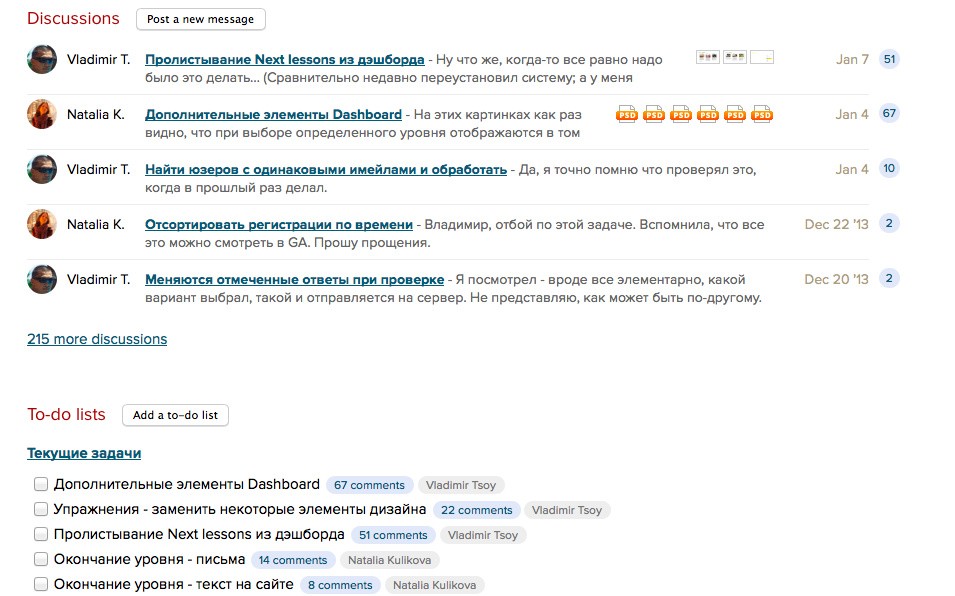


Рисунок 1.4 – Скриншот ПО Basecamp

Преимущества:

* огромный функционал;
* синхронизация с Google сервисами;
* возможность хранить проекты на своем сайте, используя SSL-шифрование.

Недостатки:

* неудобный интерфейс;
* плохая оптимизация приложения.

Wrike – это корпоративный онлайн-сервис для совместной работы и управления проектами [20]. Сервис содержит впечатляющий набор функций и инструментов для отслеживания задач и подзадач. Скриншот программы Wrike представлен на ниже (Рисунок 1.5).

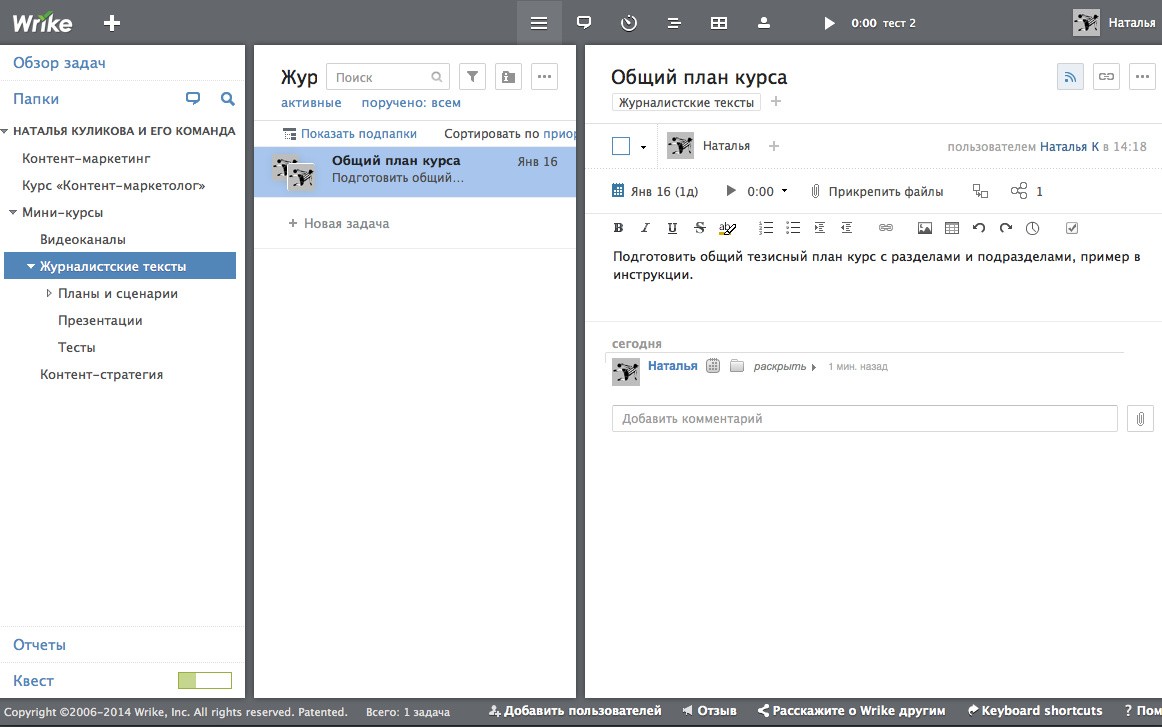


Рисунок 1.5 – Скриншот ПО Wrike

Преимущества:

* доступен бесплатный тарифный план;
* широкий функционал;
* доступна мобильная версия.

Недостатки:

* ограниченная интеграция с другими технологиями;
* требует обучения для удобной и правильной работы.

Таким образом, рассмотрев пять приложений, позволяющих создавать и просматривать задачи команды, были рассмотрены преимущества и недостатки каждого из приведенных приложений, что позволит учесть выявленные преимущества и недостатки данных приложений при создании приложения для создания и мониторинга задач рабочей группы.

## **1.3 Техническое задание на разработку приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

Полное наименование системы: приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы. Краткое название: менеджер задач.

Разрабатываемое приложение должно демонстрировать пользователю его задачи, реализовывать администрирование группы для организации проекта.

Менеджер задач предназначен для:

1. Создания задачи.
2. Назначения задачи исполнителю.
3. Отслеживания статуса задачи.

Менеджер задач создается с целью:

1. Распределения задач между командой.
2. Отслеживания хода выполнения задач.
3. Управления задачами проекта руководителем.

В приложении планируется один режим функционирования – это авторизованный режим, в которой пользователь будет заходить, используя специальную форму для ввода логина и пароля.

Все данные, с которыми пользователь будет работать, должны храниться в БД.

Приложение будет состоять из двух полноценных функционирующих компонентов:

1. Фронтенд – клиентская сторона пользовательского интерфейса (веб-интерфейс приложения).
2. Бэкенд – программно-аппаратная часть сервиса с доступом к БД.

Пользователь будет обращаться к данным, находящимся в БД, посредством взаимодействия с фронтендом, который в свою очередь обращается к бэкенду, используя синхронный способ взаимодействия – REST API и протокол HTTP. Бэкенд же, получив запрос с клиентской стороны, будет обращаться к БД с помощью языка SQL и предусмотренного API от Java EE.

Система должна поддерживать следующие режимы функционирования:

* основной режим, в котором пользователь сможет просматривать, создавать, назначать задачи;
* режим ожидания авторизации, в котором пользователь должен только авторизироваться.

Веб-интерфейс (фронтенд) должен реализовать весь представленный бэкендом API.

Таким образом, разработано техническое задание на разработку мобильного веб-приложения для учета и контроля личных денежных средств и приложение должно соответствовать приведенным выше требованиям.

**1.4 Выбор технических средств для разработки приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

После определения требований к системе, нужно выбрать технические средства для разработки приложения: языки программирования, фреймворки, СУБД и среда разработки. Из вышесказанного следует, что необходимо определить:

1. Технические средства для серверной части приложения.
2. Технические средства для пользовательского интерфейса.
3. СУБД.
4. Среду разработки приложения.

Рассмотрим их более подробно.

### 1.4.1 Выбор технических средств для серверной части приложения

Рассмотрим существующие языки программирования, подходящие по требованиям для проекта.

Kotlin – это статически типизированный, лаконичный, безопасный и прагматичный язык, совместимый с Java [6]. Этот язык можно использовать везде, где применяется или применялась Java:

* разработка серверный приложений;
* разработка мобильных приложений;
* разработка веб-интерфейсов;
* разработка приложений под Android (рекомендуется Google [4]).

Основные преимущества языка программирования Kotlin [6]:

1. Прагматичность. Kotlin является практичным языком программирования, подходящим для решения реальных задач. При создании языка разработчики выбирали его характеристики, исходя из задач, которые чаще всего приходилось решать разработчикам. Кроме того, язык не требует применения какого-либо конкретного стиля программирования или парадигмы.
2. Лаконичность. Эта характеристика означает, что синтаксис Kotlin явно выражает намерения кода, не загромождая его вспомогательными конструкциями и деталями реализации.
3. Безопасность. Несмотря на то, что ни один язык не может быть полностью безопасным, при создании языка Kotlin авторы смогли достичь более высокого уровня безопасности, чем в Java, с минимальными накладными расходами. Кроме гарантий безопасности кода на JVM, которые унаследовал себе Kotlin, этот язык обеспечивает безопасность типов в приложении, благодаря автоматическому определению типа. Плюс ко всему этому Kotlin помогает избавить программу от NullPointerException с помощью внутренних конструкций по проверке типа на значение null.
4. Совместимость. Kotlin полностью совестим с Java, являясь практически оберткой для нее. С помощью Kotlin можно вызывать Java-методы, наследовать Java-классы, использовать Java-библиотеки и т. д.

Его используют следующие компании: Square, Pinterest, Basecamp и Corda [8].

Конечно, нельзя не сравнить Kotlin с Java. Несмотря на нарастающую популярность Koltin, у Java существуют еще некоторые преимущества, например, Java обладает более высокой скоростью компиляции и имеет большое сообщество с готовыми проектами. Но редкий Kotlin-разработчик не знает язык Java, что доказывает более превосходящую позицию Kotlin [10].

Python – это интерпретируемый язык программирования с динамической типизацией [1], который отличается от Koltin производительностью и скоростью, обширными библиотеками и доступностью сторонних модулей, хотя Kotlin легче во внедрении, обладает многоплатформенной совместимостью и менее подвержен ошибкам [7]. Таким образом, Python и Kotlin могут применяться в одной сфере (серверные приложения) и выполняют свои обязанности идентично, однако существуют случаи, когда проекты требуют статической типизации как у Kotlin, хотя Python и считается лучим динамическим языком [7].

Golang – это компилируемый многопоточный язык программирования, разработанный внутри компании Google, который отличается от Kotlin минимальным ортогональным набором функций, который позволяет молниеносно компилировать, предлагая при этом скорость выполнения нативных приложений [9]. Так, язык Golang является более популярным чем, Kotlin, однако популярность не влияет на качество языка.

У каждого из этих языков есть плюсы и минусы, но в целом они похожи. Но языком программирования для серверной части приложения для создания и мониторинга задач рабочей группы был выбран Kotlin.

Что касается фреймворка, то лучшим выбором будет Spring (и его расширение Spring Boot). Его предпочитает большинство разработчиков на этом языке, в том числе благодаря возможности внедрения зависимостей. Spring идеально подходит для создания корпоративных приложений и моделей конфигурации на базе Java. Он позволяет разработчикам сосредоточиться на бизнес-логике приложения [11]. Этот фреймворк позволяет разрабатывать [5]:

* микросервисные приложения;
* неблокируемую архитектуру;
* веб-приложения;
* автоматизированные задачи и не только.

Таким образом, выбор языка Kotlin обусловлен его безопасностью, лаконичностью, совместимостью с Java и т. д. Также аргументирован выбор фреймворка Spring, который также используется в Netflix.

### 1.4.2 Выбор технических средств для разработки пользовательского интерфейса

Для создания пользовательского интерфейса выбран фреймворк Angular на языке TypeScript.

Angular – это фреймворк с открытым исходным кодом, написанный JavaScript и поддерживаемый компанией Google [14]. С помощью него можно разрабатывать приложения не только на JavaScript, но еще и на Dart или TypeScript. Последний будет использоваться при создании приложения ВКР. Преимущества этого Angular [12]:

* скорость и производительность;
* широкий спектр шаблонов;
* хорошая индексация Google.

С этим фреймворком работают: Adobe, Amazon, Disney, NASA, Microsoft и т.д [13].

Таким образом, несмотря на не такую большую популярность, как, например, у библиотеки для JavaScript - React, этот фреймворк является одним из лучших в своей сфере, который используют многие компании.

### 1.4.3 Выбор СУБД для приложения

В качестве СУБД в приложении используется PostgreSQL, который является одним из нескольких популярных бесплатных вариантов СУБД [15]. Из достоинств стоит отметить следующее:

* является масштабируемым решением и позволяет обрабатывать терабайты данных;
* бесплатное обслуживание;
* поддержка формата JSON;
* множество предопределенных функций;
* доступен ряд интерфейсов.

Недостатки:

* плохо проработанная документация;
* неоднозначная конфигурация;
* скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения.

Существует ряд других достойных СУБД, но PostgreSQL является лучшим бесплатным вариантом. Oracle – это очень надежная СУБД с соответствующей высокой ценой; MySQL – СУБД с платной и бесплатной версиями, которая обладает хорошей гибкостью и внутренними функциями, но некоторые системные задачи требуется выполнять самому; MongoDB – СУБД для NoSQL-систем с высокой скоростью и лёгким обращением.

Таким образом, для создания и функционирования небольших приложений PostgeSQL будет лучшим вариантом.

### 1.4.4 Описание среды разработки программного средства приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы

В ходе процесса разработки любого приложения используется специальная среда разработки. Её выбор зависит от языка программирования, с помощью которого будет создано приложение. Очевидно, что с развитием ПО и языков появится компания, целью которой является создания необходимых инструментов для разработчиков и команд. Самая популярная компания, специализирующаяся на подобной области – это JetBrains.

Эта компания была основана в 2000 году тремя российскими разработчиками - Сергеем Дмитриевым, Валентином Кипятковым и Евгением Беляевым [26]. Сегодня JetBrains является ведущей компанией по разработке ПО для разработчиков, например [25], Intellij IDEA, WebStorm, RyCharm, PhpStorm, CLion, GoLand и т. д.

Для создания серверной, написанной на языке программирования Kotlin, части приложения используется среда разработки Intellij IDEA. Эта среда была первым продуктом компании JetBrains. Кроме того, именно они не только создали язык Kotlin, но и поддерживают, и разрабатывают этот язык до сих пор. Intellij IDEA прекрасно подходит для Java - разработчика, позволяя разрабатывать проекты как на языке Java, так и на Kotlin и Groovy. Кроме того, позволяет создавать проекты на фреймворках Spring, Quarkus, Micronaut и т. д., использовать разные системы сборки (Maven и Gradle) и многое другое (Рисунок 1.6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.6 – Спектр возможностей создания нового проекта в Intellij IDEA

На рисунке ниже (Рисунок 1.7) представлен текстовый редактор Intellij IDEA, в котором видно, насколько лаконична и удобна эта среда разработки.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.7 – Скриншот текстового редактора в Intellij IDEA

Основные преимущества среды разработки Intellij IDEA [27]:

* Возможность продуктивной разработки – умный реактор кода позволяет сделать разработку приятной.
* Глубокое понимание кода – индексирование исходного кода предоставляет массу возможностей для быстрой и эффективной разработки: автодополнение, анализ кода и надежный рефакторинг.
* Нет необходимости устанавливать плагины для интеграции с системой контроля версий (например, Git) и фреймворков.
* Несмотря на то, что основной язык этой среды разработки – Java, она понимает и предоставляет помощь в других языках, например, JavaScript, HTML, CSS и т. д.

Для создания веб-интерфейса используется среда разработки WebStorm, тоже от компании JetBrains. Эта среда разработки позволяет создавать веб-интерфейс с помощью таких фреймворков и библиотек (Рисунок 1.8), как AngularJS, Angular CLI, React, React Native, Bootstrap, Vue.js и т. д.

Основными преимуществами WebStorm являются:

* умный редактор кода;
* встроенные инструменты для разработчиков;
* быстрая навигация и поиск;
* кастомизация интерфейса среды разработки и т. д.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.8 - Спектр возможностей создания нового проекта в WebStorm

На рисунке ниже (Рисунок 1.9) представлен текстовый редактор WebStorm.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.9 - Скриншот текстового редактора в WebStorm

Таким образом, были рассмотрены две среды разработки для двух частей приложения с веб-интерфейсов для создания и мониторинга задач рабочей группы. Для серверной части выбрана среда разработки Intellij IDEA, а для создания веб-интерфейса – WebStorm.

## **1.5 Вывод по первому разделу**

В данном разделе был осуществлен анализ предметной области для разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

На основе анализа предметной области выделена актуальность разработки приложения для создания и мониторинга задач рабочей группы.

Аналитический обзор позволил сравнить существующее ПО (Jira, Redmine, Trello, Basecamp и Wrike) со схожим функционалом как у разрабатываемого приложения. Преимущества и недостатки рассматриваемых систем указали на возможные преимущества и недостатки разрабатываемой системы.

Поставлена задача ВКР – разработать ПО с веб-интерфейсом, в котором будет доступен просмотр и управление задачами

ПО содержит следующие цели:

* распределение задач между командой;
* отслеживание хода выполнения задач;
* управление задачами проекта руководителем.

В качестве требований к системе были выделены следующие категории:

* приложение состоит их двух режимов: когда пользователь авторизирован и нет, причем во первой случае приложение имеет основной функционал;
* приложение состоит их двух основных компонентов – фронтенд, пользовательский интерфейс, и бэкенд, программно-аппаратная часть;
* компоненты синхронно взаимодействуют между собой через REST API, используя HTTP протокол;
* данные хранятся в БД.

На основе предметной области мобильного приложения разработали техническое задание для правильной реализации приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Была произведен выбор технических средств для разработки программного средства для создания приложения:

1. Для бэкенда используется фреймворк Spring (и его расширение Spring Boot) для языка программирования Kotlin.
2. Для фронтенда используется фреймворк Angular на языке TypeScript.
3. В качестве СУБД используется PostgreSQL.
4. Для каждой части приложения была выбрана среда разработки: для бэкенда на Kotlin – Intellij IDEA, для фронтенда на Angular CLI – WebStorm.

Таким образом, собранные во время анализа предметной области сведения являются основой для проектирования приложения.

# 2 Проектирование приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы

## **2.1 Анализ требований разработки на основе построения диаграмм прецендентов приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

Одной из диаграмм, которые применяются на этапе проектирования модели информационной системы, является диаграмма прецедентов (диаграмма вариантов использования), предназначенная для построения модели функционирующей системы в окружающей среде на концептуальном уровне [17]. Кроме того, диаграмма прецедентов позволяет проанализировать требования к ПО.

Основными элементами диаграммы прецедентов являются:

* актёр – элемент, обозначающий роль пользователя, который взаимодействует с определенной сущностью (обозначается фигурой человека);
* прецедент – это элемент, который отражает действия, выполняемые системой (обозначается овалом или кругом).

Существует рад связей, который может быть установлен между прецедентами:

* обобщение – указывает общность ролей (рисунок 2.1 – а);
* ассоциация – эта связь предназначена для соединения актеров и прецедентов (рисунок 2.1 – б);
* включение (include) – указывает взаимосвязь нескольких вариантов использования, базовый из которых всегда использует функциональное поведение связанных с ним прецедентов (рисунок 2.1 – в);
* расширение (extend) – указывает взаимосвязь базового варианта использования и таких вариантов использования, которые являются его специальными случаями (рисунок 2.1 – г).

