

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

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

ДИПЛОМНЫЙ ПРОЕКТ

на тему

КЛИЕНТ-СЕРВЕРНАЯ СИСТЕМА ДЛЯ ПЛАНИРОВАНИЯ,
РАСПРЕДЕЛЕНИЯ И УПРАВЛЕНИЯ ПРОЕКТАМИ

Студент 19.06.23 Томлев Антон Валерьевич
(подпись, дата) (ФИО полностью)

Руководитель Зинкина С.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Филатов А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент 20.06.23 Петерский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 2.06.2023 г. № 14)

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

Работа защищена с отметкой хорошо
(протокол заседания ГЭК от 23.06.2023 г. № 8)

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

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2023 г.

**ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ**

1. Студент Паняев Антон Валерьевич гр. 18ВОЭ1 факультета вычислительной техники.
2. Тема работы «Клиент-серверная система для планирования, распределения и управления проектами».

Тема утверждена приказом ПГУ 458/0 от 26.04.2023 г.

3. Руководитель работы д.т.н., профессор каф. ВТ ПГУ Зинкин С.А.

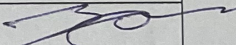
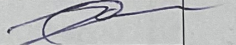
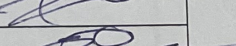
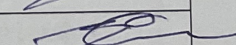
4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – клиент-серверная система, предназначенный для планирования, распределения и управления проектами. Приложение должно обладать графическим и интуитивно понятным интерфейсом, а также безопасной аутентификация и авторизацией.

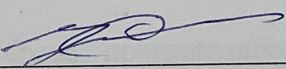
5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области: изучить аналоги клиент-серверных систем со схожей функциональностью, выделить достоинства и недостатки. Определить архитектуру приложения и средства разработки. Выбрать язык программирования и доступные библиотеки

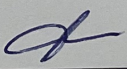
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	10	26.04.2023 - 28.04.2023	
2	Анализ требований	10	29.04.2023 - 30.04.2023	
3	Выбор инструментов для разработки программы	10	31.04.2023 - 03.05.2023	
4	Реализация графического интерфейса	10	04.05.2023 - 08.05.2023	
5	Реализация программы	20	09.05.2023 - 13.05.2023	
6	Сборка приложения в контейнере	10	16.05.2023 - 23.05.2023	
7	Оформление выпускной работы	30	24.05.2023 - 02.06.2023	

Дата выдачи задания «26» 04. 2023 г.

Руководитель дипломного проекта  Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  Паняев А.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
22.06.2023

Реферат

Работа содержит 128 листов, 16 рисунков, 9 таблиц, 3 приложений
МИКРОСЕРВИС, БАЗА ДАННЫХ, СЕРВИС, КОНТРОЛЛЕР, *REST API*, *JWT*,
JSON, *UI*, *Postgresql*, *MAVEN*, *JAVA*, *UML*, *CRUD*

В рамках данной дипломной работы была создана клиент-серверная система для управления проектами.

Цель работы – создать клиент-серверную систему, управляющую проектами с удобным пользовательским интерфейсом.

Объектом разработки является клиент-серверная система для для управления проектами.

В результате проведенной работы реализован проект, удовлетворяющий поставленной задаче, а именно: удобный пользовательский интерфейс, управление проектами, безопасная авторизация и аутентификация, открытый *API*.

Разработка проводилась в среде программирования *IntelliJ IDEA Ultimate Edition*. Использованные языки программирования: *Java*.

					ПГУ 1.090501.09.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дат	Клиент-серверная система для планирования, распределения и управления проектами Пояснительная записка	Лит.	Лист	Листов
Разраб.		Паняев А.В.	КЕ				2	128
Проверил		Зинкин С.А.						
Н. Контр.		Бычков А.С.						
Утверд.		Митрохин М.А.						
						ФВТ, каф. ВТ Гр. 18ВОЭ1		

Содержание

Реферат	4
Перечень сокращений и обозначений.....	5
Введение	6
1 Клиент-серверная система для планирования, распределения и управления проектами	8
1.1 Анализ предметной области	8
1.2 Постановка задачи	12
1.3 Архитектура приложения	13
1.3.1 Микросервисная архитектура.	13
1.3.2 Шаблон <i>Controller-Service-Repository</i>	15
1.4 Инструменты разработки.....	15
1.4.1 Язык <i>Java</i>	15
1.4.2 <i>Spring Boot</i>	16
1.4.3 <i>Spring Data JPA</i>	17
1.4.4 <i>PostgreSQL</i>	17
1.4.5 <i>HikariCP</i>	18
1.4.6 <i>REST API</i>	19
1.4.7 <i>Spring Security Authorization Server</i>	20
1.4.8 <i>Maven</i>	21
2 Технология реализации клиент-серверной системы для планирования, распределения и управления проектами	23
2.1 Реализация	23
2.1.1 Ресурсный сервер	23
2.1.1.1 Слой хранения данных.....	23
2.1.1.2 Слой сервисов	28
2.1.1.3 Слой контроллеров.....	29
2.1.2 Сервис авторизации и аутентификации	31
2.1.3 Клиентское приложение	32
2.2 Описание работы системы.....	33
3 Экономическое обоснование разработки	45

4 Охрана труда	49
4.1 Анализ негативных факторов, воздействующие на человека	49
4.1.1 Эргономика рабочего места	49
4.1.2 Правила перерывов и режима работы	50
4.1.3 Электробезопасность	50
4.1.4 Освещенность рабочего места	51
Заключение	53
Список использованных источников	54
Приложение А – <i>UML</i> -диаграммы	55
Приложение А.1 – Диаграмма вариантов использования	56
Приложение А.2 – Диаграмма последовательности	57
Приложение А.3 – Диаграмма классов клиента	58
Приложение А.4 – Диаграмма классов ресурсного сервера.....	59
Приложение А.5 – Диаграмма классов auth-сервера	60
Приложение Б – листинг программы.....	61
Приложение В - презентация.....	118

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

API (Application Programming Interface) - интерфейс прикладного

CRUD (Create, Read, Update, Delete) – обозначение основных функций сущности, хранящийся в базе данных

Framework – программная платформа, которая упрощает разработку программного продукта, определяет структуру проекта

JAR (Java ARchive) - это Java-архив

JDBC платформенно независимый промышленный стандарт взаимодействия *Java-приложений* с различными СУБД

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на *JavaScript*

JWT (JSON Web Token) - это открытый стандарт (*RFC 7519*) для создания секретных ключей доступа, основанный на формате *JSON*

MAVEN - фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке *POM*

программирования, который представляет собой набор определений и протоколов для создания и интеграции прикладного программного обеспечения

Postgresql - свободная реляционная система управления базами данных

REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SQL (Structured Query Language) - декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

Введение

Технологическая революция, начавшаяся в последней четверти 20-го века, привела к невероятному расцвету информационных технологий, затрагивающем многие аспекты нашего общества. Одним из ключевых моментов этой революции было появление и быстрое распространение клиент-серверных систем. Возникшая в 1980-х годах, эта концепция стала основой современного подхода к обмену и обработке информации, а сегодня применяется в широком диапазоне областей, от коммерческого сектора до образования и науки.

В эпоху просвещения, в начале XVIII века, возникла идея проектного управления как метода, помогающего человеку управлять сложными задачами. За прошедшие столетия, этот подход претерпел значительные изменения, совершенствуясь в соответствии с требованиями развивающегося мира. В новейшее время, в связи с глобализацией и растущим количеством проектов, масштаб и сложность которых неуклонно растут, образовалась острая необходимость в эффективных средствах для планирования, распределения и управления проектами.

История информационных технологий – это история постоянного поиска более эффективных и гибких инструментов для работы с данными. С этой точки зрения, управление проектами представляет собой одну из самых сложных и важных задач, которые требуют решения в современном мире. Сегодня мы имеем дело с глобальными проектами, в которых участвуют тысячи людей, работающих в разных часовых поясах и с разными культурами. Поддержание эффективного и слаженного процесса управления в таких условиях – это не просто задача, это настоящий вызов. Однако, традиционные методы управления проектами, включая различные варианты таблиц и диаграмм Ганта, могут не справиться с уровнем сложности, который представляют собой современные проекты. Это может привести к множеству проблем, начиная от ошибок и пропусков, вызванных недостатком

					ПГУ 1.090501.09.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

информации или ее устареванием, и заканчивая серьезными задержками, которые могут подорвать доверие клиентов и повлечь за собой существенные финансовые потери.

Напротив, клиент-серверная система для планирования, распределения и управления проектами может предложить эффективное и гибкое решение для этих вызовов. Этот подход предполагает централизацию данных и автоматизацию многих процессов, что позволяет оперативно реагировать на изменения и принимать обоснованные решения. Благодаря своей способности обрабатывать большие объемы информации и предоставлять доступ к ней в режиме реального времени с любого устройства с подключением к интернету, такая система может существенно улучшить процесс управления проектами.

Основой для создания такой системы стали современные технологии и методы, разработанные и опробованные в течение последних десятилетий, а также исторический опыт, который позволил нам научиться управлять проектами более эффективно. Однако, стоит помнить, что внедрение и разработка таких систем — это сложная и трудоемкая задача. Она требует тщательного анализа множества факторов, от технических характеристик и функциональных требований до вопросов безопасности и конфиденциальности. Поэтому, анализ, проектирование и разработка клиент-серверной системы для планирования, распределения и управления проектами — это не только сложная, но и весьма актуальная и значимая задача для нашего времени.

1 Клиент-серверная система для планирования, распределения и управления проектами

1.1 Анализ предметной области

В современном быстро развивающемся бизнес-ландшафте, и особенно в IT-сфере, компетентность в управлении проектами стала критическим фактором успеха. Ведь проекты могут варьироваться от разработки программного обеспечения до управления изменениями в инфраструктуре и цифровых маркетинговых кампаниях.

Проекты, как правило, включают в себя множество различных задач, которые могут быть распределены по отдельным сотрудникам или командам. Некоторые из этих задач могут выполняться параллельно, в то время как другие могут требовать последовательного выполнения, так как следующая задача зависит от успешного завершения предыдущей. Для обеспечения эффективного выполнения таких проектов необходимо иметь хорошо разработанный план, который учитывает все аспекты процесса.

Однако разработка плана — это только начало. Важно также уметь контролировать ход выполнения этих задач. Это может включать в себя регулярные проверки статуса задач, обновление планов в ответ на изменения в сроках или доступных ресурсах, а также общение с командами и отдельными участниками, чтобы убедиться, что все продвигаются в соответствии с планом.

Кроме того, эффективное управление проектами требует строгого соблюдения дедлайнов. Это может включать в себя не только отслеживание сроков выполнения задач, но и управление рисками, которые могут привести к задержкам. Например, если критическая задача отстает от графика, менеджер проекта может потребоваться предпринять действия, чтобы ускорить ее выполнение или перераспределить ресурсы, чтобы минимизировать влияние на общий проект.

Наконец, эффективное управление проектами включает в себя грамотное распределение ресурсов. Это может означать ассигнование сотрудников на определенные задачи, управление бюджетом проекта, обеспечение необходимого оборудования или программного обеспечения, а также координация с внешними поставщиками.

Итак, управление проектами в IT-сфере — это сложный и многоаспектный процесс, который требует не только технических навыков, но и навыков лидерства, стратегического планирования, управления рисками, коммуникаций и многого другого.

Существующие инструменты управления проектами могут предложить базовый функционал для решения этих задач, однако зачастую они не обладают достаточной гибкостью или слишком сложны в использовании. Кроме того, может возникнуть потребность в автоматизации определенных процессов, например, в создании отчетов о выполнении задач или в оповещении участников проекта о наступающих дедлайнах.

В этом случае решением может стать разработка собственной клиент-серверной системы для планирования, распределения и управления проектами. При правильной реализации, она может обеспечить удобный интерфейс для работы с проектами, гибкие настройки для адаптации под конкретные бизнес-процессы и механизмы автоматизации для повышения продуктивности работы.

Например, система может автоматически создавать и отправлять отчеты о статусе проекта руководству компании, оповещать участников проекта о наступающих сроках выполнения задач или даже автоматически назначать задачи на сотрудников на основе их текущей загрузки и навыков. Такой подход не только упростит управление проектами, но и значительно повысит их эффективность.

Ключевые инструменты управления проектами, такие как *Trello*, *Microsoft Project* и *Jira*, обладают функционалом планирования и

распределения задач. Эти возможности позволяют планировать проекты, назначать задачи и отслеживать их выполнение. *Trello* и *Jira* представляют следующие преимущества:

- создание досок и карточек для задач, что позволяет визуализировать проект;
- назначение исполнителей на конкретные задачи и установка дедлайнов;
- обсуждение задач в комментариях и добавление вложений;
- интеграция с другими сервисами и программами.

Но стоит отметить и некоторые ограничения этих систем:

- в каждой системе имеются ограничения на количество бесплатных досок или проектов;
- настройка проекта и задач требует времени и определённых навыков;
- интерфейс может быть сложен для новых пользователей;
- не всегда предоставляют удобные инструменты для аналитики и отчетности;
- нет *API*, через который можно было бы управлять всеми функциями системы.

Microsoft Project – это специализированный инструмент для проектного менеджмента, который предоставляет более глубокие возможности управления проектами. Плюсы этого инструмента:

- предоставляет расширенные инструменты для планирования и управления проектами;
- позволяет составлять детальные графики работ;
- позволяет управлять ресурсами и бюджетом проекта;
- интеграция с другими продуктами *Microsoft*.

Минусы:

- для использования требуется подписка, которая может быть существенной для некоторых компаний;

					ПГУ 1.090501.09.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

- требует обучения для эффективного использования;
- интерфейс может быть перегружен для новых пользователей;
- нет удобного *API* для интеграции с другими системами;
- ограниченная возможность совместной работы в реальном времени.