Представлено обозначение связей (Рисунок 2.1).

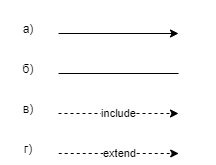


Рисунок 2.1 – Обозначение связей

В приложении будет фигурировать три роли пользователя – неавторизированный пользователь, авторизированный пользователь и администратор группы. Диаграмма прецедентов для первой роли представлена на рисунке ниже (Рисунок 2.2).

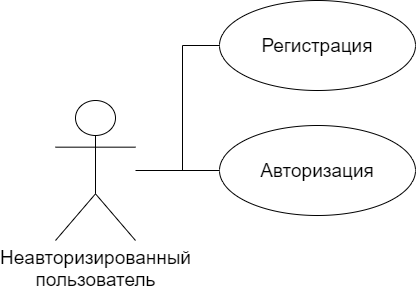


Рисунок 2.2 – Диаграмма прецедентов, роль – неавторизированный пользователь

Как видно из рисунка выше, неавторизированный пользователь способен изменить свою роль с помощью элемента «Авторизация» или зарегистрироваться. На ниже (Рисунок 2.3) представлена диаграмма прецедентов для второй роли – авторизированный пользователь.

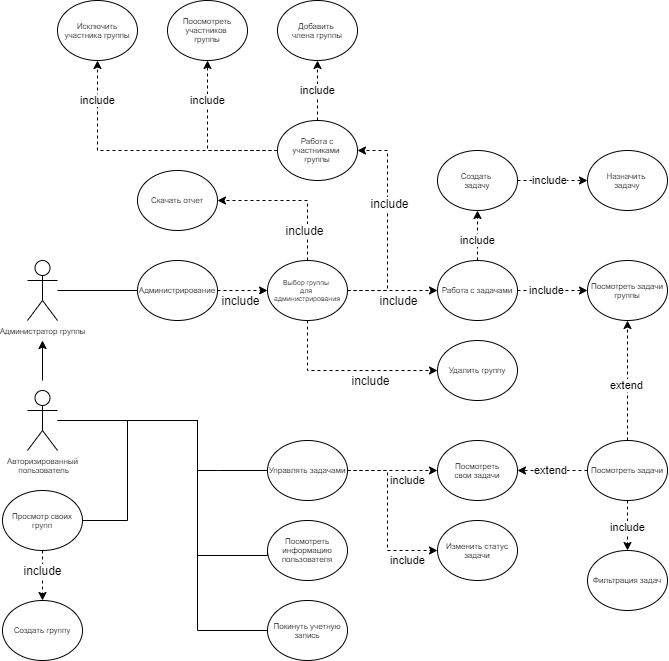


Рисунок 2.3 – Диаграмма прецедентов, роль – авторизированный пользователь

Разберем что изображено на диаграмме прецедентов (Рисунок 2.3). У авторизированного пользователя имеется четыре отношения ассоциации:

* управлять задачами;
* посмотреть информацию пользователя;
* посмотреть свои группы;
* покинуть учетную запись.

Это четыре основных прецедента, которые определяют направление действий пользователя. Когда клиент покинет учетную запись, он поменяет роль авторизированного пользователя на неавторизированного.

На рисунке выше (Рисунок 2.3) используется связь обобщения между авторизированным пользователем и администратором группы. Это означает, что администратор имеет более расширенный функционал, чем обычный авторизированный пользователь. Иными словами, авторизированный пользователь может все, кроме администрирования групп. Администратором группы может стать любой пользователь, только если он сам создаст группу.

Перед переходом к администрированию групп, необходимо выбрать группу. Далее администратор имеет возможность:

* удалить группу;
* скачать отчет;
* начать работу с задачами;
* начать работу с участниками группы.

Работа с задачами включает в себя:

* просмотр задач группы (с возможностью их фильтрации);
* создание новой задачи;
* назначение задачи участнику группы.

Работа с участниками группы состоит из следующих элементов:

* просмотр участников группы;
* добавление пользователя в группу;
* исключение пользователя из группы.

На диаграмме присутствует базовый элемент «Посмотреть задачи», который связан типом «extend» с двумя дочерними элементами:

* посмотреть задачи группы;
* посмотреть свои задачи.

Эти данные пользователь будет получать из одной таблицы в БД, поэтому решение введения базового элемента «Посмотреть задачи» полностью оправдано и является рациональным.

Таким образом, была рассмотрена диаграмма прецедентов, содержащей в себе три роли, две из которых связаны типом связи «обобщение». Была построена модель функционирующей системы с учетом особенностей темы. Анализ требований показал, что основная работа пользователей будет осуществляться с задачами.

## **2.2 Главный сценарий взаимодействия в разрабатываемом приложении**

Рассмотрим главный сценарий взаимодействия пользователя и информационной системы - «Просмотр назначенных пользователю задач».

Главный раздел

Вариант использования: просмотр задач

Актеры: пользователь, фронтенд (пользовательский веб-интерфейс приложения), бэкенд (серверная часть приложения).

Цель: увидеть список задач, назначенных конкретному пользователю.

Краткое описание: Пользователь проходит авторизацию приложения, после чего переходит в специальный раздел, где просматривает свои задачи.

Тип: базовый

Включает в себя ВИ авторизации профиля пользователя.

Раздел – «Типичный ход событий»:

Действия актеров:

1. Пользователь попадает в окно авторизации.

2. Пользователь вводит логин и пароль в специальные поля формы авторизации и отправляет их.

Исключение №1: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

3. Фронтенд отправляет данные в бэкенд.

4. Бэкенд сверяет полученные данные с данными пользователями из БД.

Исключение №2: Ошибка подключения, разорвано соединение с БД.

Исключение №3: Ошибка авторизации, неверный логин или пароль.

5. Бэкенд уведомляет фронтенд об успешной авторизации.

6. Фронтенд сохраняет логин и пароль в локальном хранилище.

7. Фронтенд перенаправляет пользователя на главную страницу приложения.

8. Пользователь выбирает раздел «Мои задачи».

9. Фронтенд создает и отправляет специальный запрос, используя логин и пароль пользователя, для бэкенда.

Исключение №4: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

Исключение №5: Ошибка отправки запроса, данные пользователя устарели и были удалены.

10. Бэкенд принимает запрос от фронтенда.

11. Бэкенд создает запрос в БД.

12. Бэкенд отправляет запрос в БД.

Исключение №6: Ошибка подключения, разорвано соединение с БД.

13. Бэкенд отправляет фронтенду полученные данные.

Исключение №7: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

14. Фронтенд преобразует полученные данные для пользователя.

15. Пользователь просматривает свои задачи.

Раздел – «Исключения»:

Действия актеров:

Исключение №1: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

3. Фронтенд уведомляет пользователя об ошибке.

2. Пользователь отправляет запрос еще раз.

Исключение №2: Ошибка подключения, разорвано соединение с БД.

5. Бэкенд уведомляет фронтенд об ошибке.

6. Фронтенд уведомляет пользователя об ошибке.

2. Пользователь отправляет запрос еще раз.

Исключение №3: Ошибка авторизации, неверный логин или пароль.

5. Бэкенд уведомляет фронтенд об ошибке.

6. Фронтенд уведомляет пользователя об ошибке.

2. Пользователь вводит данные еще раз.

Исключение №4: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

10. Фронтенд уведомляет пользователя об ошибке.

8. Пользователь отправляет запрос еще раз.

Исключение №5: Ошибка отправки запроса, данные пользователя устарели и были удалены.

10. Фронтенд перенаправляет пользователя на страницу авторизации.

2. Пользователь вводит логин и пароль в специальные поля формы авторизации и отправляет их.

Исключение №6: Ошибка подключения, разорвано соединение с БД.

13. Бэкенд уведомляет фронтенд об ошибке.

14. Фронтенд уведомляет пользователя об ошибке.

8. Пользователь повторяет свой запрос

Исключение №7: Ошибка подключения, разорвано соединение между фронтендом и бэкендом.

14. Фронтенд уведомляет пользователя об ошибке.

8. Пользователь отправляет запрос еще раз.

Таким образом, рассмотрен главный сценарий взаимодействия между пользователем и приложением в разрабатываемом приложении с веб-интерфейсом для создания и мониторинга задач рабочей группы.

## **2.****3 Построение диаграммы классов приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

Диаграмма классов – это логическое представление модели, изображающей детальную информацию о структуре модели системы с использованием терминологии классов объектно-ориентированного программирования. На такой диаграмме могут быть указаны внутренняя структура и типы отношений между отдельными объектами и подсистемами, что приводит к развитию концептуальной модели системы [17].

Класс представляет концепт, описывающий состояние (атрибуты) и поведение (методы). Каждый атрибут имеет свой тип, а каждый метод – свою сигнатуру, однако, в диаграмме классов только имя класса является обязательной информацией к заполнению. Пример класса изображен на рисунке ниже (Рисунок 2.4).

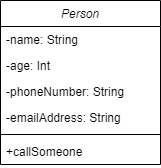


Рисунок 2.4 – Пример класса

На рисунке выше изображен класс Person, у которого есть атрибуты (name, age и т. д.) и метод (callSomeone). Все атрибуты имеют тип, название и являются закрытыми членами класса (символ минуса перед названием переменной). Методы имеют тип и название.

На рисунках ниже (Рисунки 2.5–2.7) изображены части диаграммы классов проектируемой системы.

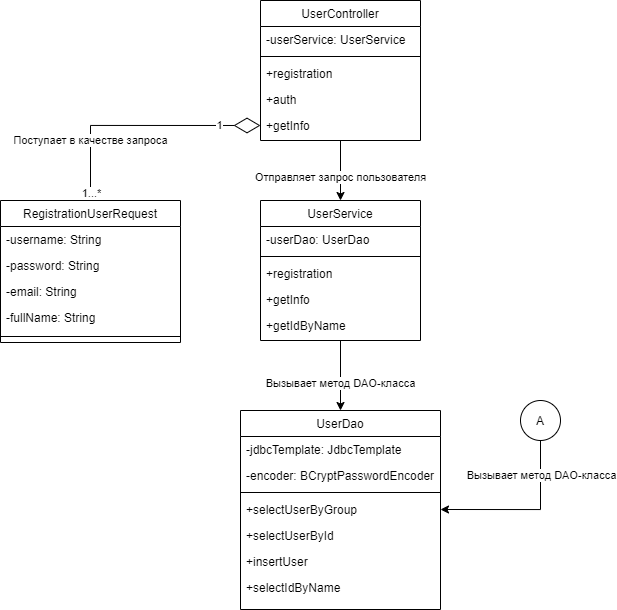


Рисунок 2.5 – Диаграмма классов разрабатываемого приложения – первая часть

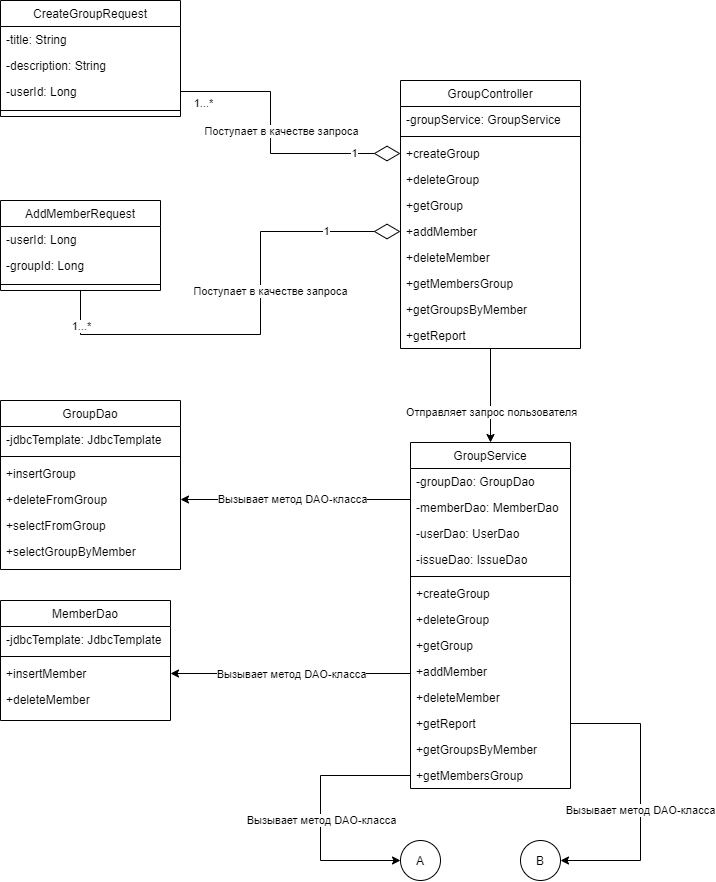


Рисунок 2.6 – Диаграмма классов разрабатываемого приложения – вторая часть

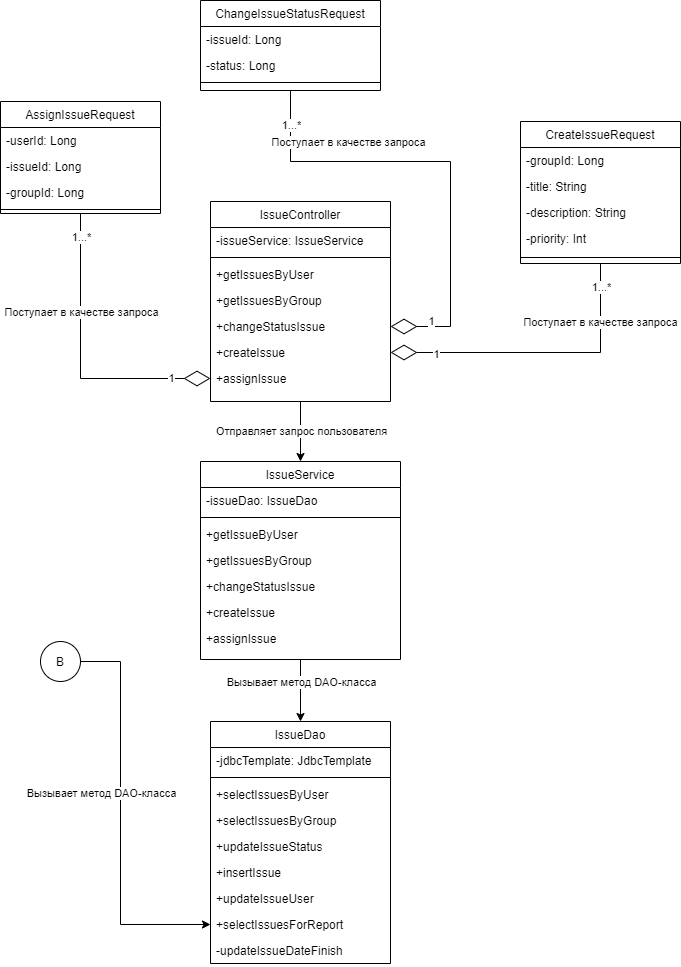


Рисунок 2.7 – Диаграмма классов разрабатываемого приложения – третья часть

Структура приложения опирается на архитектуру MVC (Model-View-Controller), которая позволяет разделить приложение на три соответствующих компонента. Таким образом, компонентом View является весь пользовательский интерфейс; компонент Controller – это совокупность следующих классов: GroupController, GroupService, GroupDao, MemberDao, IssueController, IssueService, IssueDao, UserController, UserService, UserDao; компонент Model – это сущности БД.

Механизм работы такой структуры прост:

1. На один из классов контроллеров (GroupController, IssueController, UserController) через REST API подается GET или POST запрос. Если это POST-запрос, то в качестве параметров методов котроллера используется один из шести DTO-классов (CreateGroupRequest, AddMemberRequest, AssignIssueRequest, ChangeIssueStatusRequest, CreateIssueRequest, RegistrationUserRequest).
2. Контроллер отправляет запрос методу сервиса, предназначенного для работы с конкретным контроллером (классы GroupService, IssueService, UserService).
3. Сервис обрабатывает запрос и вызывает метод DAO-класса (GroupDao, MemberDao, IssueDao, UserDao), который является прослойкой между серверной программой и БД.
4. DAO-классы создают специальный запрос в БД и конвертируют полученные данные в виде экземпляров классов моделей (Issue, Member, Group).
5. Полученные данные передаются между классами в обратном порядке, после чего контроллер создает JSON ответ для пользовательского интерфейса.

Такой подход позволяет легко расширять существующее приложение.

Таким образом, диаграмма классов отражает структуру программного содержимого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы, позволяя направить разработчика на верный ход создания информационной системы.

## **2.4 Построение диаграммы последовательности приложения с веб-интерфейсом для мониторинга и постановки задач рабочей группы**

Диаграмма последовательности – это диаграмма, отражающая упорядоченные по времени проявления взаимодействия объектов [17]. На диаграмме данного типа слева направо помещаются основные элементы: объекты; вертикальные линии, моделирующие течение времени при выполнении действий объектом; стрелки, определяющие действия, выполняемые объектом. Диаграмма последовательности представлена на рисунке ниже (Рисунок 2.8).

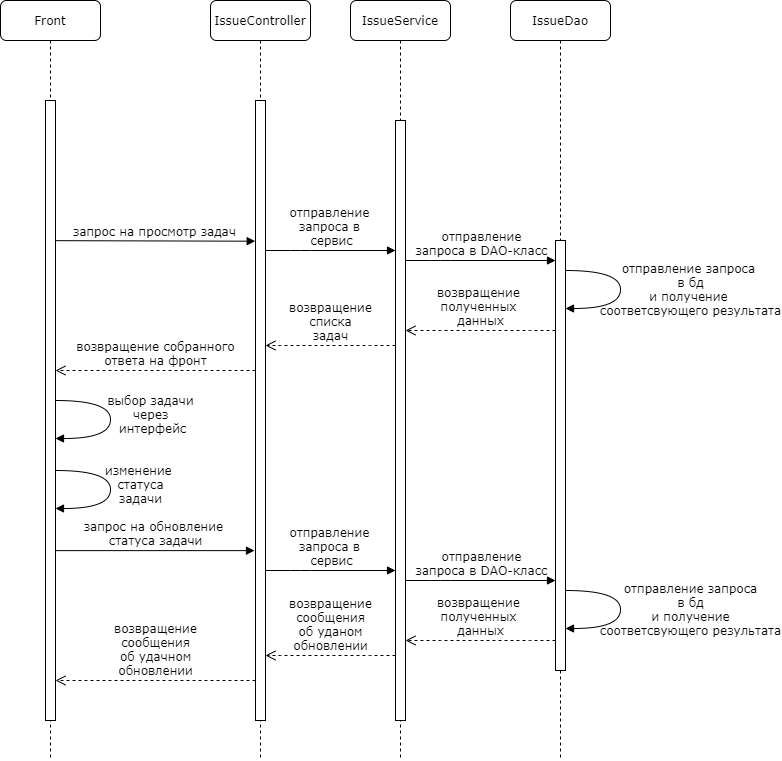


Рисунок 2.8 – Диаграмма последовательности разрабатываемого приложения

На рисунке 2.8 изображена диаграмма последовательности, на которой продемонстрирован процесс взаимодействия класса-контроллера, класса-сервиса и DAO-класса при осуществлении запроса на просмотр задач и обновлении статуса одной из представленных задач.

Из диаграммы видно, что инициатором осуществления процессов является пользователь, использующий веб-интерфейс. Кроме того, видно, что запрос обрабатывается тремя разными классами. Это сделано для того, чтобы структурировать логику создания и вызова запроса в БД. Интерфейс приложения обновляется после получения результатов данных, чтобы корректно представить их пользователю.

Таким образом, в результате построения диаграммы последовательности разработаны подробные описания процессов внутри разрабатываемого приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы, которые будут отражать функциональную ценность информационной системы.

## **2.5 Анализ рисков при разработке приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

### 2.5.1 Расчет FP-метрик разрабатываемого приложения

В оценке стоимости ПО используют две единицы размера: строка кода (LOC) и функциональная точка (FP). Эти оценки позволяют предъявить заказчику корректные требования по стоимости и затратам на разработку информационной системы.

Методика анализа FP основывается на концепции разграничения взаимодействий. Сущность ее состоит в том, что программа разделяется на классы компонентов по формату и типу логических операций. В основе этого деления лежит предположение, что область взаимодействия программы разделяется на внутреннюю – взаимодействие компонентов приложения, и внешнюю – взаимодействие с другими приложениями.

В соответствии с принятым стандартом используются 5 классов компонентов, на которых основывается анализ:

1. Внешний ввод – транзакция, при выполнении которой данные пересекают границу приложения извне.
2. Внешний вывод – транзакция, при выполнении которой данные пересекают границу приложения изнутри.
3. Внешний запрос – транзакция, при выполнении которой происходит одновременный ввод и вывод.
4. Внутренний логический файл – группа логически связанных данных, находящихся внутри границ приложения и поддерживаемых извне.
5. Внешний интерфейсный файл - группа логически связанных данных, находящихся вне границ приложения и являющихся внутренним логическим файлом для другого приложения.

Из-за того, что в приложении используется два языка программирования – Kotlin и TypeScript, то FP-метрики необходимо рассчитывать для двух языков отдельно друг от друга.

Для Kotlin:

1. Внешний ввод – 0.
2. Внешний вывод – 0.
3. Внешний запрос – 16.
4. Внутренний логический файл – 18.
5. Внешний интерфейсный файл - 4.

Для TypeScript:

1. Внешний ввод – 41
2. Внешний вывод – 26
3. Внешний запрос – 16
4. Внутренний логический файл – 18
5. Внешний интерфейсный файл - 6

Каждой из выявленных характеристик ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг, а затем формируется числовая оценка ранга. Теория для подсчета была взята из методички о размерно-ориентированных метриках [21].

Исходные данные расчетов метрик представлены в таблице 2.1 для Kotlin и в таблице 2.2 для TypeScript.

Таблица 2.1 – Расчеты FP метрик для Kotlin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя характеристики | Ранг, сложность, количество | | | |
| Низкий | Средний | Высокий | Итого |
| Внешние вводы | 0\*3=0 | 0\*4=0 | 0\*6=0 | 0 |
| Внешние выводы | 0\*4=0 | 0\*5=0 | 0\*7=0 | 0 |
| Внешние запросы | 16\*3=48 | 0\*4=0 | 0\*6=0 | 48 |
| Внутренние логические файлы | 18\*7=126 | 0\*10=0 | 0\*15=0 | 126 |
| Внешние интерфейсные файлы | 4\*5=20 | 0\*7=0 | 0\*10=0 | 20 |
| Итого | | | | 194 |

Таблица 2.2 – Расчеты FP метрик для TypeScript

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя характеристики | Ранг, сложность, количество | | | |
| Низкий | Средний | Высокий | Итого |
| Внешние вводы | 8\*3=24 | 2\*4=8 | 0\*6=18 | 50 |
| Внешние выводы | 8\*4=0 | 0\*5=0 | 0\*7=0 | 32 |
| Внешние запросы | 3\*3=0 | 0\*4=0 | 0\*6=0 | 9 |
| Внутренние логические файлы | 12\*7=84 | 0\*10=0 | 0\*15=0 | 84 |
| Внешние интерфейсные файлы | 6\*5=30 | 0\*7=0 | 0\*10=0 | 30 |
| Итого | | | | 205 |

Количество функциональных указателей вычисляется по формуле:

, (2.1)

где Fi = коэффициенты регулировки сложности, каждый коэффициент может принимать следующие значения:

0 – нет влияния

1 – случайное

2 – небольшое

3 – среднее

4 – важное

5 – основное

Значения коэффициентов регулировки приведены в таблице 2.3.

Таблица 2.3 – Значения коэффициентов регулировки сложности

|  |  |  |
| --- | --- | --- |
| № | Системный параметр | Значение |
| 1 | 2 | 3 |
| 1 | Передачи данных | 2 |
| 2 | Распределенная обработка данных | 4 |
| 3 | Производительность | 2 |
| 4 | Распространенность используемой конфигурации | 5 |
| 5 | Скорость транзакций | 3 |
| 6 | Оперативный ввод данных | 4 |
| 7 | Эффективность работы конечного пользователя | 3 |
| 8 | Оперативное обновление | 2 |
| 9 | Сложность обработки | 1 |
| 10 | Повторная используемость | 4 |
| 11 | Легкость инсталляции | 2 |
| 12 | Легкость эксплуатации | 3 |
| 1 | 2 | 3 |
| 13 | Разнообразные условия размещения | 4 |
| 14 | Простота изменений | 3 |

Сума коэффициентов равна 42. Теперь мы имеем все необходимые данные для нахождения количества функциональных указателей:

(2.2)

(2.3)

где FPK – это количество функциональных указателей для Kotlin;

FPT – это количество функциональных указателей для TypeScript.

Округляем числа до 208 и 219, соответственно, потому что количество функциональных указателей не может быть дробным.

Таким образом, получаем количество функциональных указателей для Kotlin – 208, а для TypeScript – 219.

### 2.5.2 Расчет LOC-метрик разрабатываемого приложения

После расчета FP переходим к расчетам LOC метрик. LOC – это метрика ПО, используемая для измерения его объема с помощью подсчета количества строк в тексте исходного кода.

Формула для пересчета из FP в LOC приведена ниже:

*,* (2.4)

где FP – это количество функциональных указателей, найденных в прошлом разделе;

КоличествоОператоров – это константа, которая зависит от языка программирования, обозначающая количество операторов на один FP.

Используем соответствующие константы для Kotlin и TypeScript из таблицы 2.4.

Таблица 2.4 – Количество операторов на один FP

|  |  |
| --- | --- |
| Язык программирования | Кол-во операторов на один FP |
| Assembler | 320 |
| C | 128 |
| Fortran | 106 |
| Pascal | 90 |
| C++ | 64 |
| TypeScript | 47 |
| C# | 53 |
| Kotlin | 43 |
| Java | 53 |
| Visual Basic | 32 |
| Visual C++ | 34 |

Таким образом, получаем для Kotlin LOCK = 208 \* 43 = 8944 (строк кода) и для TypeScript LOCT = 219 \* 47 = 10293 (строк кода).

### 2.5.3 Расчет COCOMO-метрик разрабатываемого приложения

COCOMO – это алгоритмическая модель для оценки разработки ПО. Она используется в период, когда стабилизируются требования и определяется базисная программная архитектура.

Основное уравнение этой модели имеет вид:

*,* (2.5)

где А – масштабный коэффициент, равный 2.5;

РАЗМЕР – размер ПО, выраженный в LOC;

МЕ – множитель поправки зависит от семи формирователей затрат, характеризующих продукт, процесс и персонал;

В – отражает нелинейную зависимость затрат от размера проекта (от длины кода LOC).

Значение показателя степени В изменяется в диапазоне от 1,01…1,26, зависит от пяти масштабны факторов Wi и вычисляется по формуле:

.(2.6)

Общая характеристика масштабных факторов Wi приведена в таблице 2.5, где очень низкий фактор (=5), а сверхвысокий (=0).

Таблица 2.5 – Характеристика масштабных факторов Wi

|  |  |  |
| --- | --- | --- |
| Масштабный фактор (Wi) | Пояснение | Wi |
| Предсказуемость | Отражает предыдущий опыт организации в реализации проектов этого типа. | 3 |
| Гибкость разработки | Отражает степень гибкости процесса разработки. | 3 |
| Разрешение рисков в архитектуре | Отражает степень выполняемого анализа риска. | 3 |
| Связность группы | Отражает, насколько хорошо разработчики группы знают друг друга и насколько удачно они совместно работают. | 1 |
| Зрелость процесса | Означает зрелость процесса в организации. | 3 |
| Итого | | 13 |

Вычисляем показатель В:

. (2.7)

Множитель МЕ зависит от набора формирователей затрат ЕМi, перечисленных в таблице 2.6.

Таблица 2.6 – Формирователи затрат ЕМi для раннего этапа проектирования

|  |  |  |
| --- | --- | --- |
| Обозначение | Название | ЕМi |
| PERS | Возможности персонала | Среднее = 1 |
| PCPX | Надежность и сложность продукта | Среднее = 1 |
| RUSE | Требуемое повторное использование | Среднее = 1 |
| PDIF | Трудность платформы | Среднее = 1 |
| PREX | Опытность персонала | Среднее = 1 |
| FCIL | Средства поддержки | Среднее = 1 |
| SCED | Сроки | Среднее = 1 |

Для расчета множителя поправки, зависящего от семи формирователей затрат, характеризующих продукт, процесс и персонал, воспользуемся формулой:

. (2.8)

Для расчета затрат на разрабатываемый продукт, воспользуемся формулой, составные части которой, мы уже подсчитали.

*.* (2.9)

*.* (2.10)

Таким образом, исходя из расчетов, выяснили, что один человек может выполнить поставленную задачу создания приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы почти за пять месяцев.

## **2.6 Проектирование базы данных** **приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

В техническом задании на разработку приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы в пункте 1.3 было упомянуто использование БД в качестве хранилища данных, с которыми пользователь будет работать, а в пункте 1.4.3 в качестве такой БД была выбрана и описана СУБД PostgreSQL.

При проектировании БД мы опирались на учебное пособие Е.П. Моргунова «PostgreSQL. Основы языка SQL» [16].

В модели БД для каждого простого класса форматируется таблица, которая включает столбцы, поставленные в соответствие атрибутам класса [17].

В основу проектирования легла задача создания отдельной таблицы для каждой модели: задача, пользователь и группа. Из-за того, что один пользователь мог быть в нескольких группах и в одной группе может быть несколько пользователей, создано отношение «многие ко многим». Схема БД представлена на рисунке ниже (Рисунок 2.9).

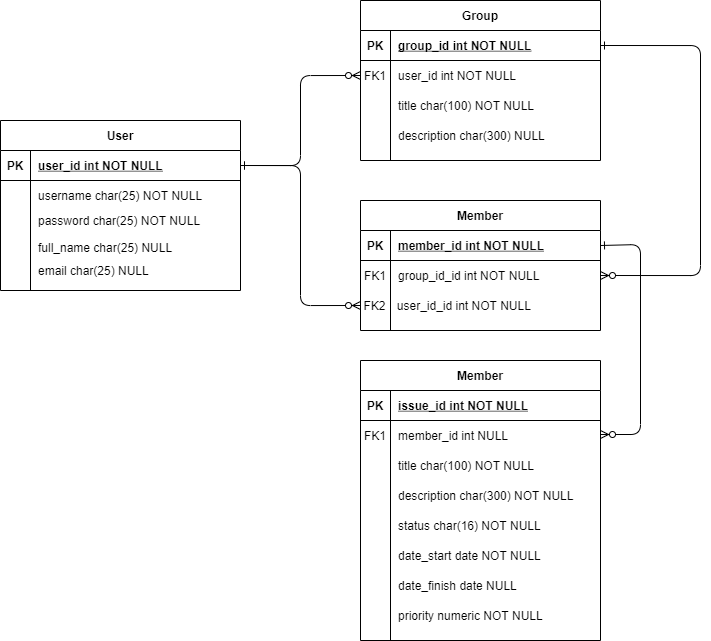


Рисунок 2.9 – Схема базы данных разрабатываемого приложения

Из схемы БД на рисунке 2.9 видно, что было создано четыре таблицы:

1. User – таблица, которая содержит данные о пользователе, в том числе и для его авторизации. Содержит четыре поля и первичный ключ (в сумме пять полей).
2. Group – таблица, в которой содержится информация о группе пользователей. Содержит три поля, одно из которых внешний ключ к таблице User, и первичный ключ (в сумме четыре поля).
3. Member – таблица, позволяющая создание отношения «многие ко многим». Содержит в себе первичный ключ и два внешних ключа – один к таблице User, другой к таблице Group. Такой подход позволяет найти всех пользователей в группе и все группы, в которых состоит пользователь.
4. Issue – таблица, в которой содержится информация о задачах. Содержит первичный ключ и шесть полей, одно из которых внешний ключ к таблице Member.

БД должна храниться отдельном сервере, чтобы при сбое сервера, например, на котором будет располагаться серверная часть, БД была в порядке.

Таким образом, спроектирована БД приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы, в которой содержится четыре таблицы.

## **2.7 Вывод по второму разделу**

В данном разделе при проектировании приложения для мониторинга и создания задач рабочей группы был выполнен анализ требований к информационной системе на основе диаграммы прецедентов, диаграммы классов и диаграммы последовательностей. Также был произведен анализ рисков и трудозатрат при помощи FP-метрик, LOG- оценок и COCOMO-метрик. Исходя из расчетов, выяснили, что один человек может выполнить поставленную задачу создания приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы почти за пять месяцев.

Для разработки БД проекта был проведен этап проектирования логической модели, в котором рассмотрены все атрибуты и сущности, необходимые для должного функционирования системы.

# 3 Программная реализация приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы

## **3.1 Веб-интерфейс приложения** **для создания и мониторинга задач рабочей группы**

Веб-интерфейс приложения для создания и мониторинга задач рабочей группы содержит в себе множество полей ввода и вывода. В этом пункте будут описаны все элементы ввода и вывода интерфейса.

В окне авторизации находится два поля для ввода текста (далее - текстбокс) для ввода логина и пароля пользователя (Рисунок 3.1). Кроме того, панель содержит в себе две кнопки – одна для перехода на панель регистрации («Регистрация»), вторая для инициации авторизации («Войти»).

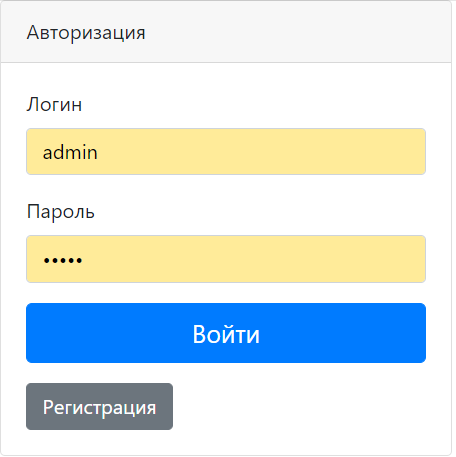


Рисунок 3.1 – Панель авторизации

На панели регистрации расположено пять кнопок для ввода имени, электронной почты, логина и пароля с его проверкой (Рисунок 3.2). Кнопка «Авторизация» вернет пользователя на панель авторизации, а кнопка «Зарегистрироваться» отправит запрос на регистрацию пользователя на сервер.

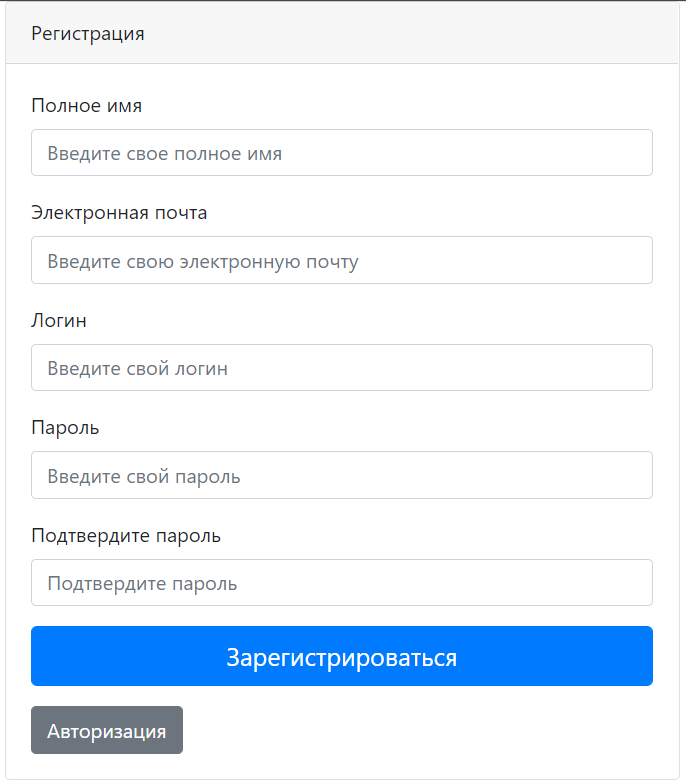


Рисунок 3.2 – Панель регистрации

На главной странице находится информация авторизированного пользователя – его ID, логин, электронная почта и полное имя (Рисунок 3.3). Кроме того, сверху находится панель навигации, состоящая из пяти пунктов: «Главная», «Мои задачи», «Мои группы», «Администрирование» и «Выход». Панель навигации доступна на каждой странице, кроме панели авторизации и регистрации.

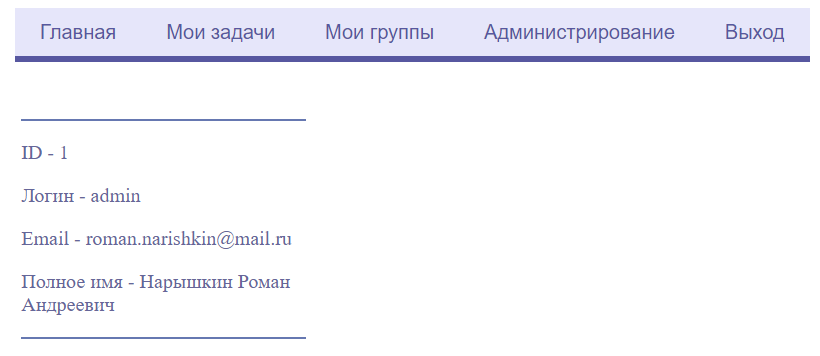


Рисунок 3.3 – Главная страница

На странице «Мои задачи» находится фильтр задач и таблица с задачами пользователя (Рисунок 3.4). Фильтр содержит в себе четыре текстбокса и одно поле ввода типа «select» для выбора статуса поисковых задач. Текстбоксы позволяют находить задачи по ID, заголовку, описанию и группе, создавшую задачу. Кнопка на фильтре нужны для следующего:

* «Найти» - активирует поиск задач;
* «Скрыть выполненные» - скрывает задачи со статусом «Done»;
* «Показать выполненные» - отменяет фильтрацию задач через предыдущую кнопку.

В таблице «Мои задачи» находятся все задачи, назначенные пользователю. В качестве столбцов используются следующие данные о задачи:

* ID задачи;
* заголовок;
* описание;
* статус;
* приоритет;
* дата начала;
* дата конца;
* группа.

Дополнительно в таблице находится столбец для кнопки «Изменить статус», которая меняет статус задачи согласно методологии Kanban [29].

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.4 – Панель задач пользователя

На панели групп пользователя находится форма для создания группы и таблица с группами пользователя (Рисунок 3.5). Форма для создания группы содержит два текстбокса для названия и описания группы, соответственно. Кнопка «Создать» отправляет запрос о создании группы на сервер.