Практически со всеми проблемами выше справляется платформа *Asana*, которая предлагает целый ряд функций для повышения производительности и эффективности работы. Это интуитивно понятный инструмент, с которым каждый пользователь сможет быстро освоиться. Кроме того, она предлагает обширную поддержку пользователей посредством чата, электронной почты и телефона.

С помощью *Asana*, можно планировать и отслеживать задачи, распределять их между командами и устанавливать сроки выполнения. Управление проектами становится более прозрачным благодаря детальной визуализации каждого этапа. Благодаря удобному интерфейсу, можно настроить уведомления и напоминания о сроках исполнения задач. К тому же, *Asana* предлагает интеграцию с множеством других сервисов, что упрощает процесс работы.

Особенностью *Asana* является возможность создания персонализированных досок и списков задач. Это позволяет удобно управлять работой как в рамках одного проекта, так и на уровне всей организации. Инструменты аналитики позволяют отслеживать выполнение задач и анализировать эффективность работы команды, что способствует повышению производительности.

Однако, у *Asana* есть и свои недостатки:

- *Asana* – платная платформа, хоть и предлагает бесплатную версию с ограниченными функциями;
- отсутствие русскоязычной поддержки может стать проблемой для некоторых пользователей;

- множество лишних технологий ненужных для реализации нашей задачи, за которые надо платить.

Из всех существующих технологий, на данный момент в открытом доступе нет бесплатного и гибкого ПО для управления проектами.

1.2 Постановка задачи

Необходимо разработать клиент-серверную систему, которая будет управлять проектами.

Система должна удовлетворять следующим требованиям:

- возможность создавать/удалять/редактировать проекты;
- возможность создавать/удалять/редактировать задачи для проектов;
- необходимо реализовать систему аутентификации и авторизации;
- интерфейс нужно реализовать в виде сайта.

Требования к аутентификации и авторизации:

- каждый пользователь имеет роли и права: пользователи с ролью «*Developer*» должны иметь ограниченные права, позволяющие закрывать задачи. Пользователям с ролью «*Teamlead*» доступна возможность создавать задачи, распределять их между пользователями с ролью «*Developer*» и устанавливать сроки, а также закрывать проекты. Пользователи с ролью «*Manager*» могут создавать проекты, устанавливать сроки и назначать на пользователей роли.

Требования к интерфейсу:

- просмотр проектов и задач;
- добавление, удаление и редактирование проектов и задач.
- изменение данных пользователя;
- отдельная форма авторизации.

1.3 Архитектура приложения

1.3.1 Микросервисная архитектура.

Программное решение будет разработано с использованием нескольких микросервисов, поскольку на данный момент этот подход является наиболее удобным и актуальным.

Микросервисы представляют собой стратегию разработки и структурирования программного обеспечения, где приложение формируется из набора малых самостоятельных служб, работающих через четко определенные API. Эти службы управляются отдельными автономными командами.

Микросервисная архитектура облегчает процесс масштабирования и ускоряет создание приложений, способствуя внедрению инноваций и быстрому запуску новых функций на рынок.

Характеристики микросервисов:

- автономность. Каждая компонентная служба в архитектуре микросервисов может разрабатываться, развертываться, эксплуатироваться и масштабироваться без влияния на функционирование других служб. Службы не должны делиться своим кодом или реализацией с другими службами. Любая связь между отдельными компонентами происходит через четко определенные *API*;

- каждый сервис рассчитан на набор возможностей и ориентирован на решение конкретной задачи. Если разработчики добавляют больше кода в службу с течением времени, и служба становится сложной, ее можно разбить на более мелкие службы.

Преимущество микросервисов:

- гибкость. Микросервисы способствуют организации небольших независимых команд, которые ответственно за свои услуги. Команды действуют в небольшом и хорошо понятном контексте и имеют возможность

					ПГУ 1.090501.09.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

работать более независимо и быстрее. Это сокращает время цикла разработки. Вы получаете значительную выгоду от совокупной пропускной способности организации;

- масштабируемость. Микросервисы позволяют независимо масштабировать каждую службу для удовлетворения спроса на поддерживаемую ею функцию приложения. Это позволяет командам правильно определять потребности инфраструктуры, точно измерять стоимость функции и поддерживать доступность, если на услугу наблюдается всплеск спроса;

- простота развертывания. Микросервисы обеспечивают непрерывную интеграцию и непрерывную доставку, что позволяет легко пробовать новые идеи и откатываться, если что-то не работает. Низкая стоимость отказа позволяет экспериментировать, упрощает обновление кода и ускоряет вывод на рынок новых функций;

- технологическая свобода. Архитектуры микросервисов не следуют подходу «один размер подходит всем». Команды имеют свободу выбора лучшего инструмента для решения своих конкретных проблем. Как следствие, команды, создающие микросервисы, могут выбрать лучший инструмент для каждой работы;

- многоразовый код. Разделение программного обеспечения на небольшие, четко определенные модули позволяет командам использовать функции для различных целей. Сервис, написанный для определенной функции, может использоваться в качестве строительного блока для другой функции;

- сочетание технологий. Команды могут выбрать технологию, которая лучше всего подходит для их службы, используя сочетание технологических стеков по мере необходимости;

- изоляция данных. Гораздо проще выполнять обновления схемы, поскольку затрагивается только один микросервис. В монолитном

					ПГУ 1.090501.09.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

приложении обновление схемы может стать очень сложным, поскольку разные части приложения могут касаться одних и тех же данных, что делает любые изменения схемы рискованными.

1.3.2 Шаблон *Controller-Service-Repository*

Шаблон *Controller-Service-Repository* является распространенным в большинстве приложений *Spring Boot*. Этот шаблон применяется из-за его эффективности в разделении обязанностей. В этом контексте, контроллер, находящийся на верхнем уровне, полностью отвечает за предоставление функциональности для использования внешними сущностями, включая, возможно, элементы пользовательского интерфейса. Слой, отвечающий за работу с данными (*Repository*), находится в нижней части и обеспечивает хранение и получение определенного набора данных. Уровень сервиса занимается всей бизнес-логикой. Если бизнес-логика требует получения или сохранения данных, она взаимодействует с слоем данных. Если требуется доступ к бизнес-логике, он осуществляется через контроллер.

Это достаточно простое разграничение обязанностей. Если код относится к хранению или поиску, он располагается в слое данных. Если речь идет о предоставлении функциональности, это относится к контроллеру. Все уникальное для бизнес-логики находится на уровне сервиса. Уровень сервиса не заботится о том, как к нему обращаются, его задача - выполнять свою работу, используя слой данных там, где это необходимо. А контроллер просто делегирует работу уровню сервиса, что позволяет ему оставаться «легким».

1.4 Инструменты разработки

1.4.1 Язык *Java*

Система будет разработана на языке *Java*, поскольку *Java* представляет собой надежный, понятный и широко используемый среди разработчиков язык программирования. В контексте создания микросервисной архитектуры,

					ПГУ 1.090501.09.001 ПЗ	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

Java особенно полезен благодаря своему простому синтаксису аннотаций, что облегчает процесс разработки. Более того, *Java* представляет собой прекрасный выбор, так как предоставляет возможности по созданию пользовательского интерфейса, интеграции с внутренними ресурсами и элементами модели.

Java демонстрирует высокую стоимость в контексте читаемости кода, особенно при работе с сложными системами.

1.4.2 *Spring Boot*

Главной библиотекой, облегчающей разработку по шаблонам и поддерживающей построение микросервисной архитектуры, является *Spring Boot*.

Spring Boot облегчает процесс создания автономных приложений на базе *Spring*, которые можно просто запустить без особых хлопот. Большинству приложений, разработанных на *Spring Boot*, требуется минимум настройки *Spring*.

Преимущества *Spring Boot* включают в себя [2]:

- упрощенное и конфликт-свободное управление зависимостями;
- возможность быстро настраивать и запускать автономные веб-приложения и микросервисы;
- простоту сборки артефактов, которые поставляются со встроенными серверами приложений, такими как *Tomcat*, *Jetty* или *Undertow*;
- предоставление *API*, который обеспечивает доступ к внутренним данным приложения, включая детализированные метрики, внутренний механизм работы приложения, статус его производительности;
- отсутствие необходимости в конфигурации на основе *XML*. Наличие упрощенных свойств. Классы, управляемые *Spring* контейнером, автоматически инициализируются, настраиваются и связываются.

1.4.3 Spring Data JPA

База данных будет хранить список пользователей и их информацию. Для взаимодействия с базой данных мы воспользуемся технологией *Spring Data JPA*. *JPA* означает *Java Persistence API*.

Spring Data JPA предоставляет программный интерфейс для управления реляционными данными, используя упрощенные методы доступа к данным. Это позволяет сохранять, обновлять, удалять и извлекать информацию из базы данных.

Spring Data JPA существенно упрощает работу с базами данных и позволяет сосредоточиться на написании бизнес-логики, а не на коде для выполнения операций *CRUD*. Он предоставляет готовые к использованию интерфейсы репозитория, а также дает возможность определения пользовательских запросов с помощью знакомого синтаксиса

1.4.4 PostgreSQL

В микросервисе в качестве СУБД применяется *PostgreSQL*. *PostgreSQL* — это мощная система управления реляционными базами данных с открытым исходным кодом, основанная на языке *SQL*.

Преимущества *PostgreSQL*:

- свободное использование. Как и *MySQL*, *PostgreSQL* является системой управления базами данных с открытым исходным кодом, что позволяет использовать его бесплатно;
- портативность. *PostgreSQL* поддерживает множество операционных систем, включая *Linux*, *Windows*, *MacOS* и др;
- безопасность подключения. *PostgreSQL* предлагает различные механизмы подключения для безопасного и надежного доступа к серверу баз данных;

- высокая производительность и надежность. Благодаря своей высокоэффективной архитектуре, *PostgreSQL* предлагает отличную производительность при работе с большими наборами данных.

Недостатки:

- сложность. *PostgreSQL* имеет более сложную кривую обучения по сравнению с некоторыми другими СУБД;

- низкая производительность при обработке некоторых видов транзакций, хотя *PostgreSQL* обычно хорошо справляется с большинством транзакционных нагрузок;

- несмотря на то, что *PostgreSQL* предоставляет множество инструментов для разработки и отладки, некоторые из них могут быть сложными в использовании для начинающих разработчиков;

- *postgreSQL* требует более тщательной настройки и оптимизации для обеспечения высокой производительности на больших объемах данных.

1.4.5 HikariCP

Чтобы оптимизировать работу микросервиса, стоит уделить внимание использованию пула соединений с базой данных. Пул соединений — это подход, который используется для увеличения производительности при работе с базой данных в динамических приложениях. Процесс открытия и закрытия соединений с базой данных может казаться незначительным в плане затрат, однако при масштабировании это может оказаться дорогостоящим.

Допустим, затраты времени на установление соединения составляют 5 мс, а на выполнение запроса - еще 5 мс (примерные цифры). В этом случае половина времени уходит именно на установление соединения. Если количество запросов возрастает до тысячи или десятков тысяч, время на установление соединений может оказаться существенно большим.

Использование пулов соединений обосновано следующими причинами:

- открытие и закрытие соединений требует ресурсов. Чтобы сэкономить их, соединения кэшируются и повторно используются;
- в случае резкого увеличения активности можно контролировать количество подключений к базе данных. Это позволяет избегать ситуаций, когда код ожидает доступного соединения, что особенно полезно в распределенных средах;
- разделение общих операций на несколько пулов позволяет более эффективно управлять ресурсами.

HikariCP — это пример быстрого и легкого пула соединений *Java*. Его *API* и кодовая база отличаются компактностью и высокой степенью оптимизации. Важным преимуществом *HikariCP* является то, что он не жертвует производительностью в погоне за упрощением, в отличие от некоторых других реализаций пулов соединений *Java*.

1.4.6 *REST API*

Для общения с микросервисом через браузер, другие микросервисы или системы, необходимо разработать набор определений и протоколов для создания *API*. Этот *API* будет строиться на основе архитектурного подхода *REST*. *REST* — это стандарт взаимодействия между системами в Интернете, который упрощает их совместную работу.

API, следующий этим принципам, называется *RESTful API*. Он обеспечивает гибкость, высокую пропускную способность и быстрое действие, что полезно для интеграции и коммуникации программного обеспечения.

REST API выступает связующим звеном между клиентом и сервером, отправляя запросы на ресурсы и получая ответы через протокол *HTTP*. В контексте *REST*, клиент инициирует взаимодействие. Например, обращение к веб-странице по *URL* обычно реализуется через *HTTP*-запрос.

Наиболее часто используемые методы *HTTP* включают:

- *GET*: для получения ресурса;

- *POST*: для создания нового ресурса;
- *PUT*: для обновления ресурса;
- *DELETE*: для удаления ресурса.

1.4.7 *Spring Security Authorization Server*

Для обеспечения безопасности системы применяется *Spring Security Authorization Server* в качестве отдельного микросервиса.

Spring Security Authorization Server предоставляет возможности аутентификации и авторизации, используя продуманные фильтры. Интернет-приложения часто подвержены угрозам безопасности и атакам, поскольку они доступны всем пользователям сети. Возможно, существуют конечные точки *REST*, доступ к которым должен быть ограничен для определенных пользователей. Например, операции обновления данных или действия, предназначенные для администратора.

Spring Security Authorization Server позволяет защитить *URL*-адреса ваших приложений. Это мощная платформа, предназначенная для обеспечения безопасности корпоративных приложений, благодаря гибким и настраиваемым функциям безопасности.

В основе работы *Spring Security Authorization Server* лежат три ключевых принципа:

- аутентификация - процесс проверки учетных данных пользователя;
- авторизация - процесс предоставления или отказа в доступе к ресурсам после успешной аутентификации;
- управление паролями - *Spring Security* предлагает ряд классов для их безопасного хранения и хеширования;
- фильтрация *HTTP*-запросов - важная составляющая обеспечения безопасности приложения.

1.4.8 *Maven*

Сборка исполняемого приложения из исходного кода требует инструмента сборки, который также используется для таких операций, как генерация исходного кода, создание документации кода, компиляция кода, упаковка скомпилированных модулей в *JAR*-файлы или *Docker*-образы. Один из наиболее распространенных и удобных инструментов для этих целей - *Maven* [4].

Maven является популярным инструментом сборки с открытым исходным кодом от Apache, предназначенным для создания, распространения и развертывания различных проектов, что облегчает управление проектами.

Благодаря использованию модели проекта *POM* (*Project Object Model*), *Maven* значительно облегчает работу разработчиков Java, предоставляя удобные механизмы для создания отчетов, контроля сборки и автоматизации тестирования.

Среди наиболее важных функций *Maven* можно выделить:

- большой и постоянно обновляемый репозиторий библиотек;
- удобная настройка проектов с использованием лучших практик;
- управление зависимостями с автоматическим обновлением;
- обратная совместимость с предыдущими версиями;
- качественная отчетность по ошибкам и целостности;
- автоматическое управление версиями родительских проектов;
- гарантирует единообразное использование во всех проектах;
- расширяемость за счет возможности написания собственных плагинов

на *Java* или скриптовых языках.

Ценность *Maven* основана на использовании объектной модели проекта (*POM*) — это *XML*-файл, который содержит всю информацию о проекте и настройки его конфигурации. *POM* включает описание проекта, детали управления версиями и конфигурацией проекта [5].

Maven широко используется в проектах на основе *Java*, помогая управлять зависимостями, связанными с библиотеками или *JAR*-файлами. Инструмент обеспечивает загрузку подходящих версий *JAR*-файлов для каждого проекта, учитывая, что могут существовать различные версии отдельных пакетов.

С помощью *Maven*, загрузка зависимостей не требует посещения официальных сайтов разных программ, и он также помогает установить правильную структуру проекта.

					ПГУ 1.090501.09.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

2 Технология реализации клиент-серверной системы для планирования, распределения и управления проектами

2.1 Реализация

Данная работа разделена 3 микросервиса.

В соответствии с этим разработку системы можно разделить на следующие этапы:

- разработка ресурсного сервера;
- разработка сервиса авторизации и аутентификации;
- разработка клиентского приложения.

2.1.1 Ресурсный сервер

В ресурсном сервере реализован архитектурный паттерн «*Controller-Service-Repository*», который разделяет систему на слои. В соответствии с этим разработку системы можно разделить на следующие этапы:

- разработка слоя хранения данных;
- разработка сервисов;
- разработка контроллеров.

2.1.1.1 Слой хранения данных

В работе было создано 5 таблиц: «*t_user*», «*t_role*», «*t_user_roles*», «*t_project*», «*t_task*». Логическая модель базы данных представлена на рисунке 2.1.

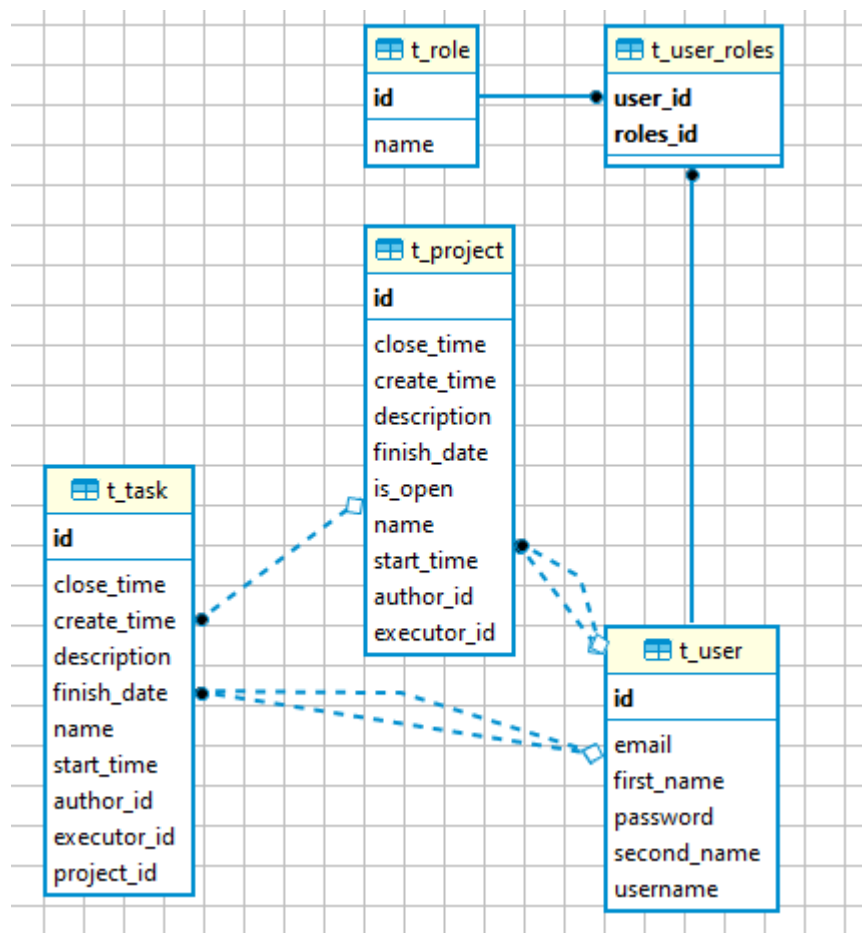


Рисунок 2.1 - Логическая модель базы данных

Таблица «*t_user*» представляет всех пользователей. Таблица хранит персональные данные, такие как имя, фамилию, адрес электронной почты, логин, пароль. Ее структура описана в таблице 2.1.

Таблица 2.1 – Структура таблицы «*t_user*»

Название поля	Описание поля	Тип данных
<i>id</i>	Уникальный идентификатор	<i>Varchar</i>
<i>first_name</i>	Имя	<i>Varchar</i>
<i>second_name</i>	Фамилия	<i>Varchar</i>
<i>email</i>	Электронная почта	<i>Varchar</i>
<i>username</i>	Логин	<i>Varchar</i>
<i>password</i>	Пароль	<i>Varchar</i>

Таблица 2.2 содержит описание структуры таблицы «*t_role*», в которой хранится информация о ролях: идентификатор и название роли.

Таблица 2.2 – Структура таблицы «*t_role*»

Название поля	Описание поля	Тип данных
<i>id</i>	Уникальный идентификатор	<i>Varchar</i>
<i>name</i>	Название роли	<i>Varchar</i>

В таблице «*t_user_roles*» представлены роли пользователей. При удалении пользователя, все его роли также удаляются. Структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «*t_user_roles*»

Название поля	Описание поля	Тип данных
<i>roles_id</i>	Идентификатор роли	<i>Varchar</i>
<i>user_id</i>	Идентификатор пользователя	<i>Varchar</i>

Информация о проектах хранится в таблице «*t_project*», в ней содержится два внешних ключа: «*author_id*» и «*executor_id*», они связывают с таблицей «*t_user*», связью «один ко многим». Ее структура описана в таблице 2.4.

Таблица 2.4 – Структура таблицы «*t_project*»

Название поля	Описание поля	Тип данных
<i>id</i>	Уникальный идентификатор	<i>Varchar</i>
<i>close_time</i>	Время закрытия проекта	<i>Timestamp</i>
<i>create_time</i>	Время создания проекта	<i>Timestamp</i>
<i>description</i>	Описание проекта	<i>Varchar</i>

Название поля	Описание поля	Тип данных
<i>finish_date</i>	Планируемая дата закрытия проекта	<i>Timestamp</i>
<i>start_time</i>	Время начала работы над проектом	<i>Timestamp</i>
<i>is_open</i>	Флаг открытия проекта	<i>Boolean</i>
<i>name</i>	Название проекта	<i>Varchar</i>
<i>author_id</i>	Идентификатор автора проекта	<i>Varchar</i>
<i>executor_id</i>	Идентификатор исполнителя проекта	<i>Varchar</i>

Информация о задачах хранится в таблице «*t_task*», в ней содержится три внешних ключа: «*author_id*» и «*executor_id*» связывают с таблицей «*t_user*», а ключ «*project_id*», с таблицей «*t_project*», связью «один ко многим». Структура данной таблицы показана в таблице 2.5.

Таблица 2.5– Структура таблицы «*t_task*»

Название поля	Описание поля	Тип данных
<i>id</i>	Уникальный идентификатор	<i>Varchar</i>
<i>close_time</i>	Время закрытия задачи	<i>Timestamp</i>
<i>create_time</i>	Время создания задачи	<i>Timestamp</i>
<i>description</i>	Описание проекта	<i>Varchar</i>
<i>finish_date</i>	Планируемая дата закрытия задачи	<i>Timestamp</i>
<i>start_time</i>	Время начала работы над задачей	<i>Timestamp</i>
<i>is_open</i>	Флаг открытия задачи	<i>Boolean</i>
<i>name</i>	Название задачи	<i>Varchar</i>
<i>author_id</i>	Идентификатор автора задачи	<i>Varchar</i>

Название поля	Описание поля	Тип данных
<i>executor_id</i>	Идентификатор исполнителя задачи	<i>Varchar</i>
<i>project_id</i>	Идентификатор исполнителя проекта	<i>Varchar</i>

Во время запуска микросервиса, автоматически запускаются подготовленные *SQL*-скрипты, которые создают соответствующие таблицы в базе данных, если их нет.

Взаимодействие с базой данных происходит в 4 классах. Каждый класс имеет методы для взаимодействия с таблицами, такие как вставка, обновление, выборка и удаление.

Для обращение к базе данных используется наследование от класса «*JpaRepository*», который представлен в пакете «*org.springframework.data.jpa*». Данный класс облегчает использование базы данных и выполняет основной рабочий его процесс. Класс имеет много удобных методов, которые осуществляют *SQL*-запросы, помогают конвертировать приходящий результат в объекты *java*, а также ловить ошибки JDBC и представлять их в привычные исключения. Чтобы использовать этот класс достаточно создать интерфейс без реализации, который наследуется от него, написать в этом интерфейсе методы, названия которых будут соответствовать желаемому запросу, например, для поиска всех проектов можно сделать метод «*List<Project> findAll()*», чтобы найти проекты с определенным автором нужно добавить в название метода «*byAuthorId*», и входящий параметр: «*List<Project> findAllByAuthorId(String userId)*», для более сложных запросов используется аннотация *@Query*, в которую передается SQL запрос, например: «*@Query("SELECT * FROM t_project")*».

Физическая модель базы данных представлена на рисунке 2.2.

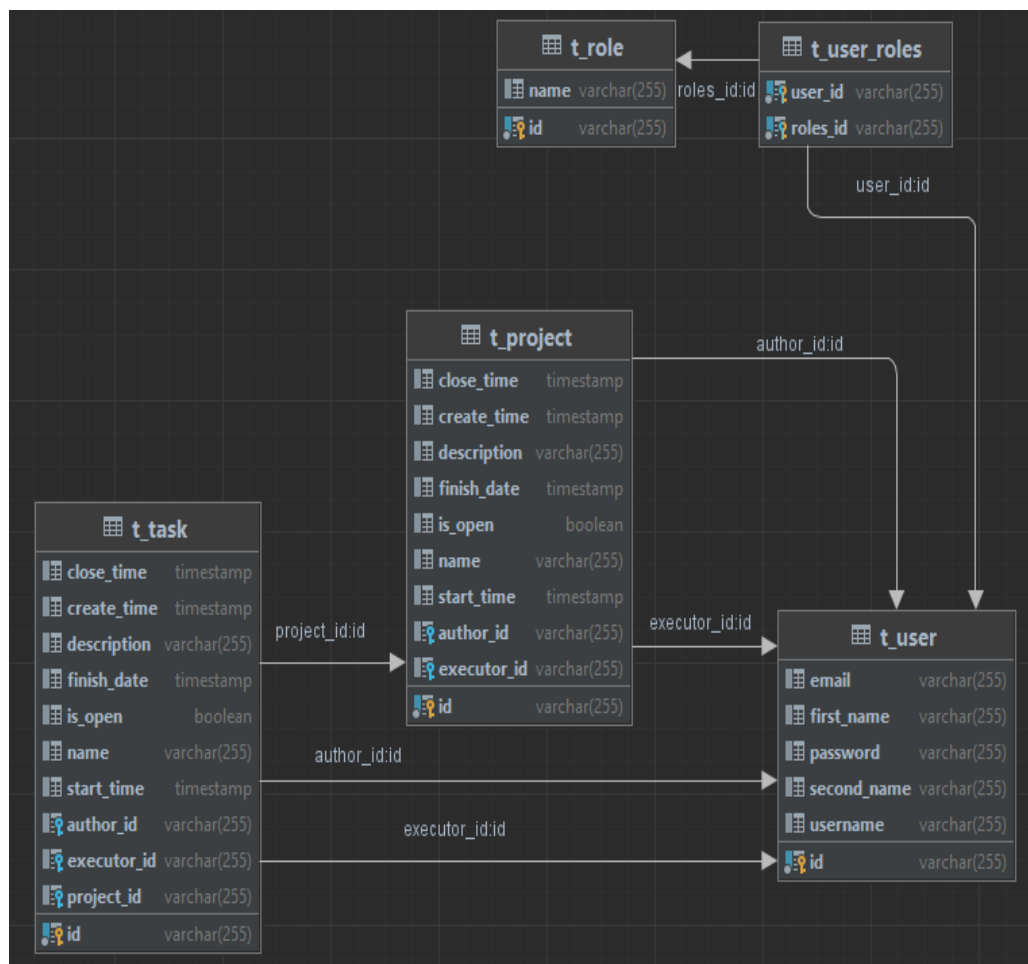


Рисунок 2.2 - Физическая модель базы данных

2.1.1.2 Слой сервисов

Слой сервисов служит основой для основных функций микросервиса, выполняя задачи валидации данных и реализации бизнес-логики. Он также обеспечивает связь между контроллером и уровнем управления данными.