В таблице «Мои группы» представлены группы пользователя. В качестве столбцов используются следующие данные о группе:

* ID группы;
* заголовок;
* описание.

Кроме того, дополнительно находится столбец, в котором пользователя извещают о том, что он является администратором группы.

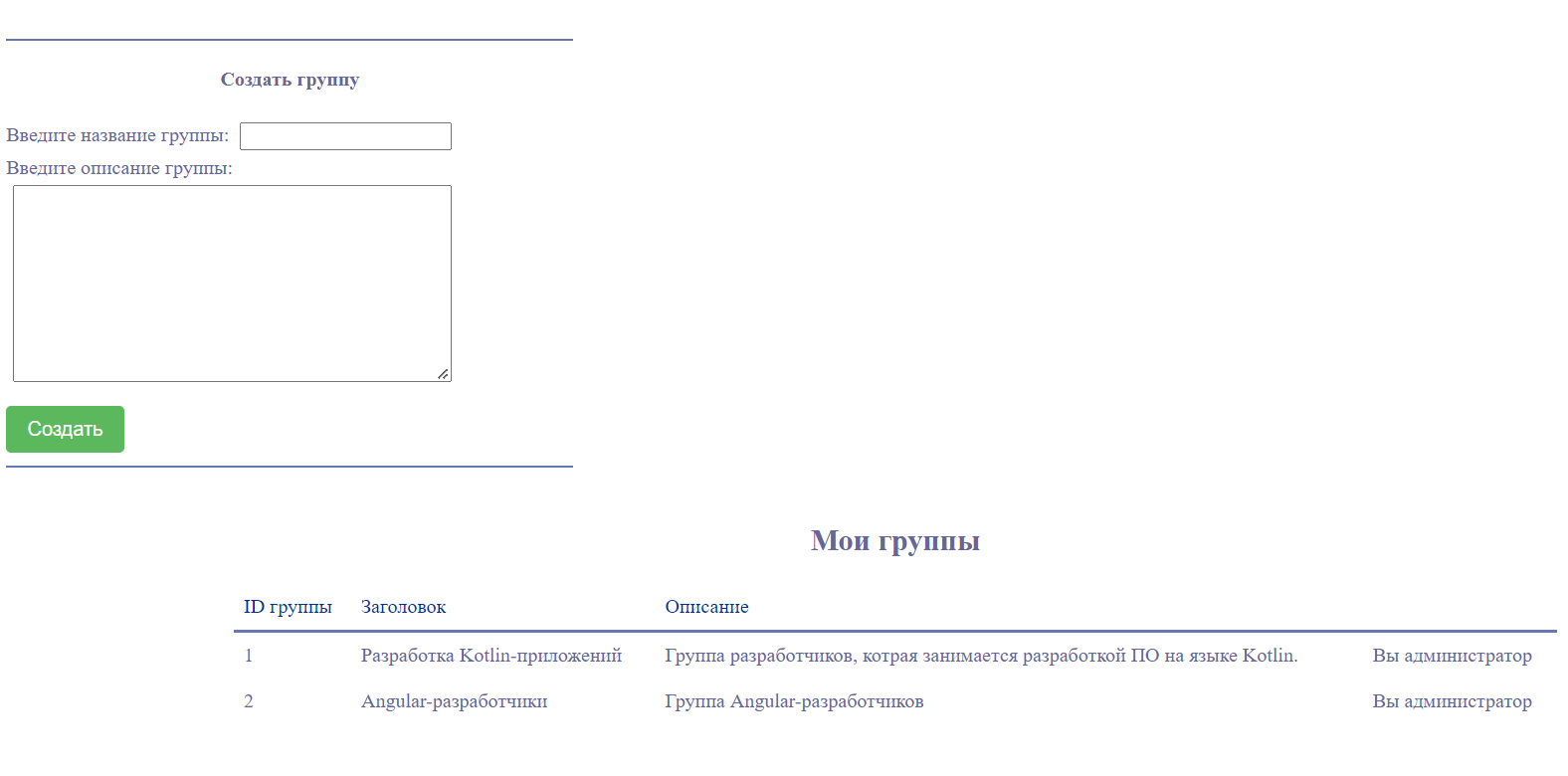


Рисунок 3.5 – Панель групп пользователя

На панели администратора групп (Рисунок 3.6) находится похожая таблица групп, как на панели групп пользователя. Однако, здесь в дополнительном столбце пользователь может выбрать администрируемую группу и перейти в специальное окно.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.6 – Панель администратора групп

Панель управления группой содержит множество компонентов, которые можно разделить на три группы (согласно диаграмме прецедентов, см. раздел 2.1) – управление непосредственно группой, управление участниками группы, управление задачами группы.

«Управление непосредственно группой» (Рисунок 3.7) содержит в себе поле с информацией о группе – ID, название и описание группы. Кнопки этой панели:

* «Скачать отчет» - скачивает отчет в формате .xlsx о задачах группы с подробной информацией;
* «Удалить группу» - удаляет группу.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.7 – Управление непосредственно группой

«Управление участниками группы» (Рисунок 3.8) включает в себя следующие компоненты: форма для добавления пользователя в группу, таблица для вывода информации об участницах группы.

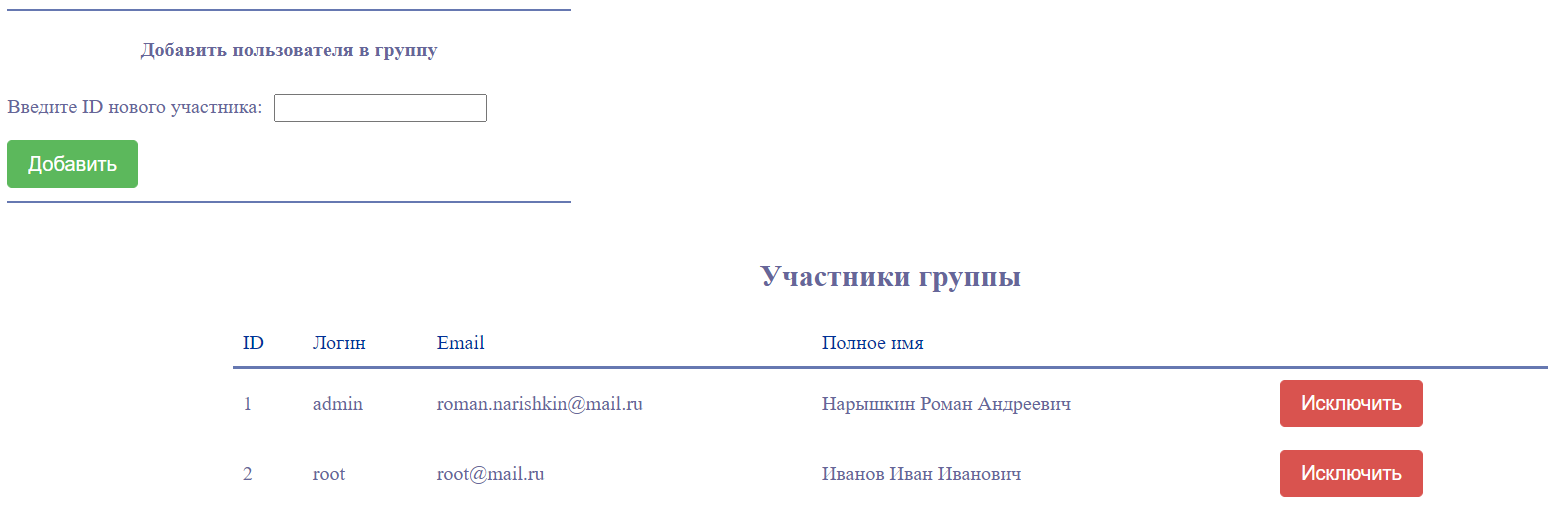


Рисунок 3.8 – Управление участниками группы

Добавление пользователя в группу осуществляется с помощью текстбокса для ID нового участника и кнопкой «Добавить». В таблице «Участники группы» столбцы содержать следующие данные:

* ID участника;
* логин;
* электронная почта;
* полное имя.

Дополнительно в таблице находится столбец с кнопками для исключения пользователей из группы.

«Управление задачами группы» включает в себя форму для создания задачи (Рисунок 3.9), форму для назначения задачи (Рисунок 3.9) и таблицей задач группы с фильтром (Рисунок 3.10).

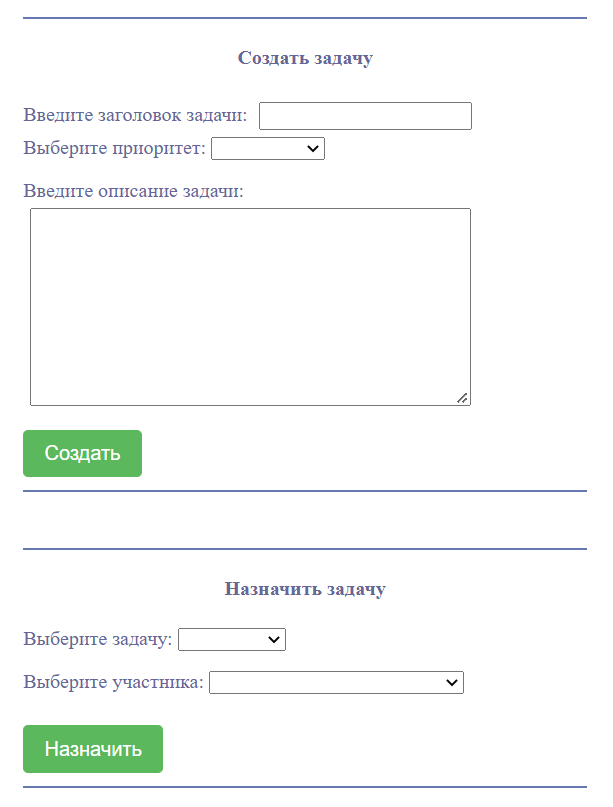


Рисунок 3.9 – Форма для создания задачи и форма для назначения задачи

На форме для создания задачи (Рисунок 3.9) находится да текстбокса для названия и описания задачи, поле ввода типа «select» позволяет выбрать приоритет задачи, кнопка «Создать» отправляет запрос на создание задачи на сервер.

На форме для назначения задачи (Рисунок 3.9) находится два поля ввода типа «select» - одно для выбора задачи, второе для выбора пользователя-исполнителя. Кнопка «Назначить» отправляет запрос назначения задачи на сервер.

Фильтр и таблица задач группы (Рисунок 3.9) аналогичны фильтру и таблице задач «Мои задачи» (Рисунок 3.4). Однако, в таблицах присутствует различие в столбцах:

* отсутствует кнопка «Изменить статус»;
* добавлены столбцы «Имя» и «Почта» с именем и почтой исполнителя задачи.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.10 – Таблица задач группы

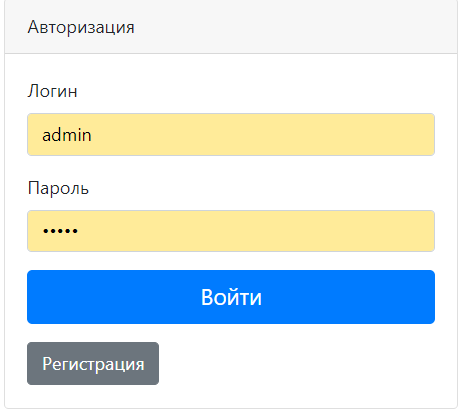
Таким образом, рассмотрен созданный веб-интерфейс приложения для создания и мониторинга задач рабочей группы: описано каждое поле ввода и вывода данных, показаны скриншоты всех компонентов интерфейса.

## **3.2 Руководство пользователя приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы**

Приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы обладает простым функционалом. В этом пункте будет описано руководство пользователя для этого приложения.

При переходе по URL пользователь увидит окно авторизации (Рисунок 3.1), если пользователь не имеет свой учетной записи, он может перейти в панель регистрации (кнопка «Регистрация») и зарегистрироваться, предварительно введя своё имя, электронную почту, логин и пароль (Рисунок 3.2). В поле «Подтвердите пароль» необходимо ввести такой же пароль, как и в поле «Пароль». После успешной регистрации пользователь сможет авторизироваться в приложении.

После авторизации пользователь будет перенаправлен на главную страницу, в которой выведена информация об авторизированном пользователе (Рисунок 3.11).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.11 – Сравнение логина в окне авторизации и в профиле авторизированного пользователя

На рисунке выше видно, что логин при авторизации идентичен логину на главной странице приложения.

На панели задач пользователя находится таблица задач авторизированного пользователя и фильтр таблицы. Для фильтрации задач можно использовать любой текстбокс, поля для ввода типа «select» для фильтрации по статусу задачи, кнопки для скрытия или показа выполненных задач. Кнопка «Найти» активирует фильтрацию. Сравним задачи в исходном состоянии (Рисунок 3.12) с задачами при фильтрации по ID (Рисунок 3.13), статусу (Рисунок 3.14) и при скрытии выполненных задач (Рисунок 3.15).

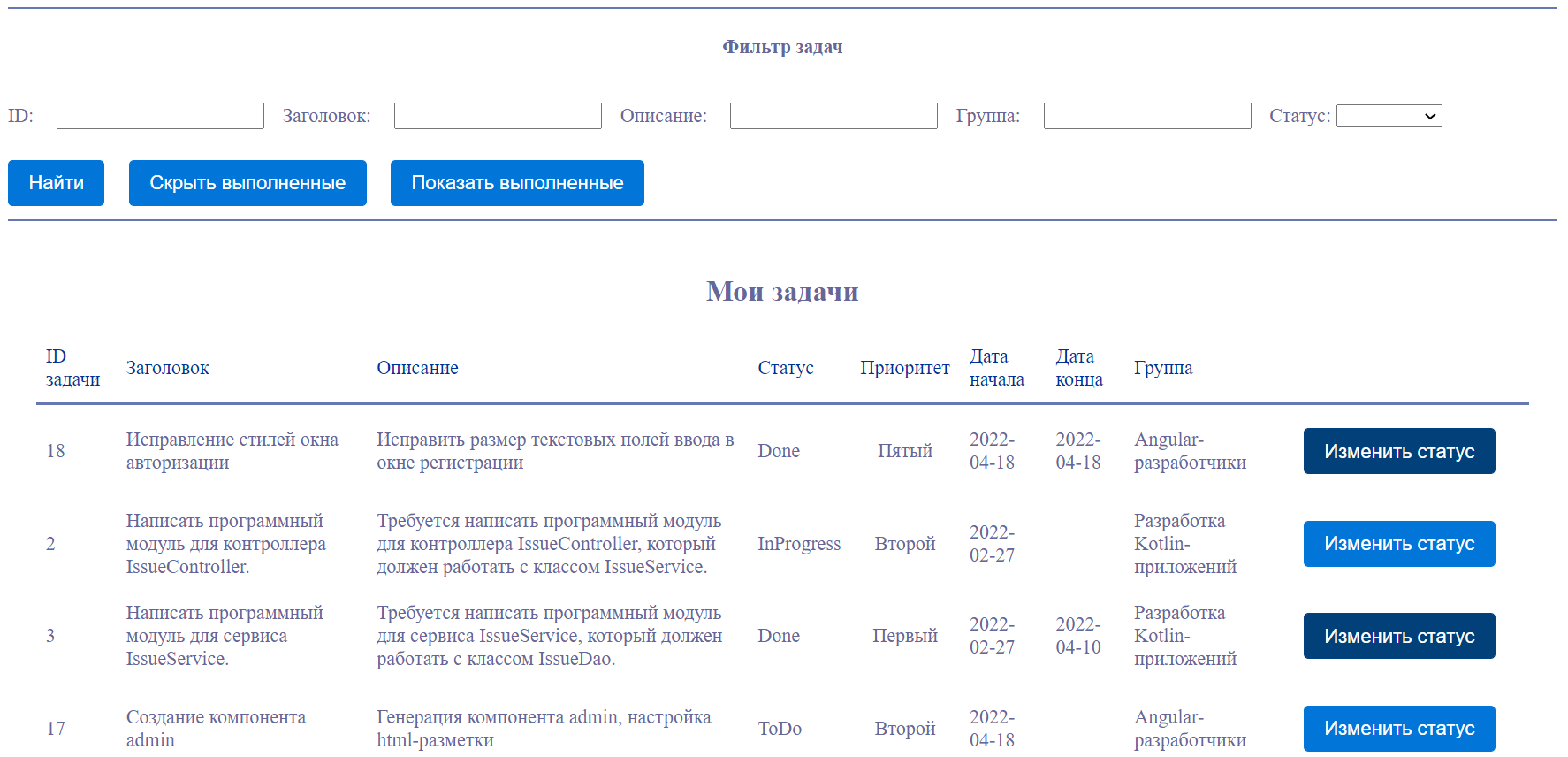


Рисунок 3.12 – Исходное состояние пользователя admin

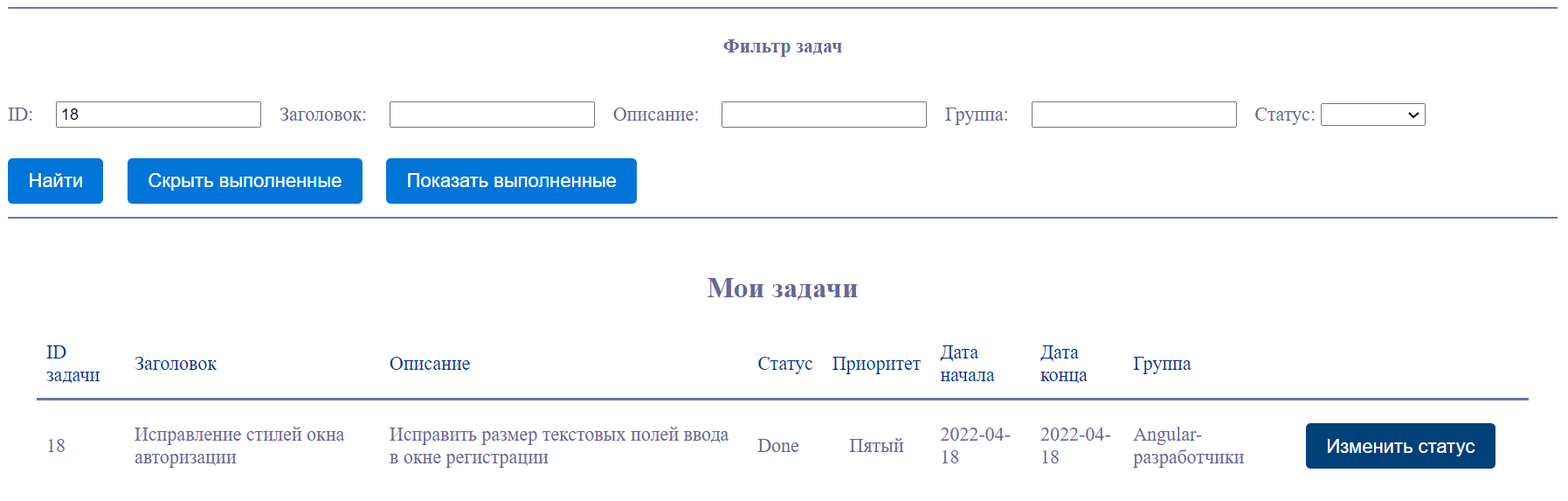


Рисунок 3.13 – Фильтрация задач по ID = 18

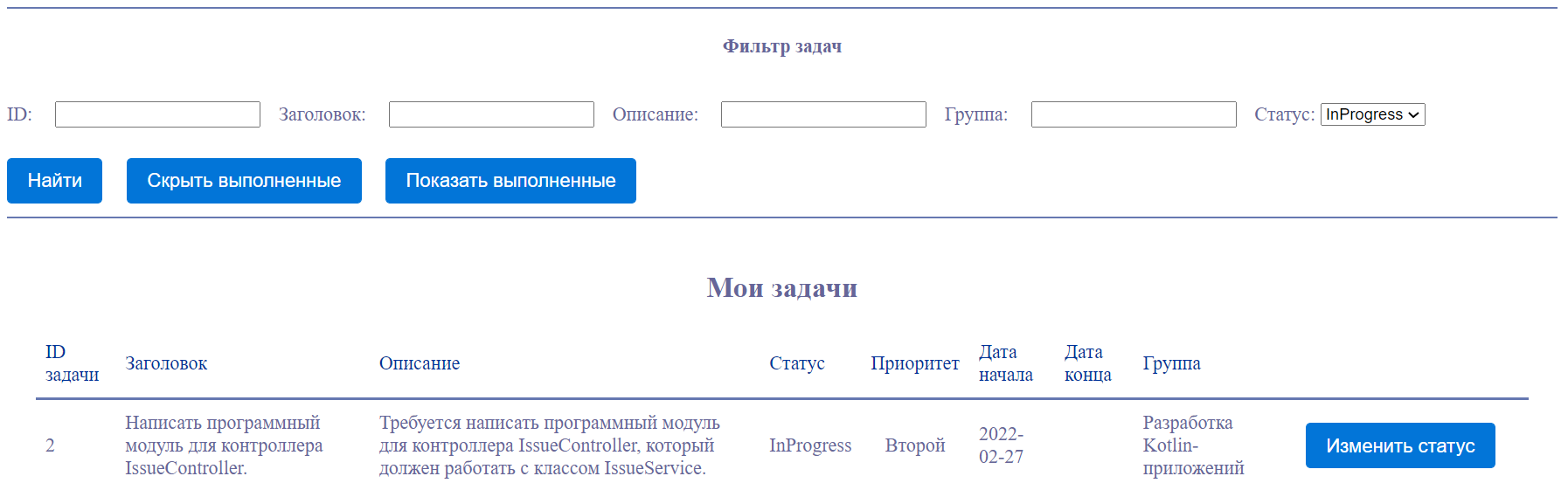


Рисунок 3.14 – Фильтрация задач по статусу «InProgress»