Сервисы, представляющие собой методы для выполнения *CRUD*-операций с такими объектами как пользователь, проект и задача, напрямую связаны с уровнем управления данными. Эти методы принимают входные данные от контроллеров, которые затем подвергаются валидации. Валидация представляет собой процесс проверки правильности данных, обычно на предмет отсутствия значений. Если данные не проходят проверку, генерируются соответствующие ошибки. После успешной валидации

выполняется необходимая бизнес-логика, включая обращение к уровню управления данными. Пример метода поиска проекта по *id* в сервисе «*ProjectServiceImpl*» представлен ниже:

```
@Override
public Project findById(String id) {
    if (StringUtils.hasLength(id)) throw new
    ProjectIdEmptyException();
    final Optional<Project> project = projectDao.findById(id);
    if (project.isEmpty()) throw new ProjectNotFoundException();
    return project.get();
}
```

На вход данного метода приходит строка, которую мы проверяем на пустоту с помощью утилитного класса «*StringUtils*» библиотеки «*org.springframework.util*» и его метода *hasLength*. Если поле пустое, то выбрасывается соответствующее исключение. Далее идет обращение к классу, работающему с базой данных для поиска проекта, затем идет проверка, что проект найден. После успешного нахождения в метод возвращает этот объект.

Подобная структура характерна и для методов вставки, обновления и удаления.

2.1.1.3 Слой контроллеров

Уровень контроллеров занимается обработкой входящих запросов от пользователей и возвращением результатов этих запросов. На этом уровне не должно быть никакой бизнес-логики, изменений состояния модели или непосредственного обращения к базе данных.

В этом микросервисе, контроллеры управляют *REST-API*, который использует *HTTP*-запросы для извлечения, обновления, удаления и создания данных. *Spring* облегчает реализацию этой функциональности с помощью специальных аннотаций. Если запрос предназначен для создания данных, он должен быть помечен аннотацией «*@PostMapping*». Для запросов на поиск используется аннотация «*@GetMapping*», для удаления - «*@DeleteMapping*», а для обновления данных - «*@PutMapping*». В параметрах этих аннотаций

указывается универсальный идентификатор ресурса - *URL* ресурса в интернете, а также формат обмена данными. В этом микросервисе используется формат *JSON*, который легко читается и выглядит визуально привлекательно. Если пользователь передает тело запроса, которое нужно преобразовать в объект Java, то используется аннотация «*@RequestBody*». В случае несоответствия тела запроса ожидаемому объекту Java, генерируется исключение. При передаче информации в самом *URL* запроса следует использовать аннотацию «*@PathVariable*». Пример такого метода представлен ниже:

```
@GetMapping("/projectId={projectId}")
public List<Task> findAllByProjectId(@PathVariable String
projectId)
{
    return taskService.findAllByProjectId(projectId);
}
```

Здесь идентификатор пользователя передается внутри адреса запроса и вставляется в объект «*id*» класса «*String*».

Пример метода, в который передается тело запроса:

```
@PostMapping
public void createUser(@RequestParam User user)
{
    userService.createUser(user);
}
```

Запрос при таком методе должен соответствовать структуре:

```
{
  "id": "id",
  "email": "email@mail.ru",
  "firstName": "Oleg",
  "lastName": "Andreev",
  "username": "username",
  "password": "password"
}
```

2.1.2 Сервис авторизации и аутентификации

Безопасность в микросервисе реализована с помощью отдельного сервера авторизации. *Spring Authorization Server* — это проект от команды *Spring*, который предоставляет возможности для создания сервера авторизации в соответствии со спецификацией *OAuth 2.0*. *OAuth 2.0* — это стандарт, используемый для делегирования доступа к ресурсам без раскрытия пароля пользователя третьей стороне. Вместо этого, третья сторона получает токен, который можно использовать для доступа к ресурсам от имени пользователя. Вот общий обзор того, как работает *Spring Authorization Server*:

1) **Регистрация Клиента:** перед использованием *OAuth 2.0*, клиент (обычно это приложение, которому требуется доступ к пользовательским ресурсам) должен быть зарегистрирован на сервере авторизации. Этот процесс включает в себя получение ключей клиента, которые будут использоваться для аутентификации на сервере авторизации

2) **Авторизация:** когда пользователь пытается получить доступ к ресурсу, который требует аутентификации, он перенаправляется на страницу входа сервера авторизации. Здесь пользователь вводит свои учетные данные и предоставляет разрешения на доступ к своим данным;

3) **Получение Токена:** после успешной аутентификации, сервер авторизации отправляет код авторизации на указанный адрес перенаправления. Клиент затем обменивает этот код на токен доступа, делая запрос на сервер авторизации и предоставляя код авторизации и свои учетные данные;

4) **Использование Токена:** клиент теперь может использовать полученный токен доступа для выполнения запросов к защищенным ресурсам от имени пользователя. Токен добавляется в заголовок запроса и передается на сервер ресурсов. Сервер ресурсов в свою очередь проверяет токен с сервером авторизации;

5) **Окончание Сессии:** токены доступа обычно имеют ограниченный

срок жизни. Когда токен истекает, клиент может обновить его, используя обновляющий токен, который был предоставлен вместе с токеном доступа.

Пароли пользователей хешируются с помощью *bcrypt* - криптографическая хеш-функция формирования ключа. Пользователь также имеет роли. В зависимости от роли пользователю предоставляются те или иные функции и возможности. Блок кода, в котором распределяются роли на адреса, представлен ниже:

```
httpSecurity
    .csrf()
    .disable()
    .authorizeRequests()
    .antMatchers("/css/**",                "/js/**").permitAll()
    .antMatchers("/registration").not().fullyAuthenticated()
    .antMatchers("/admin/**").hasRole("MANAGER")
    .antMatchers("/").permitAll()
    .anyRequest().authenticated()
    .and()
    .formLogin()
    .defaultSuccessUrl("/")
    .permitAll()
    .and()
    .logout()
    .permitAll()
    .logoutSuccessUrl("/");
```

Основной компонент в *Spring Authorization Server* — это *AuthorizationServerConfigurerAdapter*, который используется для настройки сервера авторизации. Он позволяет вам определить, какие клиенты могут соединиться с сервером, какие пользователи могут получить доступ, какие токены будут использоваться и т.д.

Важно отметить, что на момент моего последнего обновления (май 2023 года) *Spring Security OAuth* поддерживается в режиме обслуживания, и новые проекты рекомендуется использовать *Spring Authorization Server*.

2.1.3 Клиентское приложение

Клиентское приложение реализовано с помощью *Spring Boot*, *HTML*, *CSS* и *JS*. В отличие от ресурсного сервера, здесь присутствует исключительно

					ПГУ 1.090501.09.001 ПЗ	Лист
						32
Изм.	Лист	№ докум.	Подпись	Дата		

слой контроллеров, отвечающих за обработку входных данных из интерфейса и их последующее отображение. Действия этих контроллеров большей частью совпадают с действиями *REST*-контроллеров. Основное различие состоит в том, что в ответ они возвращают не *JSON*, а *HTML*-страницы. Для визуализации мы используем класс "*ModelAndView*", данный класс дает возможность передачи объектов *Java*, поля которых можно внедрять в *HTML* с использованием сервисной функции *Java*-шаблонов - *Thymeleaf*. Пример метода для такого контроллера приведен ниже:

```
@GetMapping
public String getProjects(
    @AuthenticationPrincipal User user,
    Model model
) {
    model.addAttribute("projects",
projectService.findAllByUserId(user.getId()));
    return PROJECTS;
}
```

2.2 Описание работы системы

Чтобы пользоваться функциями приложения для начала необходимо пройти форму авторизации, представленной на рисунке 2.3.

Рисунок 2.3 – Форма авторизации

Если нажать на кнопку «*Login*» при пустых полях ввода, появится предупреждение, представленное на рисунке 2.4.

Рисунок 2.4 – Форма авторизации

После успешной авторизации пользователь перенаправляется на главную страницу сайта, которая показана на рисунке 2.5.

Рисунок 2.5 – Главная страница пользователя с ролью «Teamlead».

В зависимости от роли пользователя на странице будут отображаться те или иные кнопки. Для пользователя с ролью «Teamlead» отображается только кнопки «Projects», «Tasks» и «Logout». Для пользователя с ролью «Manager» помимо вышеупомянутых кнопок также появляется кнопка «Users». Данная страница представлена на рисунке 1.6.

Рисунок 2.6 – Главная страница пользователя с ролью «Manager».

При нажатии кнопки «Project» откроется страница с проектами, на ней также присутствуют фильтры для поиска по названию, автору и исполнителю, а также флаг, чтобы показать скрытые проекты, страница показана на рисунке 2.7.

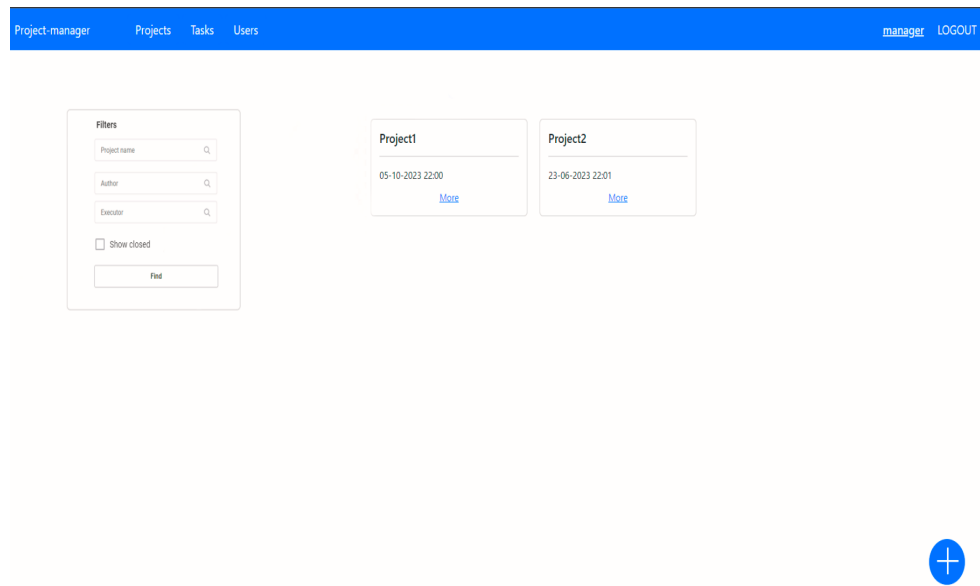


Рисунок 2.7 – Страница с проектами

Чтобы получить больше информации о проекте, необходимо нажать на кнопку «*More*»: карточка раскроется и в ней будет описание, дата создания, дата начала работы, автор, исполнитель и флаг, что проект открыт, также тут можно посмотреть все задачи связанные с проектом, редактировать его и закрыть, кнопки редактирования и закрытия доступны только пользователям с ролью «*Manager*», пример представлен на рисунке 2.8.

Project1

Deadline: 16-06-2023 23:49

Description: description1

Create time: 16-06-2023 23:49

Start time: 17-06-2023 23:49

Author: manager

Executor: dev1

Is open: true

Tasks

Edit

Close

[Less](#)

Рисунок 2.8 – Расширенная информация о проекте

Чтобы создать новый проект, необходимо нажать на кнопку «+», расположенную справа внизу, появится форма, в которую необходимо ввести название, описание, планируемую дату завершения проекта, дату начала работы над проектом и исполнителя, пример на рисунке 2.9.

Add project [X]

Name

Description

Deadline
 [calendar icon]

Start time
 [calendar icon]

Executor

[Close] [Add]

Рисунок 2.9 – Форма создания проекта

Форма редактирования выглядит похоже, но все поля там уже заполнены, пример на рисунке 2.10.

Edit project

×

Name

Project1

Description

Description1

Deadline

16.06.2023 23:49

📅

Start time

17.06.2023 23:49

📅

Executor

dev1

Close

Save

Рисунок 2.10 – Форма редактирования проекта

Чтобы посмотреть все задачи, нужно нажать на кнопку «*Tasks*» в верхней части экрана, также можно сразу перейти к задачам конкретного проекта нажав на кнопку «*Tasks*» на карточке проекта, страница с задачами показана на рисунке 2.11.

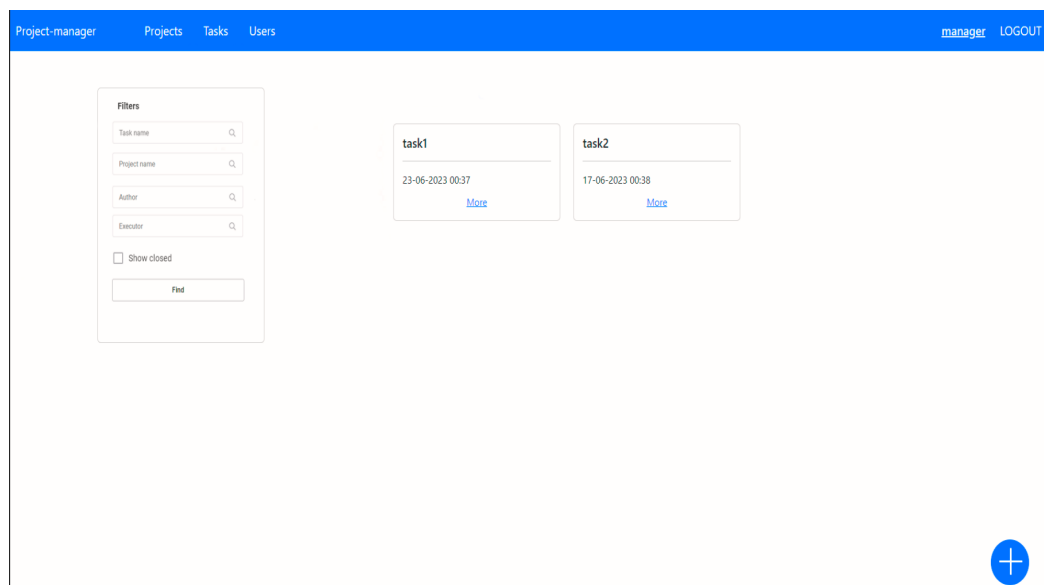


Рисунок 2.11 – Страница с задачами

Чтобы узнать более подробную информацию о задаче необходимо нажать кнопку «*More*», названия всех полей аналогичны тех, что были у проекта, только добавляется одно новое поле с названием проекта к которому относится задача, а также отсутствует кнопка «*Tasks*» пример на рисунке 2.12.

task1

Deadline: 23-06-2023 00:37

Description: desc1

Create time: 16-06-2023 23:49

Start time: 17-06-2023 23:49

Author: manager

Executor: dev1

Is open: true

Project: Project1

Edit

Delete

[More](#)

Рисунок 2.12 – Полная информация о задаче

Чтобы создать задачу необходимо нажать на кнопку «+», справа внизу экрана, форма создания задачи аналогична созданию проекта, только добавляется поле с выбором проекта, пример представлен на рисунке 2.13.

Add task
✕

Name

Name

Description

Description

Deadline

ДД.ММ.ГГГГ --:--

Start time

ДД.ММ.ГГГГ --:--

Executor

Executor

Project

Project1

Close

Add

Рисунок 2.13 – Форма создания задачи

Редактирование задачи происходит аналогично редактированию проекта, только добавляется поле с выбором проекта, рисунок 2.14

Edit task

×

Name

task1

Description

desc1

Deadline

23.06.2023 00:37

📅

Start time

17.06.2023 23:49

📅

Executor

dev1

Project

Project1

▼

Close

Save

Рисунок 2.14 – Форма редактирования задачи

У пользователей с ролью «*MANAGER*», также есть возможность просматривать, редактировать и удалять пользователей, для этого нужно нажать на кнопку «*Users*» в верхней части экрана, страница с пользователями представлена на рисунке 2.15.

Project-manager Projects Tasks Users					manager LOGOUT	
Email	Username	Role	Delete	Manage		
test@mail.ru	dev1	DEVELOPER	Delete	Manage		
test@mail.ru	manager	MANAGER	Delete	Manage		
test@mail.ru	teamlead1	TEAMLEAD	Delete	Manage		

Рисунок 2.15 – Страница с пользователями

Для редактирования пользователя необходимо нажать на кнопку «*Manage*» рядом с нужным пользователем, после чего откроется форма редактирования, в ней можно поменять почту, логин, имя, фамилию и роль или удалить пользователя, пример на рисунке 2.16.

Email address

test@mail.ru

Username

dev1

First name

name2

Second name

name2

Role

DEVELOPER

Delete

Save

Cancel

Рисунок 1.26 – Редактирование пользователя

3 Экономическое обоснование разработки

Разработка клиент-серверной системы для управления проектами осуществлялась в офисе площадью 100 кв.м. Содержание помещения в месяц, с учетом аренды и уборки приблизительно стоит 100000 рублей.

Для разработки были закуплены вычислительные машины и лицензионное программное обеспечение.

Данные по потребности в оборудовании и лицензионном программном обеспечении представлены соответственно в таблице 3.1 и 3.2.

Таблица 3.1 - Использованное оборудование

Оборудование	Кол-во	Ориентировочная цена за единицу, руб
<i>MacBook pro</i>	3	200000

Таблица 3.2 - Использованное лицензионное программное обеспечение

Лицензионное программное обеспечение	Источник	Цена, рублей	Срок действия лицензии, дней
<i>Intellij idea ultimate</i>	сайт производителя	12000	30

В разработке системы мониторинга показаний счетчиков ресурсов осуществлялась командой разработчиков. Состав команды и информация о их заработной плате представлена в таблице 5.3.

Таблица 3.3 - Состав команды

Специалист	Количество, человек	Сумма заработной платы, рублей
<i>Backend</i> разработчик	1	60000
<i>Frontend</i> разработчик	1	50000
Тестировщик	1	40000

Себестоимость разработки клиент-серверной системы включает следующие статьи затрат:

- затраты на приобретение оборудования;
- затраты на дополнительные материалы;
- затраты на заработную плату;
- затраты на обслуживание.

Затраты на оборудование в период разработки составят 600000 рублей на основании данных в таблице 3.1.

Затраты на программное обеспечение в период разработки составят 12000 рублей и представлены в таблице 3.2.

Затраты на дополнительные материалы в период разработки составят 0 рублей в соответствии с условием что потребности в закупке нет.

Затраты на заработную плату включают затраты на основную и дополнительную заработную плату разработчиков программного обеспечения и отчисления на страховые взносы.

Затраты на основную заработную плату разработчиков определяются исходя из установленной заработной платы, количества специалистов и их занятости в процессе разработки программного продукта (продолжительность работы в днях).

Дополнительная заработная плата составляет 15% от затрат на основную заработную плату разработчиков.

С учетом выходных дней при 5-дневной рабочей неделе количество рабочих дней в месяце в среднем составляет 21 день.

Затраты на основную и дополнительную заработную плату разработчиков программного обеспечения представлены в таблице 3.4.

Таким образом, затраты на заработную плату разработчиков составят 450000 рублей.

Отчисления на страховые взносы по нормам, установленным законодательством в ПРФ, ФОМС, ФСС РФ вычисляются исходя из данных представленных ниже.

Таблица 3.4 – Затраты на основную и дополнительную заработную плату разработчиков

Исполнитель	Количество, человек	Сумма заработной платы, рублей	Период занятости, дней	Основная заработная плата, рублей	Дополнительная заработная плата, рублей	Всего рублей
<i>Backend</i> разработчик	1	60000	63	180000	27000	207000
<i>Frontend</i> разработчик	1	50000	63	150000	22500	172500
Тестировщик	1	40000	63	120000	18000	138000
Итого	3	150000	189	450000	67500	517500

Отчисления на страховые взносы производится согласно «Налоговому кодексу Российской Федерации (часть вторая)» от 05.08.2000 № 117-ФЗ (ред. от 26.03.2022).

Действующие тарифы на страховые взносы работодателей установлены статьей 34 НК РФ и составляют: ОПС (обязательное пенсионное страхование) – 22%, ВНиМ (страхование временной нетрудоспособности и в связи с материнством) – 2,9%, ОМС (обязательное медицинское страхование) – 5,1%.

Общая сумма отчислений на страховые взносы составит 155250 рублей.

Общая сумма затрат является суммой: затрат на оплату труда включая налоги и сборы, затраты на закупку оборудования и лицензионного ПО и составляет 1308750 рублей.

					ПГУ 1.090501.09.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

4 Охрана труда

4.1 Анализ негативных факторов, воздействующие на человека

Излишний трафик в Интернете и на компьютере может стать причиной множества здоровья связанных проблем. Продолжительное время, проведенное за компьютером, может негативно отразиться на как физическом, так и психическом состоянии человека. К негативным последствиям относятся неправильная осанка, боли в опорно-двигательном аппарате, ожирение, проблемы со зрением и психические расстройства. Потенциальные угрозы здоровью, связанные с использованием компьютера, требуют серьезного внимания.

Для минимизации негативного влияния компьютера на здоровье, следует придерживаться нескольких принципов. Основные из них включают:

- корректное организованное рабочее пространство;
- соблюдение режима работы и перерывов
- принципы электробезопасности;
- адекватная освещенность рабочего места.

4.1.1 Эргономика рабочего места

Вот ключевые требования для удобного и безопасного рабочего места:

- стул должен иметь возможность поддерживать спину по её изгибу, его высоту можно регулировать таким образом, чтобы ноги покоились на полу или на специальной подставке, а бедра были параллельны полу. Положение подлокотников также должно быть регулируемым, чтобы руки могли свободно на них лежать, а плечи при этом оставались расслабленными;
- мышь компьютера должна находиться в пределах легкого доступа и быть на одном уровне с клавиатурой. При наборе текста или использовании

мышь запястья следует держать в натуральном положении, предплечья - близко к телу, а руки - на уровне локтей или чуть ниже;

- под столом должно быть достаточно места для комфортного размещения коленей, бедер и ног. Если стол не подходит по высоте и его нельзя отрегулировать, следует скорректировать высоту стула;

- монитор должен быть расположен на расстоянии вытянутой руки от пользователя. Верх экрана должен быть на уровне взгляда или немного ниже. Монитор ставится прямо перед клавиатурой. Если используются бифокальные очки, экран следует опустить еще на 1-2 дюйма для более комфортного чтения. Расположите монитор таким образом, чтобы основной источник света был сбоку, чтобы избежать отражения света.

4.1.2 Правила перерывов и режима работы

Сотрудник, работающий на персональном компьютере, должен совершать регулярные перерывы для поддержания эффективности и избегания переутомления. Для этого ему рекомендуется:

- уменьшать продолжительность непрерывной работы на компьютере, если это возможно;

- разделять свою работу на меньшие части и чередовать задания, требующие различных типов движений. Например, можно сменить использование мыши на чтение или поиск информации в интернете;

- делать небольшие перерывы (10-20 секунд) примерно каждые 10 минут, отводя руки от клавиатуры и немного размявшись;

- каждые 30-60 минут делать немного более длинные перерывы (2-5 минут), чтобы немного растянуться или прогуляться.

4.1.3 Электробезопасность

Персональные электронно-вычислительные машины (ПЭВМ), подключенные к сети с напряжением 220 В, представляют потенциальную

					ПГУ 1.090501.09.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

угрозу для человека. Воздействие электрического тока может вызвать серьезные ожоги и нарушить функционирование внутренних органов.

Чтобы избежать таких инцидентов, необходимо соблюдать стандарты безопасности, описанные в ГОСТ 12.2.007-75. Основные меры предосторожности включают:

- соответствие розеток электропитания вилкам устройств;
- запрет на подключение и отключение соединительных кабелей во время работы ПЭВМ;
- обязательная маркировка всех электрических розеток с указанием напряжения;
- нанесение номинального напряжения сети яркой краской, крупными символами (не менее 50 мм в высоту) на стене или щите рядом с розеткой.

При работе с компьютерной техникой каждый сотрудник должен относиться с особой внимательностью и ответственностью к электропроводке и оборудованию, помня о рисках, которые несет нарушение правил электробезопасности.

Необходимо регулярно проверять состояние электропроводки, выключателей и розеток на своем рабочем месте. В случае обнаружения неисправностей следует отключить электрооборудование от сети. Работу можно возобновить только после устранения всех обнаруженных проблем.

4.1.4 Освещенность рабочего места

Как натуральное, так и искусственное (локальное) освещение оказывают значительное влияние на наше здоровье и благосостояние, влияя на нашу умственную активность и эмоциональное состояние. Особенно ощутимо воздействие освещения на зрение рабочих. Нарушения зрения и дискомфортные ощущения могут возникнуть, когда качество освещения затрудняет работникам увидеть то, что необходимо для выполнения их профессиональных обязанностей. Это, в свою очередь, может снизить их

					ПГУ 1.090501.09.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

работоспособность. Основными факторами, которые обычно вызывают визуальный дискомфорт, являются недостаточное или избыточное освещение, теневые эффекты, отражения, блики и мерцание света. Исходя из этого, определим параметры искусственного освещения.

Расчет освещения производится для комнаты: длина (а) - 6 м, ширина (б) - 3 м, высота (h) – 3 м. Расчет производится под типовой растровый светильник с 4-ю линейными люминесцентными лампами (Кл), мощностью 18 Вт каждая (1 лампа даёт световой поток (СПл), равный 900 лк).

Поскольку должность программиста предполагает длительные монотонные операции — со высоким уровнем зрительной работы (различение объектов, размером от 3 до 5 мм), то возьмём за норму – освещённость его рабочего места (Е) в 500 люксов.

Коэффициент запаса обычно равен 1,2 (Кз).

Определяем индекс помещения (Ип), расчет:

$$Ип = S/((h1-h2) * (a+b)) = (6*3)/((3-0,8)*(6+3)) \approx 0,9$$

Для данного индекса помещения коэффициент использования (U) примерно равен 65.

Выполним, расчет количества светильников (Ксв.) для данного помещения [7]:

$$Ксв. = (E*S*100*Kз)/(U*Кл*СПл)$$

Следовательно:

$$Ксв. = (500*18*100*1,2)/(65*4*900) = 5 \text{ светильников}$$

Получается, что для кабинета программиста размером 5 х 4 м достаточно будет установить 5 растровых светильников.

Заключение

В ходе выполнения дипломного проекта была разработана клиент-серверная система для планирования, распределения и управления проектами.

На основании проделанной работы система позволяет:

- создавать/удалять/редактировать проекты;
- создавать/удалять/редактировать задачи для проектов;
- управление всеми процессами с помощью удобного пользовательского интерфейса.

Во время выполнения данной работы были получены навыки разработки системы на языке *Java* с использованием *Spring Boot* и системой управления базой данных *MySql*, а также реализации безопасного *REST API* с использованием *Spring Authorization Server*.