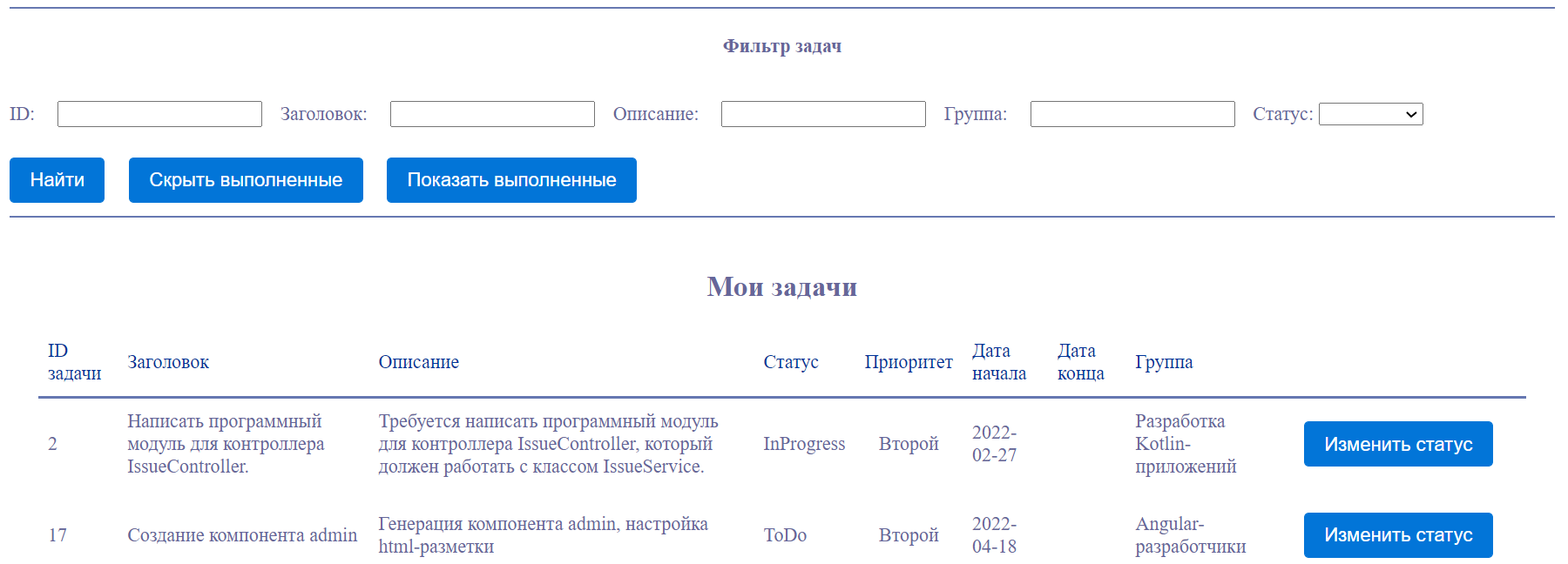


Рисунок 3.15 – Фильтрация задач с помощью кнопки «Скрыть выполненные»

На панели групп пользователя можно увидеть свои группы и создать группу. Сравним исходные группы пользователя admin (Рисунок 3.16) с группами после создания новой группы (Рисунок 3.17–3.18).

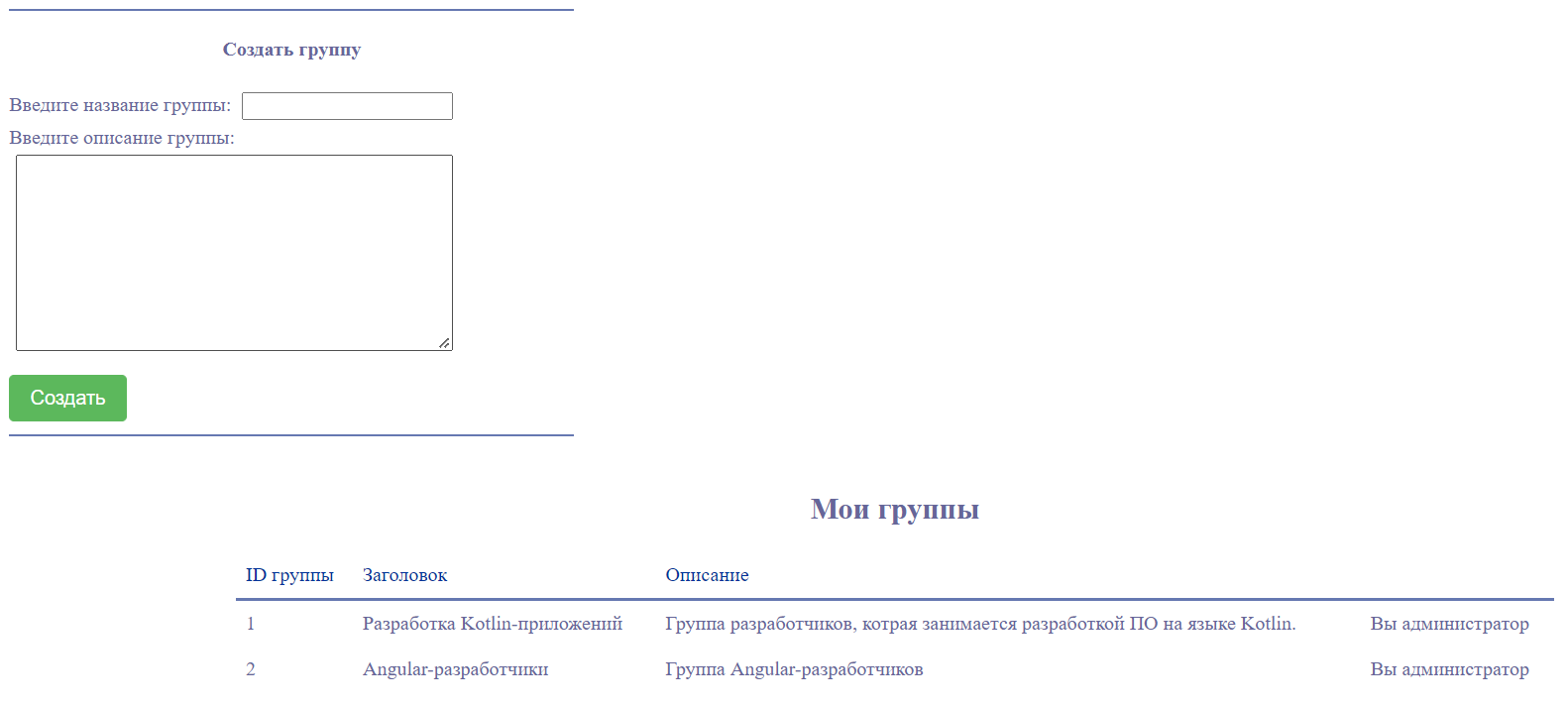


Рисунок 3.16 – Исходные группы пользователя

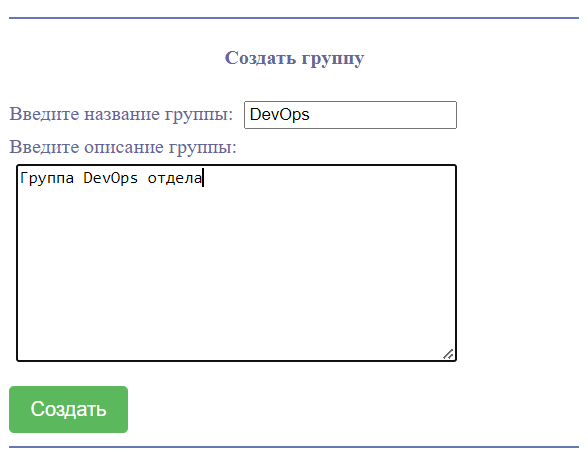


Рисунок 3.17 – Заполненная форма для создания новой группы

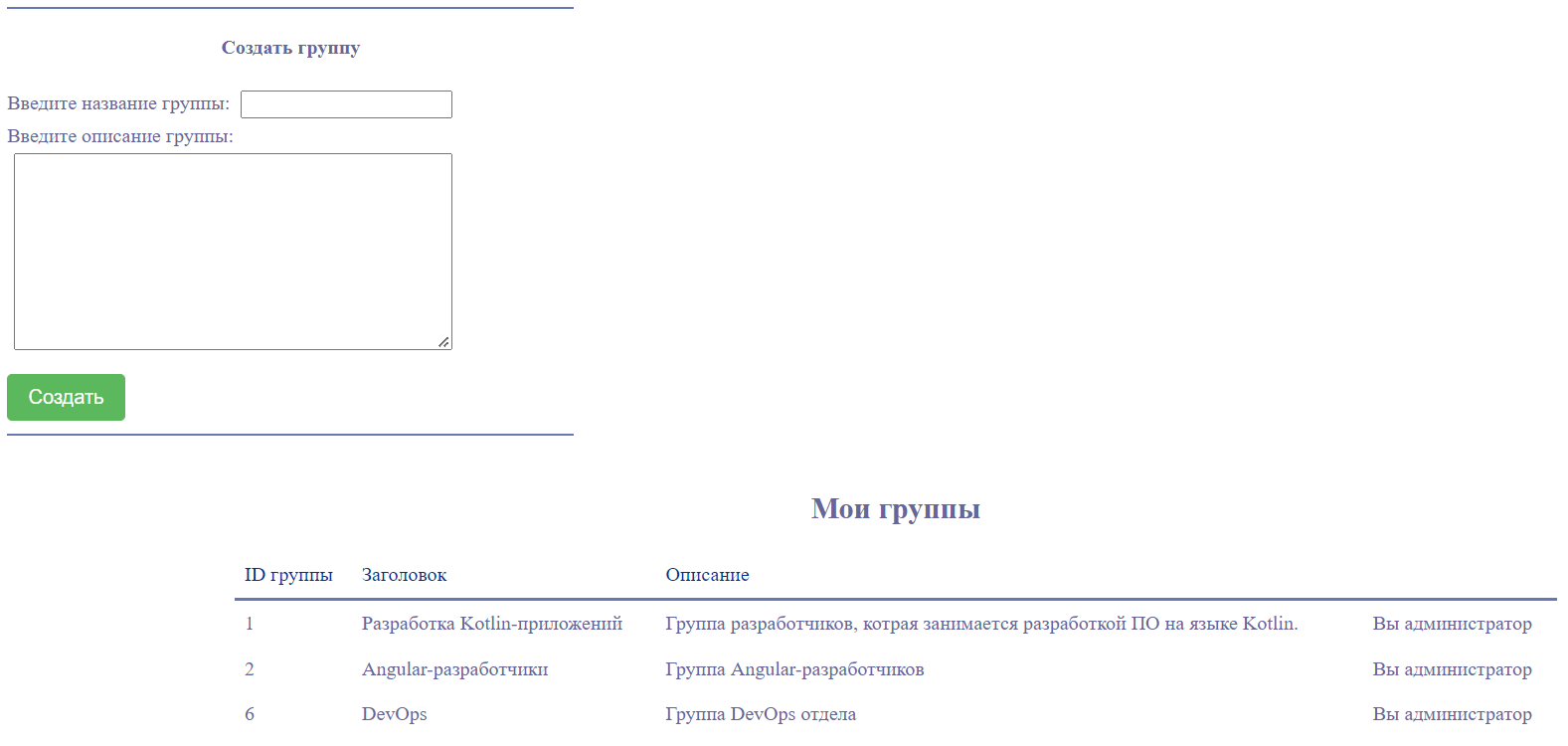


Рисунок 3.18 – Измененная таблица групп пользователя

Как видно из рисунка выше (Рисунок 3.18), после создания группы, авторизированный пользователь становится администратором созданной группы. На рисунке ниже (Рисунок 3.19) представлена таблица групп для пользователя – участника.

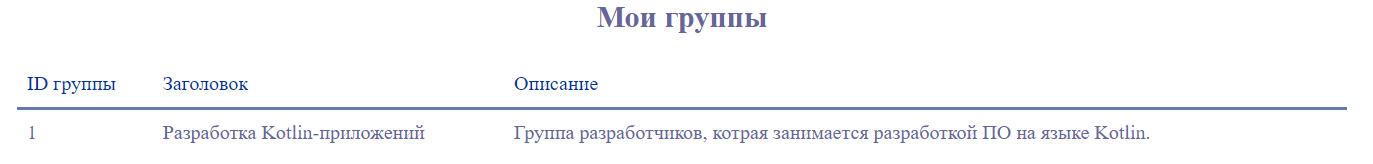


Рисунок 3.19 – Таблица групп пользователя - участника

На панели администратора групп представлена та же таблица групп, но с возможностью управления группой, если у авторизированного пользователя есть администрируемые группы.

На панели управления группой возможно совершить ряд действий:

* скачать отчет по задачам;
* удалить группу;
* добавить участника;
* исключить участника;
* создать задачу;
* назначить задачу.

Рассмотрим перечень действий подробнее.

Кнопка «Скачать отчет» позволяет скачать отчет по задачам группы в формате .xlsx (Рисунок 3.20).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.20 – Созданный и скаченный отчет по задачам группы в формате .xlsx

Сравним полученный отчет с таблицей задач группы на панели управления той же группы (Рисунок 3.21).

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.21 – Таблица задач группы

После сравнения таблица задач группы (Рисунок 3.21) и скачанного отчета (Рисунок 3.20) видно, что в скачанном отчете отсутствует поле ID, т. к. во время отчетности это поле не имеет особой важности. В остальном таблицы идентичны.

Кнопка «Удалить группу» удаляет группу, после чего она не будет отображаться в таблице «Мои группы» на панели групп пользователя.

Добавление пользователя в группу реализуется через ввод ID нужного пользователя. Для этого пользователь, которого администратор хочет добавить в свою группу, должен найти свой ID на главной странице приложения, на рисунке 3.11 ID = 1. Добавим пользователя с ID=3 (Рисунок 3.22) и сравним таблицу участников группы до (Рисунок 3.23) и после (Рисунок 3.24) добавления.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.22 – Форма для добавления пользователя с ID=3 в группу

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.23 – Первоначальная таблица участников группы

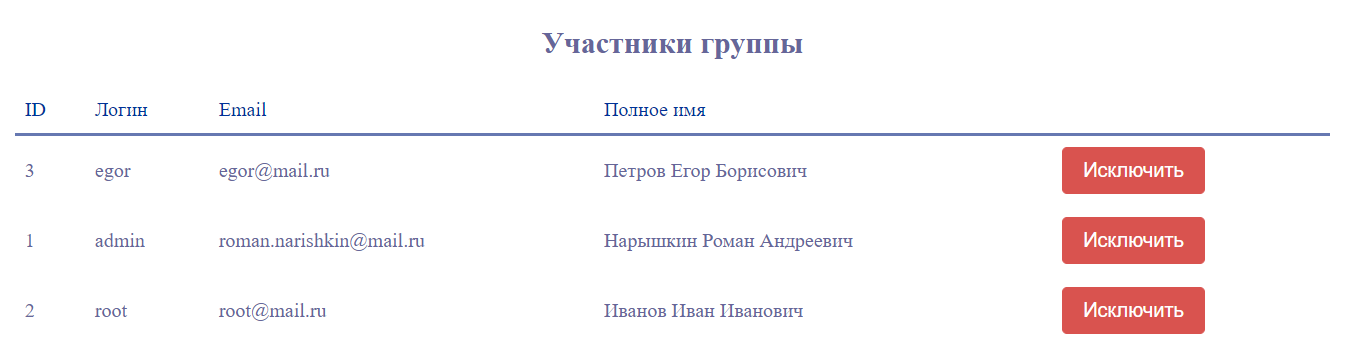


Рисунок 3.24 – Измененная таблица участников группы

После проведенной процедуры видно, что пользователь «Петров Егор Борисович» добавился в группу. Кнопка «Исключить» удаляет пользователя из группы, после чего данная группа не будет отображаться у него в таблице «Мои группы», как и назначенные задачи этой группой.

Создание задачи осуществляется в одноименном окне. Создадим задачу (Рисунок 3.25) и сравним таблицу задач группы до (Рисунок 3.26) и после (Рисунок 3.27) операции.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.25 – Заполненная форма создания задачи

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.26 – Исходная таблица задач группы

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.27 – Обновленная таблица задач группы

Как видно из рисунков 3.26 и 3.27, задача была успешно создана. Созданная задача (ID=19) в столбце «Имя» имеет значение «Не назначено» и пустое значение в столбце «Почта». Это означает, что у задачи нет назначенного пользователя-исполнителя в группе. Чтобы назначить задачу пользователю, нужно в специальной форме выбрать задачу (Рисунок 3.28) и пользователя-исполнителя (Рисунок 3.29). В этой форме можно выбрать только не назначенную задачу, но пользователи будут доступны все.