Программа написана с использованием среды программирования *IntelliJ IDEA Ultimate Edition*.

В дальнейшем можно усовершенствовать разработанную систему можно добавить графики выполнения задач и проектов. В добавок помимо ролей добавить команды и распределение задач по командам.

Список использованных источников

1. Ньюмен С. Создание микросервисов - СПб.: «Питер» 2016. - 246 с.
2. baeldung.com: [Электронный ресурс]. URL: <https://www.baeldung.com/> (дата обращения: 23.04.2023)
3. spring.io: [Электронный ресурс]. URL: <https://spring.io/> (дата обращения: 27.04.2023)
4. java.com: [Электронный ресурс]. URL: <https://www.java.com/ru/> (дата обращения: 28.04.2023)
5. maven.apache.org: [Электронный ресурс]. URL: <https://www.maven.apache.org> (дата обращения: 30.04.2023)
6. Стивенс Р. Алгоритмы. Теория и практическое применение // John Wiley & Sons.-2013. - 72 с.
7. 1posvetu.ru: [Электронный ресурс]. URL: <https://1posvetu.ru/montazh-i-nastrojka/raschet-osveshheniya-rabochego-mesta.html> (дата обращения: 4.05.2023)

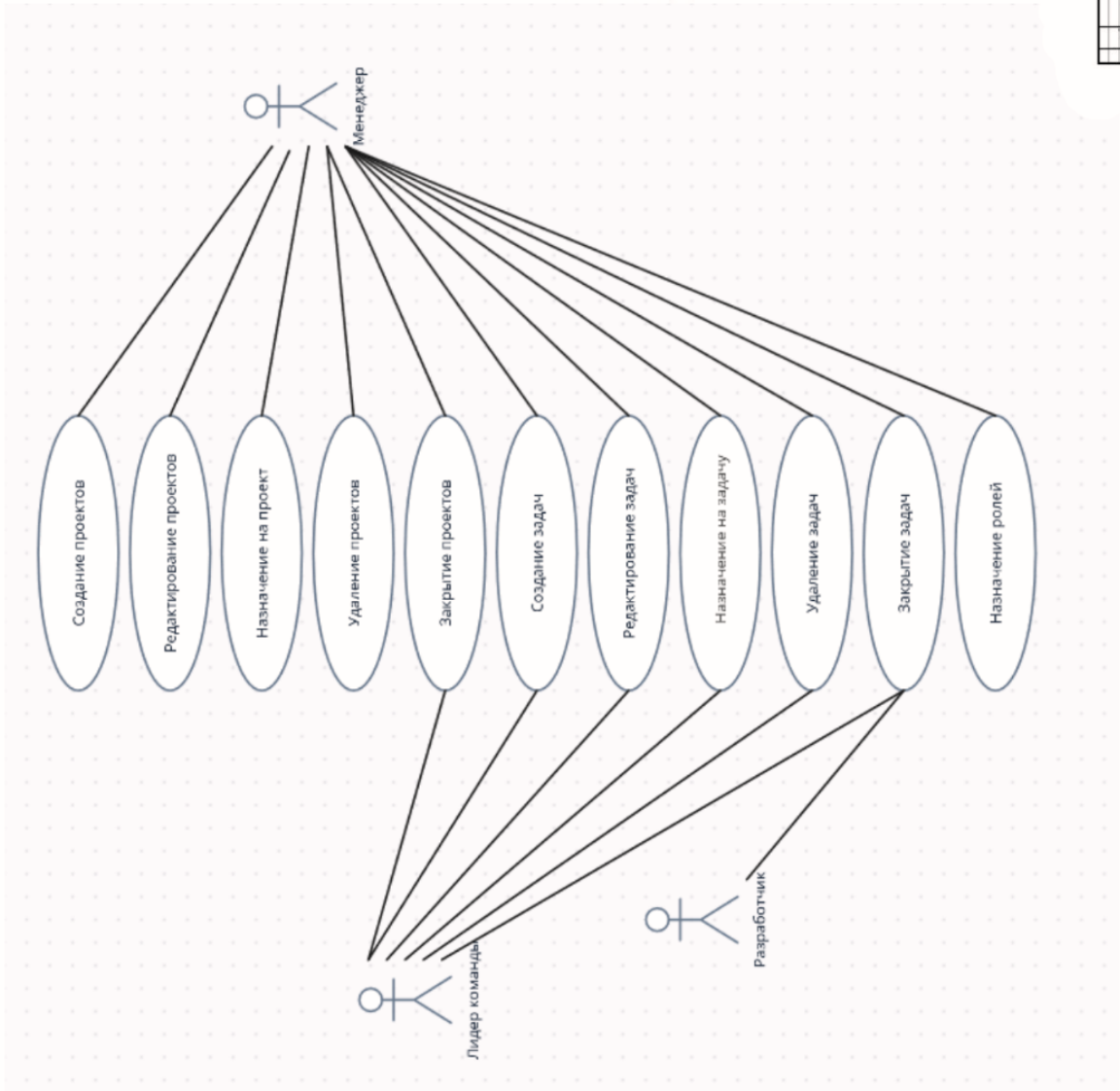
Приложение А

(Обязательное)

Клиент-серверная система для планирования, распределения и управления
проектами
UML-диаграммы

					ПГУ 1.090501.09.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

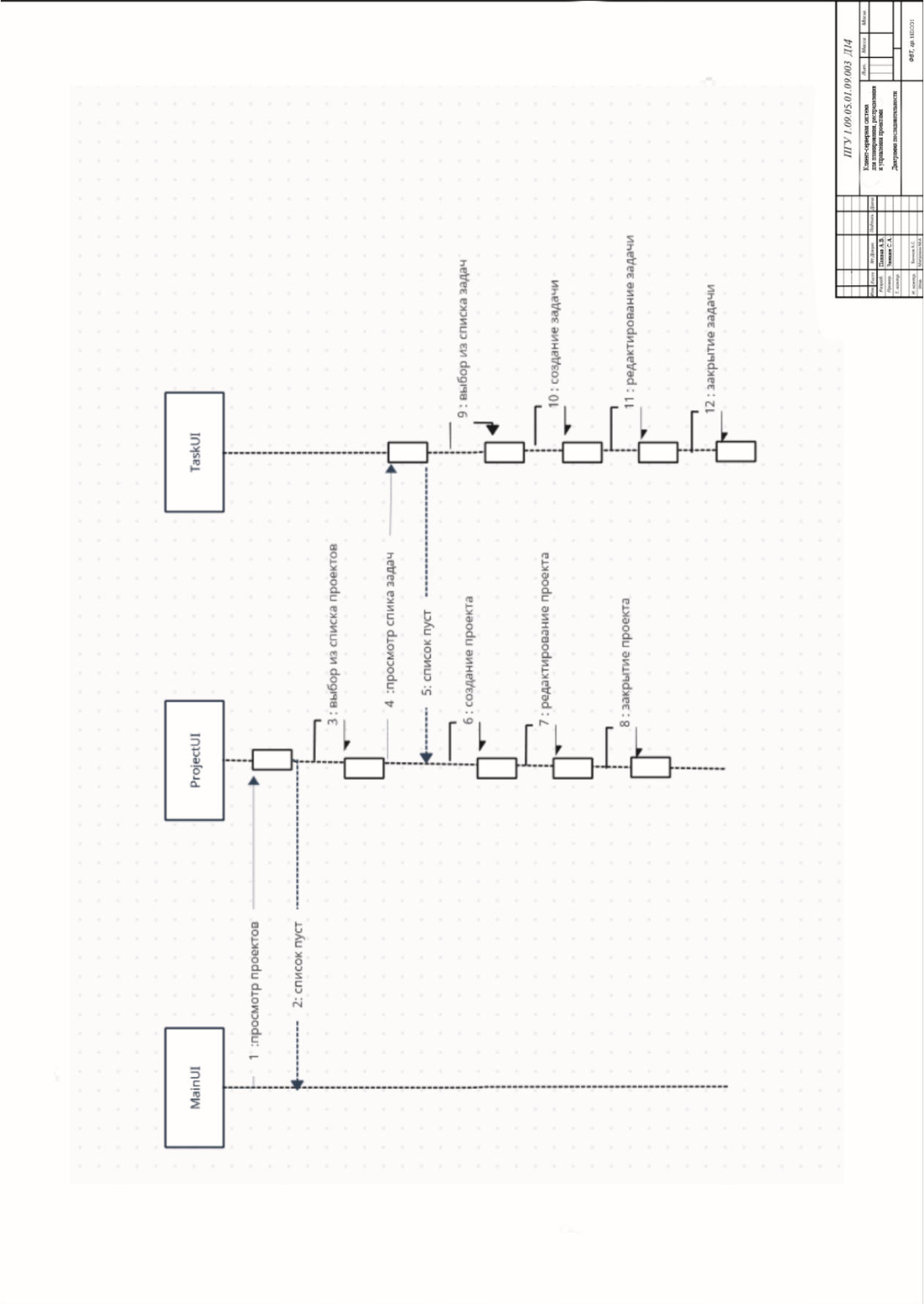
Приложение А.1 – Диаграмма вариантов использования



ПГУ 1.090501.09.001 ПЗ									
Классификация системы									
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация
Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация

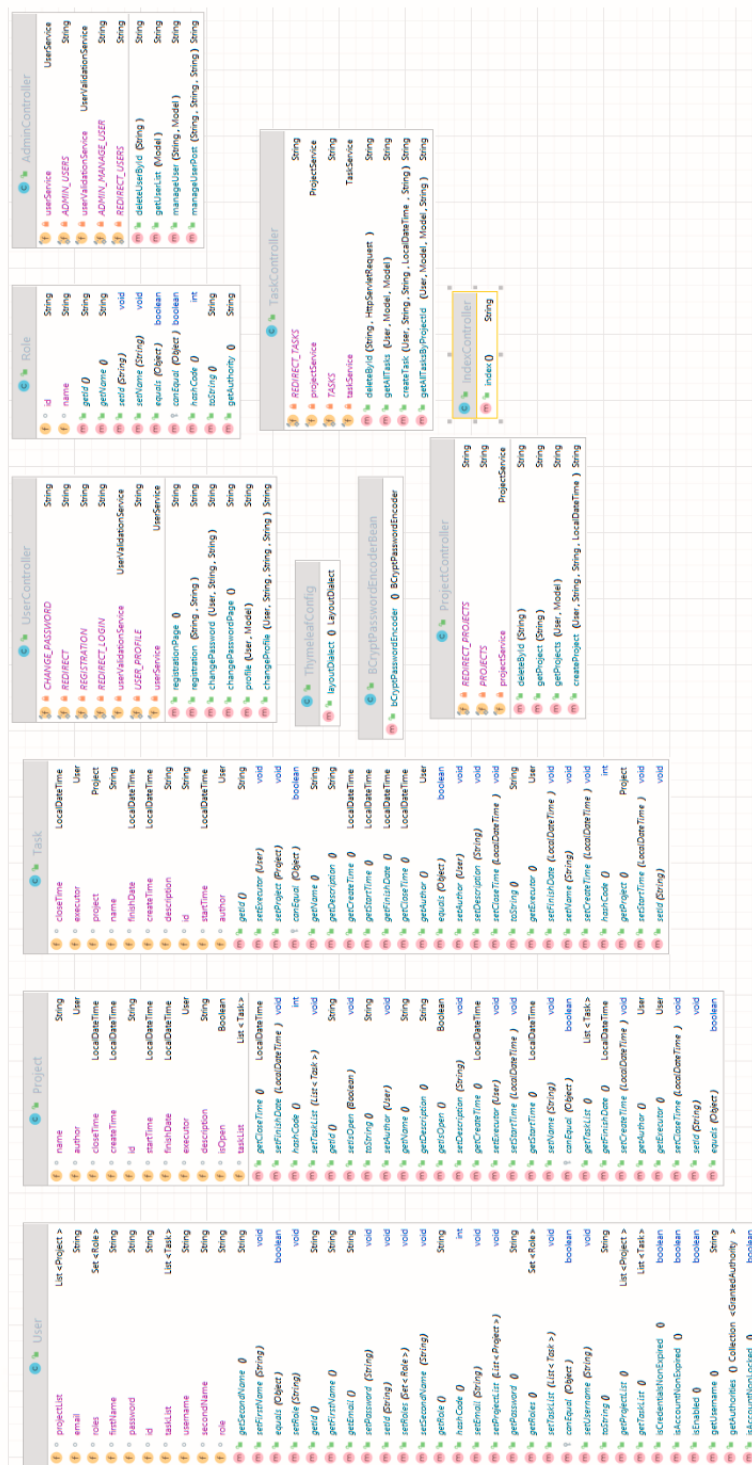
Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.2 – Диаграмма последовательности

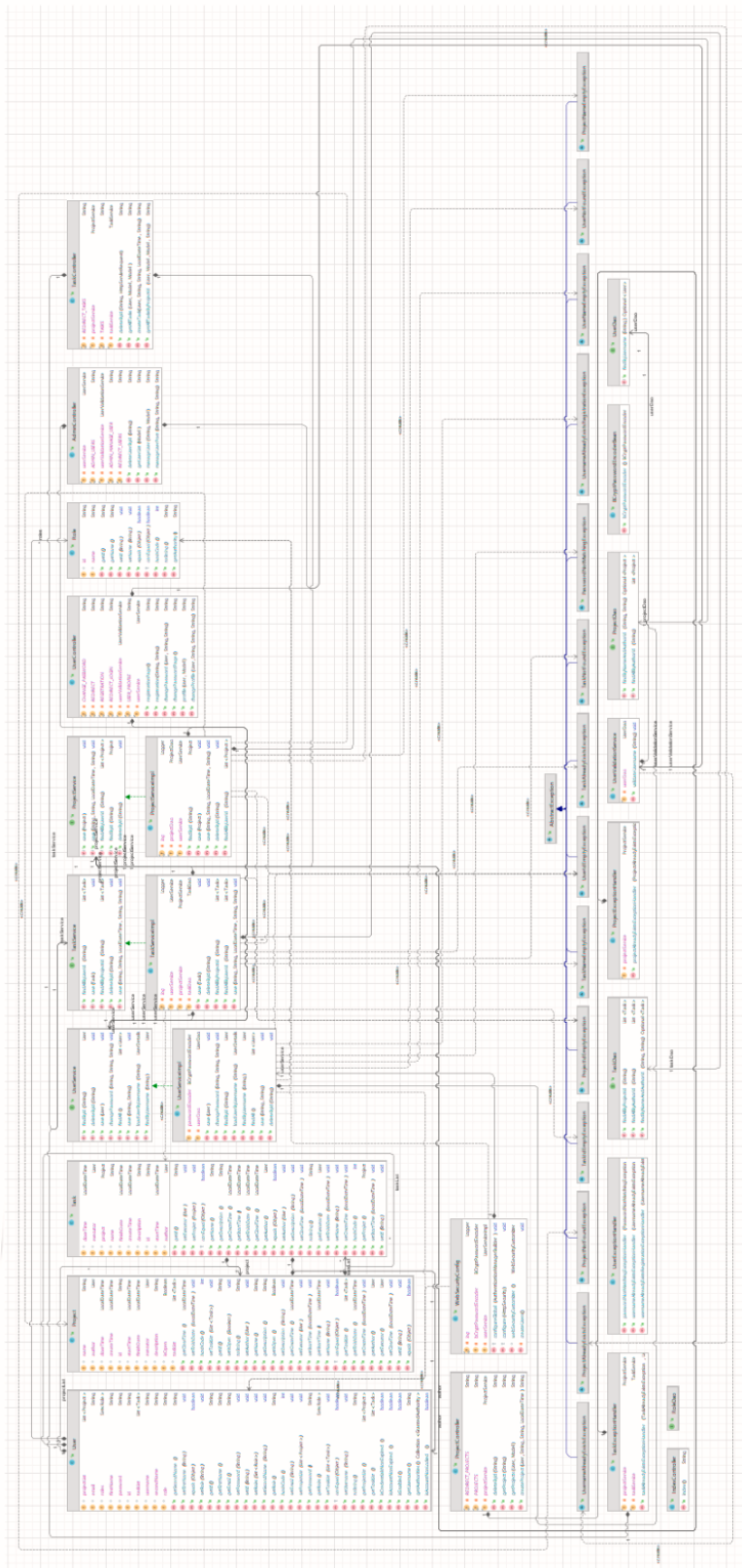


ПГУ 1.090501.09.003 Д14									
Системная схема для планирования, реализации и управления проектами									
Диаграмма последовательности									
Имя	№ документа	Исполнитель	Дата	Место					
Иванов А.А.	ПГУ 1.090501.09.003 Д14	Иванов А.А.	2023	Москва					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					
Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.	Иванов А.А.					

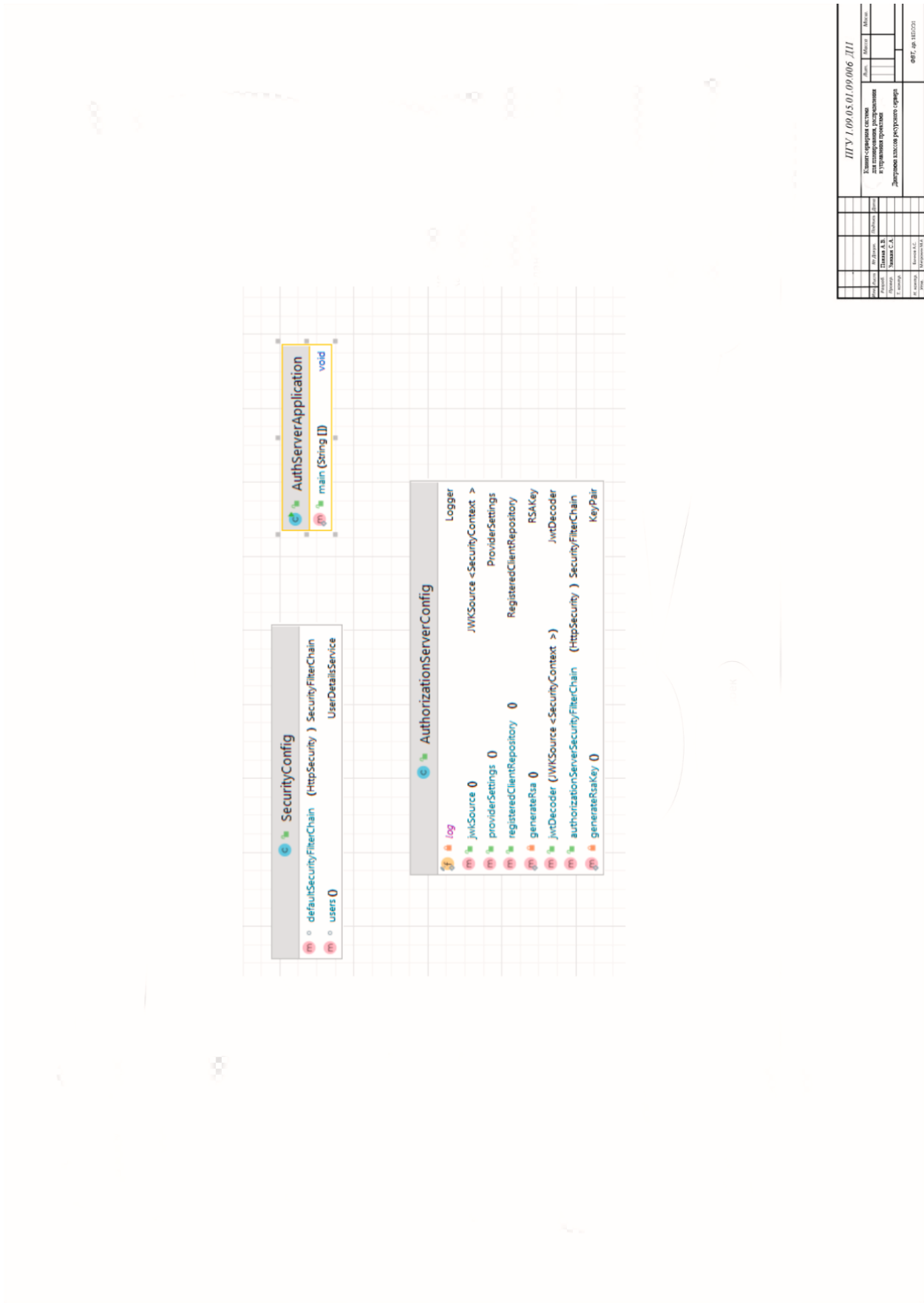
Приложение А.3 – Диаграмма классов клиента

[illegible]

Приложение А.4 – Диаграмма классов ресурсного сервера

[illegible]

Приложение А.5 – Диаграмма классов auth-сервера



Приложение Б

(обязательное)

Клиент-серверная система для планирования, распределения и управления проектами

Листинг программы

					ПГУ 1.090501.09.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

```

package ru.apanyaev.authserver.config;

import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;
import lombok.extern.log4j.Log4j2;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.OAuth2AuthorizationServerConfiguration;
import
org.springframework.security.oauth2.core.AuthorizationGrantType;
import
org.springframework.security.oauth2.core.ClientAuthenticationMet
hod;
import org.springframework.security.oauth2.core.oidc.OidcScopes;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import
org.springframework.security.oauth2.server.authorization.client.
InMemoryRegisteredClientRepository;
import
org.springframework.security.oauth2.server.authorization.client.
RegisteredClient;
import
org.springframework.security.oauth2.server.authorization.client.
RegisteredClientRepository;
import
org.springframework.security.oauth2.server.authorization.config.
ProviderSettings;
import org.springframework.security.web.SecurityFilterChain;

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.UUID;

@Log4j2

```

```

@Configuration
public class AuthorizationServerConfig {

    @Bean
    @Order(Ordered.HIGHEST_PRECEDENCE)
    public SecurityFilterChain
authorizationServerSecurityFilterChain(HttpSecurity http) throws
Exception {

OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http
);
        return
http.formLogin(Customizer.withDefaults()).build();
    }

    @Bean
    public RegisteredClientRepository
registeredClientRepository() {
        var registeredClient =
RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("user-client")
                .clientSecret("{noop}secret")

        .clientAuthenticationMethod(ClientAuthenticationMethod.BASIC)

        .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_COD
E)

        .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)

        .redirectUri("http://127.0.0.1:8080/login/oauth2/code/user-
client-oidc")
                .redirectUri("http://127.0.0.1:8080/authorized")
                .scope(OidcScopes.OPENID)
                .scope("user")
                .build();
        log.debug("client id: {} || client name: {} || client
redir: {} ", registeredClient.getClientId(),
                registeredClient.getClientName(),
                registeredClient.getRedirectUris());
        return new
InMemoryRegisteredClientRepository(registeredClient);
    }

    @Bean
    public JwtDecoder jwtDecoder(JWKSource<SecurityContext>
jwkSource) {
        return

```

```

OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
    }

    @Bean
    public JWKSSource<SecurityContext> jwkSource() throws
NoSuchAlgorithmException {
        var rsaKey = generateRsa();
        var jwkSet = new JWKSet(rsaKey);
        return (jwkSelector, securityContext) ->
jwkSelector.select(jwkSet);
    }

    private static RSAKey generateRsa() throws
NoSuchAlgorithmException {
        var keyPair = generateRsaKey();
        RSAPublicKey publicKey = (RSAPublicKey)
keyPair.getPublic();
        RSAPrivateKey privateKey = (RSAPrivateKey)
keyPair.getPrivate();
        return new RSAKey.Builder(publicKey)
            .privateKey(privateKey)
            .keyID(UUID.randomUUID().toString())
            .build();
    }

    private static KeyPair generateRsaKey() throws
NoSuchAlgorithmException {
        var keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2048);
        return keyPairGenerator.generateKeyPair();
    }

    @Bean
    public ProviderSettings providerSettings() {
        return new
ProviderSettings().issuer("http://localhost:9000");
    }
}

package ru.apanyaev.authserver.config;

import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.web.builders.Http
Security;

```

					ПГУ 1.090501.09.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.provisioning.InMemoryUserDetailsMan
ager;
import org.springframework.security.web.SecurityFilterChain;

import static
org.springframework.security.config.Customizer.withDefaults;

@EnableWebSecurity
public class SecurityConfig {

    @Bean
    SecurityFilterChain defaultSecurityFilterChain(HttpSecurity
http) throws Exception {
        http.authorizeRequests(authorizeRequests ->

authorizeRequests.anyRequest().authenticated()
        )
        .formLogin(withDefaults());
        return http.build();
    }

    @Bean
    UserDetailsService users() {
        UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("pass")
            .roles("ROLE")
            .build();
        return new InMemoryUserDetailsManager(user);
    }
}

server:
    port: 9000
spring:

```

					ПГУ 1.090501.09.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

    application:
        name: auth-server

package ru.apanyaev.server.controller;

import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;
import ru.apanyaev.server.model.Project;
import ru.apanyaev.server.service.ProjectService;

import java.util.List;

@RestController
@RequestMapping("/projects")
@AllArgsConstructor
public class ProjectController {

    private final ProjectService projectService;

    @GetMapping("/find/{projectId}")
    public Project findProjectById(@PathVariable Long projectId)
    {
        return projectService.findProjectById(projectId);
    }

    @GetMapping()
    public List<Project> findProjectAll() {
        return projectService.findAll();
    }

    @PostMapping("/delete")
    public void deleteProjectById(@RequestParam Long projectId)
    {
        projectService.deleteProject(projectId);
    }

    @PostMapping
    public void createProject(@RequestBody Project project) {
        projectService.createProject(project);
    }

}

package ru.apanyaev.server.controller;

import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;
import ru.apanyaev.server.model.Task;
import ru.apanyaev.server.service.TaskService;

```

					ПГУ 1.090501.09.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

```

import java.util.List;

@RestController
@RequestMapping("/tasks")
@AllArgsConstructor
public class TaskController {

    private final TaskService taskService;

    @PostMapping
    public void createTask(@RequestParam Task task) {
        taskService.createTask(task);
    }

    @PostMapping("/delete")
    public void deleteTask(@RequestParam Long taskId) {
        taskService.deleteTask(taskId);
    }

    @GetMapping("/{taskId}")
    public Task findTaskById(@PathVariable Long taskId) {
        return taskService.findTaskById(taskId);
    }

    @GetMapping
    public List<Task> findAll() {
        return taskService.findAll();
    }

    @GetMapping("/projectId={projectId}")
    public List<Task> findAllByProjectId(@PathVariable Long
projectId) {
        return taskService.findAllByProjectId(projectId);
    }

}

package ru.apanyaeв.server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.bu
ilders.AuthenticationManagerBuilder;
import

```

```

org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

import javax.annotation.PostConstruct;
import java.util.Collections;

@EnableWebSecurity
public class SecurityConfig {

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http.csrf()
            .disable()
            .authorizeRequests()
            .antMatchers("/css/**", "/js/**").permitAll()

.antMatchers("/registration").not().fullyAuthenticated()
            .antMatchers("/admin/**").hasRole("MANAGER")
            .antMatchers("/").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .defaultSuccessUrl("/")
            .permitAll()
            .and()
            .logout()
            .permitAll()
            .logoutSuccessUrl("/");
        return http.build();
    }

}

package ru.panyaev.server.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;

@Configuration
public class BCryptPasswordEncoderBean {