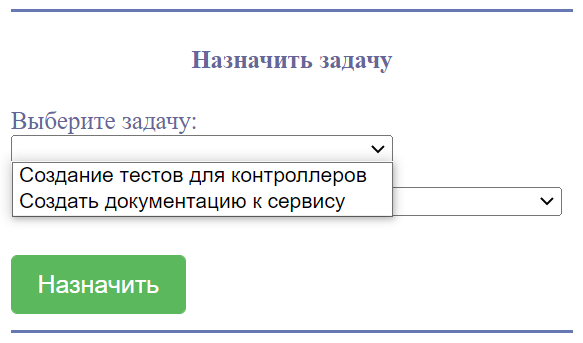


Рисунок 3.28 – Не назначенные задачи

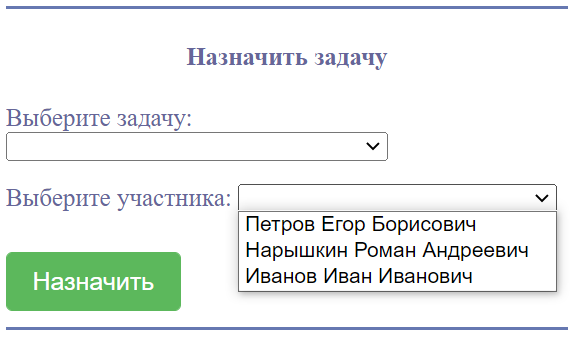


Рисунок 3.29 – Доступные пользователи для исполнения задачи

После назначения задачи сравним задачи до (Рисунок 3.27) и после (Рисунок 3.30).



Рисунок 3.30 – Обновленная таблица задач группы

После назначения задач, отчет, который получает администратор через кнопку «Скачать отчет», будет изменен.

Таким образом, приведено руководство пользователя приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы. Были приведены подробные описания для каждой функции приложения со скриншотами для проверки изменений данных.

## **3.3 Выводы по третьему разделу**

Разработано приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы. Приложение состоит из двух частей: фронтенд (веб-интерфейс) и бэкенд (серверная часть). Для веб-интерфейса был использован фреймворк Angular CLI на языке TypeScript. Серверная часть написана на языке Kotlin с использованием фреймворка Spring. Проверена работа программного средства, описан интерфейс, составлено руководство пользователя со скриншотами программы.

# Заключение

В ходе выполнения выпускной квалификационной работы было разработано приложение с веб-интерфейсом, позволяющее создавать и отслеживать задачи рабочей группы. Кроме того, существует возможность создания и управления этой группы.

Целевая аудитория приложения – это люди, объединенные одной целью. Это могут быть разработчики ПО, аналитики, фрилансеры, дизайнеры и т. д. и т. п.

Цель выпускной квалификационной работы достигнута, а именно разработано приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы. В ходе достижения этой цели были выполнены следующие задачи:

1. Произведен анализ предметной области разрабатываемого приложения с веб-интерфейсов для создания и мониторинга задач рабочей группы.
2. Спроектировано приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы.
3. Реализовано в программном виде приложение с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Каждая из вышеперечисленных задач содержит в себе подзадачи. При анализе предметной области получены следующие результаты:

1. Выделена актуальность разрабатываемого приложения.
2. Произведен анализ существующих приложений для создания и мониторинга рабочей группы.
3. Сформировано техническое задание на разработку приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
4. Выбрана среда разработки для создания приложения (выбрана среда разработки серверной части приложения, веб-интерфейса и СУБД).

Во время проектирования приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы получены следующие результаты:

1. Произведен анализ требований разработки на основе построения диаграмм прецедентов приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
2. Определен главный сценарий взаимодействия.
3. Построена диаграмму классов приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
4. Построена диаграмму последовательности для приложения с веб-интерфейсом для создания и мониторинга рабочей группы.
5. Произведен анализ рисков при разработке приложения с веб-интерфейсом для создания и мониторинга рабочей группы (рассчитаны FP-метрики, LOC-метрики, COCOMO-метрики).
6. Спроектирована база данных приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

При программной реализации приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы получены следующие результаты:

1. Описана среда разработки программного средства приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
2. Создан веб-интерфейс приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.
3. Сформировано руководство пользователя приложения с веб-интерфейсом для создания и мониторинга задач рабочей группы.

Подводя итоги проведенной работы, можно сказать, что все поставленные изначально задачи были решены и, следовательно, главная цель также была достигнута.

В рамках бакалаврской работы:

1. Реализовано распределение задач между командой.
2. Разработаны методы отслеживания хода выполнения задач.
3. Создана возможность управления задачами проекта руководителем.

В дальнейшем предполагается совершенствования работы приложения.

# Список используемой литературы

1. Как уследить за всеми и всем: восемь удобных таск-менеджеров для командной работы // Нетология медиа URL: https://netology.ru/blog/311-kak-usledit-za-vsemi-i-vsem-shest-udobnykh-task-menedjerov-dlya-komandnoy-raboty (дата обращения: 12.02.2022).
2. Как работать в Jira // Комьюнити URL: https://timeweb.com/ru/community/articles/kak-rabotat-v-jira (дата обращения: 12.02.2022).
3. Redmine // Startpack URL: https://startpack.ru/application/redmine (дата обращения: 12.02.2022).
4. A modern programming language that makes developers happier. // Kotlin Programming Language URL: https://kotlinlang.org/ (дата обращения: 15.03.2022).
5. Why Spring? // Spring URL: https://spring.io/why-spring (дата обращения: 17.03.2022).
6. **Жемеров Д., Исакова С.** Kotlin в действии . - М.: ДМК Пресс, 2018. - 402 с.
7. Kotlin против Python — какой язык программирования выбрать? // Python 3 URL: https://python-scripts.com/kotlin-vs-python#kotlin-vs-python (дата обращения: 17.03.2022).
8. Какие компании используют Kotlin? // Kotlin URL: https://kotlinlang.ru/docs/reference/faq.html (дата обращения: 19.03.2022).
9. Kotlin или Golang: Что лучше для разработки приложений? // Golang URL: https://golangify.com/kotlin-vs-golang (дата обращения: 17.03.2022).
10. Java и Kotlin: что будет лучшим выбором в 2019 году? // TECH URL: https://techrocks.ru/2019/01/28/java-vs-kotlin-best-language-for-android-in-2019/ (дата обращения: 17.03.2022).
11. Топ-10 фреймворков для разработки на Java // ИТ в Беларуси URL: https://golangify.com/kotlin-vs-golang (дата обращения: 17.03.2022).
12. The modern web developer’s platform // Angular URL: https://angular.io/ (дата обращения: 17.03.2022).
13. ng-book // ng-book URL: https://www.newline.co/ng-book/2/ (дата обращения: 17.03.2022).
14. **Файн Я.Б., Моисеев А.** Angular и TypeScript. Сайтостроение для профессионалов . — СПб.: Питер, 2018. = 446.
15. Сравнение современных СУБД // Сайт Владимира Драча URL: : https://drach.pro/blog/hi-tech/item/145-db-comparison (дата обращения: 17.03.2022).
16. **Моргунов, Е. П.** PostgreSQL. Основы языка SQL . СПб.: БХВ-Петербург, 2018. — 336 с.
17. Основы проектирования информационных систем [Электронный документ] https://books.ifmo.ru/file/pdf/1705.pdf. Проверено 16.04.2022.
18. Trello // Startpack URL: https://startpack.ru/application/ fog-trello (дата обращения: 12.02.2022).
19. Basecamp // Startpack URL: https://startpack.ru/application/basecamp (дата обращения: 12.02.2022).
20. Wrike // Startpack URL: https://startpack.ru/application/wrike-project-management (дата обращения: 12.02.2022).
21. Размерно-ориентированные метрики [Электронный документ] (http://dsplab.narfu.ru/popov/files/edu/devel/complexity.pdf) Проверено 14.04.2022.
22. **Мартин Р.** Чистая архитектура. СПб.: Питер, 2020. – 352 с.
23. **Урма Р-Г., Уорбертон Р**. Гид Java-разработчика. Проектно-ориентированный подход. Москва: Эксмо, 2022. – 224 с.
24. **Мартин Р**. Чистый код. СПб.: Питер, 2021. – 464 с.
25. Подберите себе инструмент // JetBrains URL: https://www.jetbrains.com/ru-ru/products/ (дата обращения: 17.04.2022).
26. JetBrains // Wikipedia URL: https://en.wikipedia.org/wiki/JetBrains (дата обращения: 17.04.2022).
27. Intellij IDEA // JetBrains URL: https://www.jetbrains.com/ru-ru/idea/ (дата обращения: 17.04.2022).
28. WebStorm // JetBrains URL: https://www.jetbrains.com/ru-ru/webstorm/ (дата обращения: 17.04.2022).
29. Методология Kanban: доски, принципы и возможности управления // Skillbox URL: https://skillbox.ru/media/management/vse\_chto\_nuzhno\_znat\_o\_kanban/ (дата обращения: 17.04.2022).

# Приложение А

(обязательное)

Листинг кода

SecurityConfig.kt

@EnableWebSecurity

class SecurityConfig(

val dataSource: DataSource

) : WebSecurityConfigurerAdapter() {

override fun configure(http: HttpSecurity) {

http.cors()

http.csrf().disable()

http.authorizeRequests().antMatchers("/\*\*").fullyAuthenticated().and().httpBasic()

}

public override fun configure(auth: AuthenticationManagerBuilder) {

auth.jdbcAuthentication()

.dataSource(dataSource)

.usersByUsernameQuery(

"select username, password, 'true' from users where username=?"

)

.authoritiesByUsernameQuery(

"select username, 'ROLE\_USER' from users where username=?"

)

}

@Bean

fun passwordEncoder() = BCryptPasswordEncoder()

}

GroupController.kt

private val logger = KotlinLogging.logger {}

@RestController

@CrossOrigin(origins = ["\*"])

class GroupController(

private val groupService: GroupService

) {

@PostMapping("/group/createGroup")

fun createGroup(@RequestBody request: CreateGroupRequest) = try {

groupService.createGroup(request)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/deleteGroup/{id}")

fun deleteGroup(@PathVariable id: Long) = try {

groupService.deleteGroup(id)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/getGroup/{id}")

fun getGroup(@PathVariable id: Long) = try {

groupService.getGroup(id)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@PostMapping("/group/addMember")

fun addMember(@RequestBody request: AddMemberRequest) = try {

groupService.addMember(request)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/deleteMember/{groupId}/{userId}")

fun deleteMember(@PathVariable groupId: Long, @PathVariable userId: Long) = try {

groupService.deleteMember(groupId, userId)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/getMembersGroup/{id}")

fun getMembersGroup(@PathVariable id: Long) = try {

groupService.getMembersGroup(id)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/getGroupsByMember/{id}")

fun getGroupsByMember(@PathVariable id: Long) = try {

groupService.getGroupsByMember(id)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/group/getReport/{id}")

fun getReport(@PathVariable id: Long) = try {

groupService.sendReport(id)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

}

IssueController.kt

private val logger = KotlinLogging.logger {}

@RestController

@CrossOrigin(origins = ["\*"])

class IssueController(

private val issueService: IssueService

) {

@GetMapping("/issue/get/byUser/{userId}")

fun getIssuesByUser(@PathVariable userId: Long) = try {

issueService.getIssuesByUser(userId)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/issue/get/byGroup/{groupId}")

fun getIssuesByGroup(@PathVariable groupId: Long) = try {

issueService.getIssuesByGroup(groupId)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@PostMapping("/issue/changeStatus")

fun changeStatusIssue(@RequestBody request: ChangeIssueStatusRequest) = try {

issueService.changeStatusIssue(request)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@PostMapping("/issue/create")

fun createIssue(@RequestBody request: CreateIssueRequest) = try {

issueService.createIssue(request)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@PostMapping("/issue/assignIssue")

fun assignIssue(@RequestBody request: AssignIssueRequest) = try {

issueService.assignIssue(request)

ResponseEntity<String>(HttpStatus.OK)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

}

UserController.kt

private val logger = KotlinLogging.logger {}

@RestController

@CrossOrigin(origins = ["\*"])

class UserController(

private val userService: UserService

) {

@PostMapping("/user/registration")

fun registration(@RequestBody request: RegistrationUserRequest) = try {

userService.registration(request)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/user/auth")

fun auth() = try {

val auth: Authentication = SecurityContextHolder.getContext().authentication

userService.getIdByName(auth.name)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

@GetMapping("/user/getInfo/{id}")

fun getInfo(@PathVariable id: Long) = try {

userService.getInfo(id)

} catch (e: Exception) {

logger.error { e.message }

ResponseEntity<String>(HttpStatus.BAD\_REQUEST)

}

}

GroupDao.kt

private val logger = KotlinLogging.logger {}

@Repository

class GroupDao(

private val jdbcTemplate: JdbcTemplate

) {

fun insertGroup(request: CreateGroupRequest) = try {

val insertGroup = "insert into groups(" +

" user\_id, title, description)" +

" values (${request.userId}, '${request.title}','${request.description}');"

jdbcTemplate.execute(insertGroup)

val selectIdNewGroup = "select group\_id from groups " +

" where user\_id = ${request.userId} and " +

" title like '${request.title}' and " +

" description like '${request.description}' " +

" limit 1"

val insertAdmin = "insert into members(" +

" group\_id, user\_id)" +

" values ( ($selectIdNewGroup), '${request.userId}');"

jdbcTemplate.execute(insertAdmin)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun deleteFromGroup(id: Long) = try {

val sqlDeleteMembers = "delete from members " +

" where group\_id = $id"

val sqlDeleteGroup = "delete from groups " +

" where group\_id = $id"

jdbcTemplate.execute(sqlDeleteMembers)

jdbcTemplate.execute(sqlDeleteGroup)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectFromGroup(id: Long) = try {

val sql = "select \* from groups " +

" where group\_id = $id"

jdbcTemplate.queryForObject(sql) { rs: ResultSet, \_: Int ->

Group(rs.getLong("group\_id"),

rs.getLong("user\_id"),

rs.getString("title"),

rs.getString("description"))

}

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectGroupsByMember(id: Long) = try {

val sql = "select group\_id, user\_id, title, description" +

" from groups " +

" where group\_id in (select group\_id from members where user\_id = $id)"

jdbcTemplate.queryForList(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

IssueDao.kt

private val logger = KotlinLogging.logger {}

@Repository

class IssueDao(

private val jdbcTemplate: JdbcTemplate

) {

fun selectIssuesByUser(id: Long) = try {

val sql = "select issue\_id, issues.title, issues.description, status, priority, date\_start, date\_finish, groups.title as groupTitle " +

" from public.issues " +

" join public.groups on issues.group\_id = groups.group\_id " +

" join public.members on members.member\_id = issues.member\_id " +

" join public.users on members.user\_id = users.user\_id " +

" where users.user\_id = 1"

jdbcTemplate.queryForList(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectIssuesByGroup(id: Long) = try {

val sql = "select issue\_id, issues.member\_id, title, description, status, priority, date\_start, " +

" date\_finish, users.full\_name, users.email " +

" from issues left join members on issues.member\_id = members.member\_id " +

" left join users on members.user\_id = users.user\_id " +

" where issues.group\_id = $id"

jdbcTemplate.queryForList(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun updateIssueStatus(request: ChangeIssueStatusRequest) = try {

val sql = "update issues set status = '${request.status}'" +

" where issue\_id =${request.issueId}"

val res = jdbcTemplate.update(sql)

if(request.status == "Done") {

updateIssueDateFinish(request.issueId)

}

res

} catch (e: Exception) {

logger.error { e.message }

null

}

fun insertIssue(request: CreateIssueRequest) = try {

val sql = "insert into issues(" +

" member\_id, title, description, status, date\_start, date\_finish, group\_id, priority)" +

" values (null, '${request.title}'," +

" '${request.description}', 'ToDo', now(), null, ${request.groupId}, ${request.priority});"

jdbcTemplate.execute(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun updateIssueUser(request: AssignIssueRequest) = try {

val scriptGetMemberId = "select member\_id from members" +

" where user\_id = ${request.userId} and group\_id = ${request.groupId}"

val memberId = jdbcTemplate.queryForObject(scriptGetMemberId, Long::class.java)

val sql = "update issues set member\_id = $memberId" +

" where issue\_id =${request.issueId}"

jdbcTemplate.update(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectIssuesForReport(id: Long) = try {

val sql = "select title, description, status, priority, date\_start, date\_finish, users.full\_name, users.email" +

" from issues left join members on issues.member\_id = members.member\_id " +

" left join users on members.user\_id = users.user\_id " +

" where issues.group\_id = $id"

jdbcTemplate.queryForList(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

private fun updateIssueDateFinish(issueId: Long) = try {

val sql = "update issues set date\_finish = now()" +

" where issue\_id = $issueId"

jdbcTemplate.update(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

MemberDao.kt

@Repository

class MemberDao(

private val jdbcTemplate: JdbcTemplate

) {

fun insertMember(request: AddMemberRequest) = try {

val sql = "insert into members( " +

" group\_id, user\_id) " +

" VALUES (${request.groupId}, ${request.userId});"

jdbcTemplate.execute(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun deleteMember(groupId: Long, userId: Long) = try {

val scriptGetMemberId = "select member\_id from members" +

" where user\_id = $userId and group\_id = $groupId"

val memberId = jdbcTemplate.queryForObject(scriptGetMemberId, Long::class.java)

val returnIssues = "update issues set member\_id = null, status = 'ToDo' " +

" where member\_id = $memberId"

jdbcTemplate.update(returnIssues)

val deleteMember = "delete from members " +

" where group\_id = $groupId and user\_id = $userId"

jdbcTemplate.execute(deleteMember)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

UserDao.kt

private val logger = KotlinLogging.logger {}

@Repository

class UserDao(

private val jdbcTemplate: JdbcTemplate,

private val encoder: BCryptPasswordEncoder

) {

fun selectUserByGroup(id: Long) = try {

val sql = "select user\_id, username, email, full\_name " +

" from users " +

" where user\_id in " +

" (select user\_id from members where group\_id = $id) "

jdbcTemplate.queryForList(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectUserById(id: Long) = try {

val sql = "select user\_id, username, email, full\_name " +

" from users " +

" where user\_id = $id "

jdbcTemplate.queryForObject(sql) { rs: ResultSet, \_: Int ->

User(rs.getLong("user\_id"),

rs.getString("username"),

rs.getString("email"),

rs.getString("full\_name"))

}

} catch (e: Exception) {

logger.error { e.message }

null

}

fun insertUser(request: RegistrationUserRequest) = try {

val sql = "insert into users(" +

" username, password, email, full\_name)" +

" VALUES ('${request.username}', '${encoder.encode(request.password)}'," +

" '${request.email}', '${request.fullName}');"

jdbcTemplate.execute(sql)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun selectIdByName(name: String) = try {

val sql = "select user\_id " +

" from users " +

" where username like '$name' "

jdbcTemplate.queryForObject(sql, Long::class.java)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

Group.kt

data class Group (

val group\_id: Long = 0,

val user\_id: Long = 0,

val title: String? = null,

val description: String? = null

)

User.kt

data class User(

val userId: Long = 0,

val username: String = "",

val email: String? = null,

val fullName: String? = null

)

AddMemberRequest.kt

data class AddMemberRequest(

var userId: Long = 0,

var groupId: Long = 0

)

AssignIssueRequest.kt

data class AssignIssueRequest(

var userId: Long = 0,

var issueId: Long = 0,

var groupId: Long = 0

)

ChangeIssueStatusRequest.kt

data class ChangeIssueStatusRequest(

var issueId: Long = 0,

var status: String = ""

)

CreateGroupRequest.kt

data class CreateGroupRequest(

var title: String = "",

var description: String? = null,

var userId: Long = 0

)

CreateIssueRequest.kt

data class CreateIssueRequest(

var groupId: Long = 0,

var title: String = "",

var description: String = "",

var priority: Int = 0

)

RegistrationUserRequest.kt

data class RegistrationUserRequest(

val username: String = "",

val password: String = "",

val email: String = "",

val fullName: String = ""

)

GroupService.kt

@Service

class GroupService(

private val groupDao: GroupDao,

private val memberDao: MemberDao,

private val userDao: UserDao,

private val issueDao: IssueDao

) {

fun createGroup(request: CreateGroupRequest) = try {

groupDao.insertGroup(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun deleteGroup(id: Long) = try {

groupDao.deleteFromGroup(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getGroup(id: Long) = try {

groupDao.selectFromGroup(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun addMember(request: AddMemberRequest) = try {

memberDao.insertMember(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun deleteMember(groupId: Long, userId: Long) = try {

memberDao.deleteMember(groupId, userId)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getMembersGroup(id: Long) = try {

userDao.selectUserByGroup(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getGroupsByMember(id: Long) = try {

groupDao.selectGroupsByMember(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun sendReport(id: Long) = try {

issueDao.selectIssuesForReport(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

IssueService.kt

private val logger = KotlinLogging.logger {}

@Service

class IssueService(

private val issueDao: IssueDao

) {

fun getIssuesByUser(userId: Long) = try {

issueDao.selectIssuesByUser(userId)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getIssuesByGroup(groupId: Long) = try {

issueDao.selectIssuesByGroup(groupId)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun changeStatusIssue(request: ChangeIssueStatusRequest) = try {

issueDao.updateIssueStatus(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun createIssue(request: CreateIssueRequest) = try {

issueDao.insertIssue(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun assignIssue(request: AssignIssueRequest) = try {

issueDao.updateIssueUser(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

}

UserService.kt

private val logger = KotlinLogging.logger {}

@Service

class UserService(

private val userDao: UserDao

) {

fun registration(request: RegistrationUserRequest) = try {

userDao.insertUser(request)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getInfo(id: Long) = try {

userDao.selectUserById(id)

} catch (e: Exception) {

logger.error { e.message }

null

}

fun getIdByName(name: String?) = try {

if(name != null) {

userDao.selectIdByName(name)

} else {

null

}

} catch (e: Exception) {

logger.error { e.message }

null

}

}

IssueManagerApplication.kt

@SpringBootApplication

class IssueManagerApplication

fun main(args: Array<String>) {

runApplication<IssueManagerApplication>(\*args)

}

application.yml

spring:

security:

user:

name: jadmin

password: jadmin

datasource:

url: 'jdbc:postgresql://localhost:5432/issue-manager'

username: 'postgres'

password: '0000'

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.6.2</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>ru</groupId>

<artifactId>issue-manager</artifactId>

<version>0.0.1</version>

<name>issue-manager</name>

<description>Issue Manager</description>

<properties>

<java.version>11</java.version>

<kotlin.version>1.5.31</kotlin.version>

</properties>

<dependencies>

<!--Kotlin-->

<dependency>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-reflect</artifactId>

</dependency>

<dependency>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-stdlib-jdk8</artifactId>

</dependency>

<!--Spring Boot-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<!--Test-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--Web-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--Security-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!--Utils-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.thymeleaf.extras</groupId>

<artifactId>thymeleaf-extras-springsecurity5</artifactId>

</dependency>

<!--Database-->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.2.18</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!--Logging-->

<dependency>

<groupId>io.github.microutils</groupId>

<artifactId>kotlin-logging-jvm</artifactId>

<version>2.1.20</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-maven-plugin</artifactId>

<version>1.4.10</version>

<configuration>

<args>

<arg>-Xjsr305=strict</arg>

</args>

<compilerPlugins>

<plugin>spring</plugin>

</compilerPlugins>

</configuration>

<dependencies>

<dependency>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-maven-allopen</artifactId>

<version>${kotlin.version}</version>

</dependency>

</dependencies>

</plugin>

</plugins>

</build>

</project>

Group.ts

export class Group {

group\_id: number = 0

user\_id: number = 0

title: string = ""

description: string = ""

constructor(group\_id: number,

user\_id: number,

title: string,

description: string) {

this.group\_id = group\_id

this.user\_id = user\_id

this.title = title

this.description = description

}

}

Issue.ts

export class Issue {

issue\_id: number = 0

member\_id: number | null = 0

title: string = ""

description: string = ""

status: string = ""

priority: number = 0

date\_start: string = ""

date\_finish: string | null = ""

grouptitle: string = ""

constructor(issue\_id: number,

member\_id: number,

title: string,

description: string,

status: string,

priority: number,

date\_start: string,

date\_finish: string,

grouptitle: string) {

this.issue\_id = issue\_id

this.member\_id = member\_id

this.title = title

this.description = description

this.status = status

this.priority = priority

this.date\_start = date\_start

this.date\_finish = date\_finish

this.grouptitle = grouptitle

}

}

IssueGroup.ts

export class IssueGroup {

issue\_id: number = 0

member\_id: number | null = 0

title: string = ""

description: string = ""

status: string = ""

priority: number = 0

date\_start: string = ""

date\_finish: string | null = ""

full\_name: string = ""

email: string = ""

constructor(issue\_id: number,

member\_id: number,

title: string,

description: string,

status: string,

priority: number,

date\_start: string,

date\_finish: string,

full\_name: string,

email: string) {

this.issue\_id = issue\_id

this.member\_id = member\_id

this.title = title

this.description = description

this.status = status

this.priority = priority

this.date\_start = date\_start

this.date\_finish = date\_finish

this.full\_name = full\_name

this.email = email

}

}

Report.ts

export class Report{

title: string = ""

description: string = ""

status: string = ""

priority: number = 0

date\_start: string = ""

date\_finish: string | null = ""

full\_name: string = ""

email: string = ""

constructor(title: string,

description: string,

status: string,

priority: number,

date\_start: string,

date\_finish: string,

full\_name: string,

email: string) {

this.title = title

this.description = description

this.status = status

this.priority = priority

this.date\_start = date\_start

this.date\_finish = date\_finish

this.full\_name = full\_name

this.email = email

}

}

User.ts

export class User {

user\_id: number = 0

username: string = ""

email: string = ""

full\_name: string = ""

constructor(user\_id: number,

username: string,

email: string,

full\_name: string) {

this.user\_id = user\_id

this.username = username

this.email = email

this.full\_name = full\_name

}

}

admin.component.css

table {

margin:auto;

width:120%;

height:100%;

font-size: 16px;

background: white;

max-width: 70%;

border-collapse: collapse;

text-align: left;

}

th {

font-weight: normal;

color: #039;

border-bottom: 2px solid #6678b1;

padding: 10px 8px;

}

td {

color: #669;

padding: 9px 8px;

transition: .3s linear;

}

tr:hover td {

color: #6699ff;

}

.h-style {

text-align-last: center;

color: #669;

}

.btn {

display: inline-block;

font-weight: normal;

line-height: 1.25;

text-align: center;

white-space: nowrap;

vertical-align: middle;

cursor: pointer;

-webkit-user-select: none;

-moz-user-select: none;

-ms-user-select: none;

user-select: none;

border: 1px solid transparent;

padding: 0.5rem 1rem;

font-size: 1rem;

border-radius: 0.25rem;

}

.btn-primary {

color: #fff;

background-color: #0275d8;

border-color: #0275d8;

}

admin.component.html

<title>Панель администратора групп</title>

<div class="p-container">

<app-navigation></app-navigation>

<div>

<h2 class="h-style">Администрируемые группы</h2>

<table>

<tr>

<th>ID группы</th>

<th>Заголовок</th>

<th>Описание</th>

<th></th>

</tr>

<tr \*ngFor="let group of allGroups">

<td>{{group.group\_id}}</td>

<td>{{group.title}}</td>

<td>{{group.description}}</td>

<td><button class="btn btn-primary"(click)="getManage(group)">Администрирование</button></td>

</tr>

</table>

</div>

</div>

admin.component.ts

export class AdminComponent implements OnInit {

allGroups: Group[] = []

constructor(private restapi: RestapiService,

private router: Router) {

this.getGroups()

}

ngOnInit(): void {

}

getGroups() {

this.restapi.get('/group/getGroupsByMember/' + localStorage.getItem('id'))

.subscribe((data: any) => {

this.allGroups = []

let res = JSON.parse(data)

res.forEach((group: Group) => {

if(group.user\_id.toString() == localStorage.getItem('id')) {

this.allGroups.push(group)

}

})

}

)

}

getManage(group: Group) {

this.router.navigate(["manage"], {queryParams: {group: group.group\_id}});

}

}

group.component.css

.card {

border-bottom: 2px solid #6678b1;

border-top: 2px solid #6678b1;

height:100%;

font-size: 16px;

background: white;

max-width: 30%;

text-align: left;

margin: 45px;

margin-inside: 10px;

color: #669;

}

table {

margin:auto;

width:120%;

height:100%;

font-size: 16px;

background: white;

max-width: 70%;

border-collapse: collapse;

text-align: left;

}

th {

font-weight: normal;

color: #039;

border-bottom: 2px solid #6678b1;

padding: 10px 8px;

}

td {

color: #669;

padding: 9px 8px;

transition: .3s linear;

}

tr:hover td {

color: #6699ff;

}

.h-style {

text-align-last: center;

color: #669;

}

.my-input{

margin: 5px;

}

.btn {

display: inline-block;

font-weight: normal;

line-height: 1.25;

text-align: center;

white-space: nowrap;

vertical-align: middle;

cursor: pointer;

-webkit-user-select: none;

-moz-user-select: none;

-ms-user-select: none;

user-select: none;

border: 1px solid transparent;

padding: 0.5rem 1rem;

font-size: 1rem;

border-radius: 0.25rem;

}

.btn-success {

color: #fff;

background-color: #5cb85c;

border-color: #5cb85c;

margin-bottom: 10px;

margin-top: 10px;

}

.form-control {

display: block;

width: 100%;

padding: 0.5rem 0.75rem;

font-size: 1rem;

line-height: 1.25;

color: #55595c;

background-color: #fff;

background-image: none;

-webkit-background-clip: padding-box;

background-clip: padding-box;

border: 1px solid rgba(0, 0, 0, 0.15);

border-radius: 0.25rem;

}

group.component.html

<title>Панель групп пользователя</title>

<div class="p-container">

<app-navigation></app-navigation>

<div class="card">

<h4 class="h-style">Создать группу</h4>

<form>

<label>Введите название группы: </label>

<input class="my-input" type="text" name="titleGroup" [(ngModel)] = "createGroupRequest.title">

<br>

<label>Введите описание группы: </label>

<textarea class="my-input" rows="10" cols="45" name="descriptionGroup" [(ngModel)] = "createGroupRequest.description"></textarea>

<br>

<button name="createGroup" class="btn btn-success" (click)="createGroup()">Создать</button>

</form>

</div>

<div>

<h2 class="h-style">Мои группы</h2>

<table>

<tr>

<th>ID группы</th>

<th>Заголовок</th>

<th>Описание</th>

<th></th>

</tr>

<tr \*ngFor="let group of allGroups">

<td>{{group.group\_id}}</td>

<td>{{group.title}}</td>

<td>{{group.description}}</td>

<td>{{isAdmin(group)}}</td>

</tr>

</table>

</div>

</div>

group.component.ts

export class GroupComponent implements OnInit {

allGroups: Group[] = []

createGroupRequest = {

title: null,

description: null,

userId: null

}

constructor(private restapi: RestapiService) {

this.getGroups()

// @ts-ignore

this.createGroupRequest.userId = localStorage.getItem('id')

}

ngOnInit(): void {

}

getGroups() {

this.restapi.get('/group/getGroupsByMember/' + localStorage.getItem('id'))

.subscribe((data: any) => {

this.allGroups = []

let res = JSON.parse(data)

res.forEach((group: Group) => this.allGroups.push(group))

}

)

}

isAdmin(group: Group) {

if (group.user\_id.toString() == localStorage.getItem('id')) {

return "Вы администратор"

} else {

return ""

}

}

createGroup() {

if (this.createGroupRequest.title !== null) {

this.restapi.post('/group/createGroup', this.createGroupRequest)

.subscribe((data: any) => {

this.getGroups()

})

}

}

}

home.component.css

.card {

border-bottom: 2px solid #6678b1;

border-top: 2px solid #6678b1;

height:100%;

font-size: 16px;

background: white;

max-width: 15%;

text-align: left;

margin: 45px;

margin-inside: 10px;

color: #669;

}

home.component.html

<title>Главная страница</title>

<app-navigation></app-navigation>

<div class="container card">

<p>ID - {{info.userId}}</p>

<p>Логин - {{info.username}}</p>

<p>Email - {{info.email}}</p>

<p>Полное имя - {{info.fullName}}</p>

</div>

home.component.ts

export class HomeComponent implements OnInit {

info: { userId: number, username: string, email: string | null, fullName: string | null } =

{

userId: 0,

username: '',

email: null,

fullName: null

}

constructor(private service: RestapiService) {

this.getInfo()

}

ngOnInit() {

}

getInfo() {

let resp = this.service.get("/user/getInfo/" + localStorage.getItem('id'))

resp.subscribe((data: any) => {

let res = JSON.parse(data)

this.info.userId = res.userId

this.info.username = res.username

this.info.email = res.email

this.info.fullName = res.fullName

})

}

}

issue.component.css

.filter {

border-bottom: 2px solid #6678b1;

border-top: 2px solid #6678b1;

height:100%;

font-size: 16px;

background: white;

max-width: 100%;

text-align: left;

margin: 45px;

margin-inside: 10px;

color: #669;

}

table {

margin:auto;

width:150%;

height:100%;

font-size: 16px;

background: white;

max-width: 90%;

border-collapse: collapse;

text-align: left;

}

th {

font-weight: normal;

color: #039;

border-bottom: 2px solid #6678b1;

padding: 10px 8px;

}

td {

color: #669;

padding: 9px 8px;

transition: .3s linear;

}

tr:hover td {

color: #6699ff;

}

.h-style {

text-align-last: center;

color: #669;

}

.my-input{

margin: 15px;

}

.btn {

display: inline-block;

font-weight: normal;

line-height: 1.25;

text-align: center;

white-space: nowrap;

vertical-align: middle;

cursor: pointer;

-webkit-user-select: none;

-moz-user-select: none;

-ms-user-select: none;

user-select: none;

border: 1px solid transparent;

padding: 0.5rem 1rem;

font-size: 1rem;

border-radius: 0.25rem;

margin-right: 20px;

}

.btn-primary {

color: #fff;

background-color: #0275d8;

border-color: #0275d8;

margin-bottom: 10px;

margin-top: 10px;

}

.btn-primary:disabled {

cursor: default;

color: #fff;

background-color: #014079;

border-color: #014281;

}

issue.component.html

<title>Панель задач пользователя</title>

<div class="p-container">

<app-navigation></app-navigation>

<div>

<div class="filter">

<h4 class="h-style">Фильтр задач</h4>

<form>

<label>ID: </label>

<input class="my-input" type="text" name="issue\_id" [(ngModel)]="filterIssue.issue\_id">

<label>Заголовок: </label>

<input class="my-input" type="text" name="title" [(ngModel)]="filterIssue.title">

<label>Описание: </label>

<input class="my-input" type="text" name="description" [(ngModel)]="filterIssue.description">

<label>Группа: </label>

<input class="my-input" type="text" name="grouptitle" [(ngModel)]="filterIssue.grouptitle">

<label>Статус: </label>

<select name="statuses" [(ngModel)]="filterIssue.status">

<option \*ngFor="let option of options" [ngValue]="option">{{option}}</option>

</select>

<br>

<button name="filter" class="btn btn-primary" (click)="filter()">Найти</button>

<button name="filter" class="btn btn-primary" (click)="hideDone(true)">Скрыть выполненные</button>

<button name="filter" class="btn btn-primary" (click)="hideDone(false)">Показать выполненные</button>

</form>

</div>

</div>

<div>

<h2 class="h-style">Мои задачи</h2>

<table>

<tr>

<th>ID задачи</th>

<th>Заголовок</th>

<th>Описание</th>

<th>Статус</th>

<th>Приоритет</th>

<th>Дата начала</th>

<th>Дата конца</th>

<th>Группа</th>

<th></th>

</tr>

<tr \*ngFor="let issue of visibleIssues">

<td>{{issue.issue\_id}}</td>

<td>{{issue.title}}</td>

<td>{{issue.description}}</td>

<td>{{issue.status}}</td>

<td align="center">{{getStrByPriority(issue.priority)}}</td>

<td>{{issue.date\_start}}</td>

<td>{{issue.date\_finish}}</td>

<td>{{issue.grouptitle}}</td>

<td><button class = "btn btn-primary" (click)="changeStatus(issue)"

[disabled]="(issue.status == 'Done')">Изменить статус</button></td>

</tr>

</table>

</div>

</div>

issue.component.ts

export class IssueComponent implements OnInit {

allIssues: Issue[] = []

visibleIssues: Issue[] = []

filterIssue = {

issue\_id: "",

title: "",

description: "",

status: "",

grouptitle: "",

}

options = ['', 'ToDo', 'InProgress', 'InTest', 'Done']

constructor(private restapi: RestapiService) {

this.getIssues()

}

ngOnInit(): void {

}

getIssues() {

this.restapi.get('/issue/get/byUser/' + localStorage.getItem('id'))

.subscribe((data: any) => {

this.allIssues = []

let res = JSON.parse(data)

res.forEach((issue: Issue) => this.allIssues.push(issue))

this.visibleIssues = this.allIssues

}

)

}

changeStatus(issue: Issue) {

let newStatus: string = issue.status

switch (issue.status) {

case 'ToDo':

newStatus = 'InProgress';

break

case 'InProgress':

newStatus = 'InTest';

break

case 'InTest':

newStatus = 'Done';

break

}

console.log(newStatus)

console.log(issue.status)

this.restapi.post('/issue/changeStatus', {issueId: issue.issue\_id, status: newStatus})

.subscribe((data: any) => {

this.getIssues()

})

}

filter() {

this.visibleIssues = []

this.allIssues.forEach((issue: Issue) => {

if (this.findSubstr(issue.title, this.filterIssue.title) &&

this.findSubstr(issue.description, this.filterIssue.description) &&

this.findSubstr(issue.status, this.filterIssue.status) &&

this.findSubstr(issue.grouptitle, this.filterIssue.grouptitle) &&

this.findSubstr(String(issue.issue\_id), String(this.filterIssue.issue\_id))) {

this.visibleIssues.push(issue)

}

})

}

private findSubstr(str1: string, str2: string) {

return !str1.toLowerCase().indexOf(str2.toLowerCase())

}

hideDone(flag: boolean) {

this.visibleIssues = []

if(flag) {

this.allIssues.forEach((issue: Issue) => {

if(issue.status !== 'Done') {

this.visibleIssues.push(issue)

}

})

} else {

this.visibleIssues = this.allIssues

}

}

getStrByPriority(priority: number) {

switch (priority) {

case 1: return "Первый"

case 2: return "Второй"

case 3: return "Третий"

case 4: return "Четвертый"

default: return "Пятый"

}

}

}

login.component.html

<link href="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="stylesheet" id="bootstrap-css">

<script src="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"></script>

<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<!------ Include the above in your HEAD tag ---------->

<title>Панель авторизации</title>

<div class="container-login">

<div class="row justify-content-center">

<div class="col-md-3">

<div class="card">

<div class="card-header">Авторизация</div>

<div class="card-body">

<form class="form-horizontal">

<div class="form-group">

<label for="username" class="cols-sm-2 control-label">Логин</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="text" class="form-control" name="username" id="username" placeholder="Введите свой логин"

[(ngModel)]="username"/>

</div>

</div>

</div>

<div class="form-group">

<label for="password" class="cols-sm-2 control-label">Пароль</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="password" class="form-control" name="password" id="password"

placeholder="Введите свой пароль"

[(ngModel)]="password"/>

</div>

</div>

</div>

<div class="form-group ">

<button type="button" class="btn btn-primary btn-lg btn-block login-button" (click)="doLogin()">Войти

</button>

</div>

<div class="login-register">

<button class="btn btn-secondary" (click)="goToRegistration()">Регистрация</button>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

login.component.ts

export class LoginComponent implements OnInit {

id: Object = 0

username: string = "";

password: string = "";

constructor(private service: RestapiService,

private router: Router) {

}

ngOnInit() {

}

doLogin() {

let resp = this.service.login(this.username, this.password);

resp.subscribe(res => {

localStorage.setItem('id', String(res));

this.router.navigate(["/home"])

}, error => {

alert("Неверный логин или пароль!")

});

}

goToRegistration() {

this.router.navigate(["/registration"])

}

}

manage-group.component.css

.card {

border-bottom: 2px solid #6678b1;

border-top: 2px solid #6678b1;

height:100%;

font-size: 16px;

background: white;

max-width: 30%;

text-align: left;

margin: 45px;

margin-inside: 10px;

color: #669;

}

.filter {

border-bottom: 2px solid #6678b1;

border-top: 2px solid #6678b1;

height:100%;

font-size: 16px;

background: white;

max-width: 100%;

text-align: left;

margin: 45px;

margin-inside: 10px;

color: #669;

}

table {

margin:auto;

width:120%;

height:100%;

font-size: 16px;

background: white;

max-width: 95%;

border-collapse: collapse;

text-align: left;

}

.table-member {

max-width: 70%;

}

th {

font-weight: normal;

color: #039;

border-bottom: 2px solid #6678b1;

padding: 10px 8px;

}

td {

color: #669;

padding: 9px 8px;

transition: .3s linear;

}

tr:hover td {

color: #6699ff;

}

.my-btn {

margin-top: 10px;

margin-bottom: 10px;

}

.h-style {

text-align-last: center;

color: #669;

}

.my-input{

margin: 5px;

}

.btn {

display: inline-block;

font-weight: normal;

line-height: 1.25;

text-align: center;

white-space: nowrap;

vertical-align: middle;

cursor: pointer;

-webkit-user-select: none;

-moz-user-select: none;

-ms-user-select: none;

user-select: none;

border: 1px solid transparent;

padding: 0.5rem 1rem;

font-size: 1rem;

border-radius: 0.25rem;

}

.btn-danger {

color: #fff;

background-color: #d9534f;

border-color: #d9534f;

}

.btn-success {

color: #fff;

background-color: #5cb85c;

border-color: #5cb85c;

margin-bottom: 10px;

margin-right: 10px;

margin-top: 10px;

}

.main-border {

border-top: 2px solid #6678b1;

max-width: 100%;

}

.btn-primary {

color: #fff;

background-color: #0275d8;

border-color: #0275d8;

margin-bottom: 10px;

margin-top: 10px;

margin-right: 20px;

}

select{

margin-bottom: 15px;

}

manage-group.component.html

<title>Управление группой</title>

<app-navigation></app-navigation>

<div class="card">

<p>ID - {{group?.group\_id}}</p>

<p>Назание - {{group?.title}}</p>

<p>Описание - {{group?.description}}</p>

<button class="btn btn-success my-btn" name="deleteGroup" (click)="getReport()">Скачать отчет</button>

<button class="btn btn-danger my-btn" name="deleteGroup" (click)="deleteGroup()">Удалить группу</button>

</div>

<div class="card">

<h4 class="h-style">Добавить пользователя в группу</h4>

<form>

<label>Введите ID нового участника: </label>

<input class="my-input" type="text" name="memberId" [(ngModel)]="newMemberRequest.userId">

<br>

<button name="member" class="btn btn-success" (click)="addMember()">Добавить</button>

</form>

</div>

<div>

<h2 class="h-style">Участники группы</h2>

<table class="table-member">

<tr>

<th>ID</th>

<th>Логин</th>

<th>Email</th>

<th>Полное имя</th>

<th></th>

</tr>

<tr \*ngFor="let user of allMembers">

<td>{{user.user\_id}}</td>

<td>{{user.username}}</td>

<td>{{user.email}}</td>

<td>{{user.full\_name}}</td>

<td>

<button class="btn btn-danger" name="deleteMember" (click)="deleteMember(user.user\_id)">Исключить</button>

</td>

</tr>

</table>

</div>

<div class="main-border"></div>

<div>

<div class="card">

<h4 class="h-style">Создать задачу</h4>

<form>

<label>Введите заголовок задачи: </label>

<input class="my-input" type="text" name="title" [(ngModel)]="createIssueRequest.title">

<br>

<label>Выберите приоритет: </label>

<select name="priority" [(ngModel)]="createIssueRequest.priority">

<option \*ngFor="let priority of priorities" [ngValue]="priority.value">{{priority.title}}</option>

</select>

<br>

<label>Введите описание задачи: </label>

<textarea class="my-input" rows="10" cols="45" name="description" [(ngModel)] = "createIssueRequest.description"></textarea>

<br>

<button name="member" class="btn btn-success" (click)="createIssue()">Создать</button>

</form>

</div>

</div>

<div class="card">

<h4 class="h-style">Назначить задачу</h4>

<form>

<label>Выберите задачу: </label>

<select name="issueId" [(ngModel)]="assignIssueRequest.issueId">

<option \*ngFor="let option of optionsIssues" [ngValue]="option.id">{{option.title}}</option>

</select>

<br>

<label>Выберите участника: </label>

<select name="memberId" [(ngModel)]="assignIssueRequest.userId">

<option \*ngFor="let option of optionsMembers" [ngValue]="option.id">{{option.full\_name}}</option>

</select>

<br>

<button name="member" class="btn btn-success" (click)="assignIssue()">Назначить</button>

</form>

</div>

<div class="filter">

<h4 class="h-style">Фильтр задач</h4>

<form>

<label>ID: </label>

<input class="my-input" type="text" name="issue\_id" [(ngModel)]="filterIssue.issue\_id">

<label>Заголовок: </label>

<input class="my-input" type="text" name="title" [(ngModel)]="filterIssue.title">

<label>Описание: </label>

<input class="my-input" type="text" name="description" [(ngModel)]="filterIssue.description">

<label>Участник: </label>

<input class="my-input" type="text" name="fullname" [(ngModel)]="filterIssue.full\_name">

<label>Статус: </label>

<select name="statuses" [(ngModel)]="filterIssue.status">

<option \*ngFor="let option of options" [ngValue]="option">{{option}}</option>

</select>

<br>

<button name="filter" class="btn btn-primary" (click)="filter()">Найти</button>

<button name="filter" class="btn btn-primary" (click)="hideDone(true)">Скрыть выполненные</button>

<button name="filter" class="btn btn-primary" (click)="hideDone(false)">Показать выполненные</button>

</form>

</div>

<div>

<h2 class="h-style">Задачи группы</h2>

<table>

<tr>

<th>ID задачи</th>

<th>Заголовок</th>

<th>Описание</th>

<th>Статус</th>

<th>Приоритет</th>

<th>Дата начала</th>

<th>Дата конца</th>

<th>Имя</th>

<th>Почта</th>

</tr>

<tr \*ngFor="let issue of visibleIssues">

<td>{{issue.issue\_id}}</td>

<td>{{issue.title}}</td>

<td>{{issue.description}}</td>

<td>{{issue.status}}</td>

<td align="center">{{getStrByPriority(issue.priority)}}</td>

<td>{{issue.date\_start}}</td>

<td>{{issue.date\_finish}}</td>

<td>{{checkMember(issue.full\_name)}}</td>

<td>{{issue.email}}</td>

</tr>

</table>

</div>

<footer style="margin-top: 40px"></footer>

manage-group.component.ts

export class ManageGroupComponent implements OnInit {

id: number = 0

group: Group | null = null

allMembers: User[] = []

newMemberRequest = {

userId: null,

groupId: null

}

createIssueRequest = {

groupId: null,

title: null,

description: null,

priority: null

}

assignIssueRequest = {

userId: null,

issueId: null,

groupId: null

}

filterIssue = {

issue\_id: "",

title: "",

description: "",

status: "",

full\_name: ""

}

optionsIssues: { title: string, id: number }[] = []

optionsMembers: { full\_name: string, id: number }[] = []

options = ['', 'ToDo', 'InProgress', 'InTest', 'Done']

priorities = [

{title: 'Первый', value: 1},

{title: 'Второй', value: 2},

{title: 'Третий', value: 3},

{title: 'Четвертый', value: 4},

{title: 'Пятый', value: 5},

]

allIssues: IssueGroup[] = []

visibleIssues: IssueGroup[] = []

report: Report[] = []

constructor(private restapi: RestapiService,

private excel: ExcelService,

private router: Router,

private route: ActivatedRoute) {

this.route.queryParams.subscribe(params => {

this.id = params['group'];

})

this.restapi.get('/group/getGroup/' + this.id).subscribe(

(data: any) => this.group = JSON.parse(data)

)

this.getMembers()

this.getIssues()

// @ts-ignore

this.newMemberRequest.groupId = this.id

// @ts-ignore

this.createIssueRequest.groupId = this.id

// @ts-ignore

this.assignIssueRequest.groupId = this.id

}

ngOnInit(): void {

}

getMembers() {

this.restapi.get('/group/getMembersGroup/' + this.id)

.subscribe((data: any) => {

this.allMembers = []

this.optionsMembers = []

let res = JSON.parse(data)

res.forEach((user: User) => {

this.allMembers.push(user)

this.optionsMembers.push({full\_name: user.full\_name, id: user.user\_id})

})

}

)

}

deleteMember(user\_id: number) {

if (user\_id == this.group?.user\_id) {

alert('Нельзя удалить себя из группы!')

} else {

this.restapi.get('/group/deleteMember/' + this.id + '/' + user\_id)

.subscribe((data: any) => {

this.getMembers()

this.getIssues()

})

}

}

deleteGroup() {

this.restapi.get('/group/deleteGroup/' + this.id)

.subscribe((data: any) => {

this.router.navigate(["admin"]);

})

}

addMember() {

if (this.newMemberRequest.userId !== null && this.newMemberRequest.userId != localStorage.getItem('id')) {

this.restapi.post('/group/addMember', this.newMemberRequest)

.subscribe((data: any) => {

this.getMembers()

this.newMemberRequest.userId = null

})

}

}

getIssues() {

this.restapi.get('/issue/get/byGroup/' + this.id)

.subscribe((data: any) => {

this.allIssues = []

this.optionsIssues = []

let res = JSON.parse(data)

res.forEach((issue: IssueGroup) => {

this.allIssues.push(issue)

if (issue.member\_id === null) {

this.optionsIssues.push({title: issue.title, id: issue.issue\_id})

}

})

this.visibleIssues = this.allIssues

}

)

}

createIssue() {

if (this.createIssueRequest.title !== null) {

this.restapi.post('/issue/create', this.createIssueRequest)

.subscribe((data: any) => {

this.getIssues()

this.createIssueRequest.title = null

this.createIssueRequest.description = null

this.createIssueRequest.priority = null

})

}

}

checkMember(name: string | null) {

if (name === null) {

return "Не назначено"

} else {

return name

}

}

assignIssue() {

if (this.assignIssueRequest.userId !== null && this.assignIssueRequest.issueId !== null) {

this.restapi.post('/issue/assignIssue', this.assignIssueRequest)

.subscribe((data: any) => {

this.getIssues()

})

}

}

filter() {

this.visibleIssues = []

this.allIssues.forEach((issue: IssueGroup) => {

if (this.findSubstr(issue.title, this.filterIssue.title) &&

this.findSubstr(issue.description, this.filterIssue.description) &&

this.findSubstr(issue.status, this.filterIssue.status) &&

this.findSubstr(issue.full\_name, this.filterIssue.full\_name) &&

this.findSubstr(String(issue.issue\_id), String(this.filterIssue.issue\_id))) {

this.visibleIssues.push(issue)

}

})

}

hideDone(flag: boolean) {

this.visibleIssues = []

if (flag) {

this.allIssues.forEach((issue: IssueGroup) => {

if (issue.status !== 'Done') {

this.visibleIssues.push(issue)

}

})

} else {

this.visibleIssues = this.allIssues

}

}

getReport() {

this.restapi.get('/group/getReport/' + this.id)

.subscribe((data: any) => {

this.report = []

let res = JSON.parse(data)

res.forEach((report: Report) => {

this.report.push(report)

})

this.excel.exportAsExcelFile(this.report, 'report')

})

}

getStrByPriority(priority: number) {

switch (priority) {

case 1: return "Первый"

case 2: return "Второй"

case 3: return "Третий"

case 4: return "Четвертый"

default: return "Пятый"

}

}

private findSubstr(str1: string, str2: string) {

return !str1.toLowerCase().indexOf(str2.toLowerCase())

}

}

navigation.component.css

.menu li {

display: inline-block;

}

.menu a {

text-decoration: none;

font-family: sans-serif;

color: #5757a0;

display: inline-block;

padding: 10px 20px;

background-color: lavender;

border-bottom: 5px solid #5757a0;

}

.menu a:hover {

color: #800000;

}

navigation.component.html

<ul class="menu">

<li>

<a href="#" [routerLink]="'/home'">Главная</a>

</li>

<li>

<a href="#" [routerLink]="'/issue'">Мои задачи</a>

</li>

<li>

<a href="#" [routerLink]="'/group'">Мои группы</a>

</li>

<li>

<a href="#" [routerLink]="'/admin'">Администрирование</a>

</li>

<li>

<a href="#" [routerLink]="'/login'">Выход</a>

</li>

</ul>

navigation.component.ts

export class NavigationComponent implements OnInit {

constructor(private router: Router) { }

ngOnInit(): void {

}

}

registration.component.html

<link href="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="stylesheet" id="bootstrap-css">

<script src="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"></script>

<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<!------ Include the above in your HEAD tag ---------->

<title>Панель регистрации</title>

<div class="container">

<div class="row justify-content-center">

<div class="col-md-6">

<div class="card">

<div class="card-header">Регистрация</div>

<div class="card-body">

<form class="form-horizontal">

<div class="form-group">

<label for="name" class="cols-sm-2 control-label">Полное имя</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="text" class="form-control" name="name" id="name" placeholder="Введите свое полное имя"

[(ngModel)] = "registrationRequest.fullName"/>

</div>

</div>

</div>

<div class="form-group">

<label for="email" class="cols-sm-2 control-label">Электронная почта</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="text" class="form-control" name="email" id="email" placeholder="Введите свою электронную почту"

[(ngModel)] = "registrationRequest.email"/>

</div>

</div>

</div>

<div class="form-group">

<label for="username" class="cols-sm-2 control-label">Логин</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="text" class="form-control" name="username" id="username" placeholder="Введите свой логин"

[(ngModel)] = "registrationRequest.username"/>

</div>

</div>

</div>

<div class="form-group">

<label for="password" class="cols-sm-2 control-label">Пароль</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="password" class="form-control" name="password" id="password" placeholder="Введите свой пароль"

[(ngModel)] = "registrationRequest.password"/>

</div>

</div>

</div>

<div class="form-group">

<label for="confirm" class="cols-sm-2 control-label">Подтвердите пароль</label>

<div class="cols-sm-10">

<div class="input-group">

<input type="password" class="form-control" name="confirm" id="confirm" placeholder="Подтвердите пароль"

[(ngModel)] = "confirmPassword"/>

</div>

</div>

</div>

<div class="form-group">

<button type="button" class="btn btn-primary btn-lg btn-block login-button" (click)="registration()">Зарегистрироваться</button>

</div>

<div class="login-register">

<button class="btn btn-secondary" (click)="goToLogin()">Авторизация</button>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

registration.component.ts

export class RegistrationComponent implements OnInit {

registrationRequest = {

username: null,

password: null,

email: null,

fullName: null

}

confirmPassword: string = ""

constructor(private restapi: RestapiService,

private router: Router) {

}

ngOnInit(): void {

}

goToLogin() {

this.router.navigate(["/login"])

}

registration() {

if(this.registrationRequest.password === this.confirmPassword) {

this.restapi.post('/user/registration', this.registrationRequest)

.subscribe((data: any) => {

alert("Успешная регистрация!")

this.router.navigate(["/login"])

})

}

}

}

app.component.html

<router-outlet></router-outlet>

app.component.ts

export class AppComponent {

}

app.module.ts

@NgModule({

declarations: [

AppComponent,

LoginComponent,

HomeComponent,

NavigationComponent,

IssueComponent,

GroupComponent,

AdminComponent,

ManageGroupComponent,

RegistrationComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

HttpClientModule

],

providers: [RestapiService],

bootstrap: [AppComponent]

})

export class AppModule { }

app-routing.module.ts

const routes: Routes = [

{path:"",redirectTo:"login",pathMatch:"full"},

{path:"login",component:LoginComponent},

{path:"home",component:HomeComponent},

{path:"issue",component:IssueComponent},

{path:"group",component:GroupComponent},

{path:"admin",component:AdminComponent},

{path:"manage",component:ManageGroupComponent},

{path:"registration", component: RegistrationComponent}

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

excel.service.ts

const EXCEL\_TYPE = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet;charset=UTF-8';

const EXCEL\_EXTENSION = '.xlsx';

@Injectable({

providedIn: 'root'

})

export class ExcelService {

constructor() { }

public exportAsExcelFile(json: any[], excelFileName: string): void {

const myworksheet: XLSX.WorkSheet = XLSX.utils.json\_to\_sheet(json);

const myworkbook: XLSX.WorkBook = { Sheets: { 'data': myworksheet }, SheetNames: ['data'] };

const excelBuffer: any = XLSX.write(myworkbook, { bookType: 'xlsx', type: 'array' });

this.saveAsExcelFile(excelBuffer, excelFileName);

}

private saveAsExcelFile(buffer: any, fileName: string): void {

const data: Blob = new Blob([buffer], {

type: EXCEL\_TYPE

});

FileSaver.saveAs(data, fileName + '\_exported'+ EXCEL\_EXTENSION);

}

}

restapi.service.ts

export class RestapiService {

constructor(private http: HttpClient) {

}

login(username: string, password: string) {

localStorage.setItem('username', username);

localStorage.setItem('password', password);

const headers = new HttpHeaders({Authorization: 'Basic ' + btoa(username + ':' + password)});

return this.http.get("http://localhost:8080/user/auth", {headers, responseType: 'text' as 'json'})

}

get(endpoint: string) {

const headers = new HttpHeaders({

Authorization: 'Basic ' + btoa(

localStorage.getItem('username') + ':' + localStorage.getItem('password'))

}

);

return this.http.get("http://localhost:8080" + endpoint, {headers, responseType: 'text' as 'json'})

}

post(endpoint: string, request: any) {

const headers = new HttpHeaders({

Authorization: 'Basic ' + btoa(

localStorage.getItem('username') + ':' + localStorage.getItem('password'))

}

);

return this.http.post("http://localhost:8080" + endpoint, request,{headers, responseType: 'text' as 'json'})

}

}

index.html

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="src/favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html>

# Приложение В

(обязательное)

Антиплагиат к бакалаврской работе