```

```

        @Bean
        public BCryptPasswordEncoder bCryptPasswordEncoder() {
            return new BCryptPasswordEncoder();
        }
    }

package ru.panyaev.server.config;

import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.util.ObjectUtils;
import ru.panyaev.server.exception.user.UserNotFoundException;
import ru.panyaev.server.model.Role;
import ru.panyaev.server.model.User;
import ru.panyaev.server.service.impl.UserServiceImpl;

import javax.annotation.PostConstruct;
import java.util.Collections;

@Log4j2
@EnableWebSecurity
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {

    @Bean
    WebSecurityCustomizer webSecurityCustomizer() {
        return web -> web.ignoring().antMatchers("/webjars/**");
    }

```

```

    }

    @Autowired
    private UserServiceImpl userService;

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    protected void configure(HttpSecurity httpSecurity) throws
Exception {
        httpSecurity
            .csrf()
            .disable()
            .authorizeRequests()
            .antMatchers("/css/**", "/js/**").permitAll()

.antMatchers("/registration").not().fullyAuthenticated()
            .antMatchers("/admin/**").hasRole("MANAGER")
            .antMatchers("/").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .defaultSuccessUrl("/")
            .permitAll()
            .and()
            .logout()
            .permitAll()
            .logoutSuccessUrl("/");
    }

    @Autowired
    protected void configureGlobal(AuthenticationManagerBuilder
auth) throws Exception {

auth.userDetailsService(userService).passwordEncoder(bCryptPassw
ordEncoder);
    }

    @PostConstruct
    private void createUsers() {
        try {
            userService.save("user", "user");
        } catch (Exception e) {
            log.warn(e.getMessage());
        }
        try {
            userService.findByUsername("1");

```

```

        } catch (UserNotFoundException e) {
            log.warn(e.getMessage());
            var admin = new User("1", "1");
            admin.setRoles(Collections.singleton(new
Role("ROLE_MANAGER")));

admin.setPassword(bCryptPasswordEncoder.encode("1"));
            admin.setRole("MANAGER");
            admin.setEmail("test@mail.ru");
            userService.save(admin);
        }
        try {
            userService.findByUsername("dev1");
        } catch (UserNotFoundException e) {
            log.warn(e.getMessage());
            var admin = new User("dev1", "1");
            admin.setRoles(Collections.singleton(new
Role("ROLE_DEVELOPER")));

admin.setPassword(bCryptPasswordEncoder.encode("1"));
            admin.setRole("DEVELOPER");
            userService.save(admin);
        }
        try {
            userService.findByUsername("manager1");
        } catch (UserNotFoundException e) {
            log.warn(e.getMessage());
            var admin = new User("manager1", "1");
            admin.setRoles(Collections.singleton(new
Role("ROLE_MANAGER")));

admin.setPassword(bCryptPasswordEncoder.encode("1"));
            admin.setRole("MANAGER");
            userService.save(admin);
        }
        try {
            userService.findByUsername("teamlead1");
        } catch (UserNotFoundException e) {
            log.warn(e.getMessage());
            var admin = new User("teamlead1", "1");
            admin.setRoles(Collections.singleton(new
Role("ROLE_TEAMLEAD")));

admin.setPassword(bCryptPasswordEncoder.encode("1"));
            admin.setRole("TEAMLEAD");
            userService.save(admin);
        }
    }
}

```

```

}

package ru.panyaev.server.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.panyaev.server.model.Project;

import java.util.List;
import java.util.Optional;

@Repository
public interface ProjectDao extends JpaRepository<Project,
String> {

    List<Project> findAllByAuthorId(String userId);

    Optional<Project> findByNameAndAuthorId(String name, String
userId);
}

package ru.panyaev.server.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.panyaev.server.model.Role;

@Repository
public interface RoleDao extends JpaRepository<Role, String> {
}

package ru.panyaev.server.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.panyaev.server.model.Task;

import java.util.List;
import java.util.Optional;

@Repository
public interface TaskDao extends JpaRepository<Task, String> {

    List<Task> findAllByProjectId(String projectId);

    List<Task> findAllByAuthorId(String userId);

    Optional<Task> findByNameAndAuthorId(String name, String

```

```

userId);

}

package ru.panyaev.server.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.panyaev.server.model.User;

import java.util.Optional;

@Repository
public interface UserDao extends JpaRepository<User, String> {

    Optional<User> findByUsername(String username);

}

package ru.panyaev.server.exception.handler;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import
ru.panyaev.server.exception.project.ProjectAlreadyExistsExceptio
n;
import ru.panyaev.server.model.User;
import ru.panyaev.server.service.ProjectService;

@ControllerAdvice
public class ProjectExceptionHandler {

    private final ProjectService projectService;

    @Autowired
    public ProjectExceptionHandler(ProjectService
projectService) {
        this.projectService = projectService;
    }

    @ExceptionHandler({ProjectAlreadyExistsException.class})
    public String projectAlreadyExistsExceptionHandler(
        ProjectAlreadyExistsException ex,
        @AuthenticationPrincipal User user,

```

```

        Model modelProject,
        Model modelError
    ) {
        modelProject.addAttribute("projects",
projectService.findAllByUserId(user.getId()));
        modelError.addAttribute("error", ex.getMessage());
        return "projects";
    }

}

package ru.panyaev.server.exception.handler;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import
ru.panyaev.server.exception.task.TaskAlreadyExistsException;
import ru.panyaev.server.model.User;
import ru.panyaev.server.service.ProjectService;
import ru.panyaev.server.service.TaskService;

@ControllerAdvice
public class TaskExceptionHandler {

    private final ProjectService projectService;
    private final TaskService taskService;

    @Autowired
    public TaskExceptionHandler(ProjectService projectService,
TaskService taskService) {
        this.projectService = projectService;
        this.taskService = taskService;
    }

    @ExceptionHandler({TaskAlreadyExistsException.class})
    public String taskAlreadyExistsExceptionHandler(
        TaskAlreadyExistsException ex,
        @AuthenticationPrincipal User user,
        Model modelTask,
        Model modelProject,
        Model modelError
    ) {
        modelTask.addAttribute("tasks",

```

```

taskService.findAllById(user.getId()));
        modelProject.addAttribute("projects",
projectService.findAllById(user.getId()));
        modelError.addAttribute("error", ex.getMessage());
        return "tasks";
    }

}

package ru.panyaev.server.exception.handler;

import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import
ru.panyaev.server.exception.user.PasswordNotMatchingException;
import
ru.panyaev.server.exception.user.UsernameAlreadyExistsException;
import
ru.panyaev.server.exception.user.UsernameAlreadyExistsRegistrationException;
import ru.panyaev.server.model.User;

@ControllerAdvice
public class UserExceptionHandler {

    @ExceptionHandler({UsernameAlreadyExistsException.class})
    public String usernameAlreadyExistsExceptionHandler(
        UsernameAlreadyExistsException ex,
        @AuthenticationPrincipal User user,
        Model modelUser,
        Model modelError
    ) {
        modelUser.addAttribute("user", user);
        modelError.addAttribute("error", ex.getMessage());
        return "user-profile";
    }

    @ExceptionHandler({UsernameAlreadyExistsRegistrationException.class})
    public String
usernameAlreadyExistsRegistrationExceptionHandler(
        UsernameAlreadyExistsRegistrationException ex,
        Model modelError

```

```

    ) {
        modelError.addAttribute("error", ex.getMessage());
        return "registration";
    }

    @ExceptionHandler({PasswordNotMatchingException.class})
    public String passwordNotMatchingExceptionHandler(
        PasswordNotMatchingException ex,
        Model modelError
    ) {
        modelError.addAttribute("error", ex.getMessage());
        return "change-password";
    }
}

package ru.panyaev.server.exception.project;

import ru.panyaev.server.exception.AbstractException;

public class ProjectAlreadyExistsException extends
AbstractException {

    public ProjectAlreadyExistsException() {
        super("Project already exists!");
    }

}

package ru.panyaev.server.exception.project;

import ru.panyaev.server.exception.AbstractException;

public class ProjectIdEmptyException extends AbstractException {

    public ProjectIdEmptyException() {
        super("Project id is empty!");
    }

}

package ru.panyaev.server.exception.project;

import ru.panyaev.server.exception.AbstractException;

public class ProjectNameEmptyException extends AbstractException
{

    public ProjectNameEmptyException() {

```

```

        super("Project name is empty!");
    }

}

package ru.panyaev.server.exception.project;

import ru.panyaev.server.exception.AbstractException;

public class ProjectNotFoundException extends AbstractException
{

    public ProjectNotFoundException() {
        super("Project not found!");
    }

}

package ru.panyaev.server.exception.task;

import ru.panyaev.server.exception.AbstractException;

public class TaskAlreadyExistsException extends
AbstractException {

    public TaskAlreadyExistsException() {
        super("Task already exists!");
    }

}

package ru.panyaev.server.exception.task;

import ru.panyaev.server.exception.AbstractException;

public class TaskIdEmptyException extends AbstractException {

    public TaskIdEmptyException() {
        super("Task id is empty!");
    }

}

package ru.panyaev.server.exception.task;

import ru.panyaev.server.exception.AbstractException;

public class TaskNameEmptyException extends AbstractException {

    public TaskNameEmptyException() {

```

```

        super("Task name is empty!");
    }

}

package ru.panyaev.server.exception.task;

import ru.panyaev.server.exception.AbstractException;

public class TaskNotFoundException extends AbstractException {

    public TaskNotFoundException() {
        super("Task not found!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class PasswordNotMatchingException extends
AbstractException {

    public PasswordNotMatchingException() {
        super("You must provide a valid current password!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class UserIdEmptyException extends AbstractException {

    public UserIdEmptyException() {
        super("User id is empty!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class UsernameAlreadyExistsException extends
AbstractException {

    public UsernameAlreadyExistsException() {

```

```

        super("Username already exists!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class UsernameAlreadyExistsRegistrationException extends
AbstractException {

    public UsernameAlreadyExistsRegistrationException() {
        super("Username already exists!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class UserNameEmptyException extends AbstractException {

    public UserNameEmptyException() {
        super("User name is empty!");
    }

}

package ru.panyaev.server.exception.user;

import ru.panyaev.server.exception.AbstractException;

public class UserNotFoundException extends AbstractException {

    public UserNotFoundException() {
        super("User not found!");
    }

}

package ru.panyaev.server.exception;

public abstract class AbstractException extends RuntimeException
{

    protected AbstractException() {
    }

}

```

```

        protected AbstractException(Throwable cause) {
            super(cause);
        }

        protected AbstractException(String message) {
            super(message);
        }
    }

package ru.panyaev.server.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.List;
import java.util.UUID;

@Data
@Entity
@Table(name = "t_project")
@NoArgsConstructor
public class Project {

    @Id
    String id = "id" + UUID.randomUUID();
    String name;
    String description;
    Boolean isOpen;
    LocalDateTime createTime = LocalDateTime.now();
    LocalDateTime startTime;
    LocalDateTime finishDate;
    LocalDateTime closeTime;
    @ManyToOne
    User author;
    @ManyToOne
    User executor;
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL,
        orphanRemoval = true)
    List<Task> taskList;

}

package ru.panyaev.server.model;

import lombok.Data;
import lombok.NoArgsConstructor;

```

```

import org.springframework.security.core.GrantedAuthority;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.UUID;

@Data
@Entity
@Table(name = "t_role")
@NoArgsConstructor
public class Role implements GrantedAuthority {

    @Id
    String id = "id" + UUID.randomUUID();
    String name;

    public Role(String name) {
        this.name = name;
    }

    @Override
    public String getAuthority() {
        return getName();
    }

}

package ru.panyaev.server.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import java.time.LocalDateTime;
import java.util.UUID;

@Data
@Entity
@Table(name = "t_task")
@NoArgsConstructor
public class Task {

    @Id
    String id = "id" + UUID.randomUUID();

```

```

        String name;
        String description;
        LocalDateTime createTime = LocalDateTime.now();
        LocalDateTime startTime;
        LocalDateTime finishDate;
        LocalDateTime closeTime;
        @ManyToOne
        User author;
        @ManyToOne
        User executor;
        @ManyToOne
        Project project;
    }

package ru.panyaev.server.model;

import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.util.Collection;
import java.util.List;
import java.util.Set;
import java.util.UUID;

@Data
@Entity
@Table(name = "t_user")
@NoArgsConstructor
public class User implements UserDetails {

    @Id
    String id = "id" + UUID.randomUUID();
    String username;
    String firstName;
    String secondName;
    String email;
    String role;
    String password;
    @ManyToMany(fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
    Set<Role> roles;
    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL,
orphanRemoval = true)

```

```

        List<Project> projectList;
        @OneToMany(mappedBy = "author", cascade = CascadeType.ALL,
orphanRemoval = true)
        List<Task> taskList;

        public User(String username, String password) {
            this.username = username;
            this.password = password;
        }

        @Override
        public String getUsername() {
            return username;
        }

        @Override
        public boolean isAccountNonExpired() {
            return true;
        }

        @Override
        public boolean isAccountNonLocked() {
            return true;
        }

        @Override
        public boolean isCredentialsNonExpired() {
            return true;
        }

        @Override
        public boolean isEnabled() {
            return true;
        }

        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
            return getRoles();
        }

    }

    package ru.panyaev.server.service.impl;

    import lombok.extern.log4j.Log4j2;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;

```

```

import ru.panyaev.server.dao.ProjectDao;
import
ru.panyaev.server.exception.project.ProjectAlreadyExistsException;
import
ru.panyaev.server.exception.project.ProjectIdEmptyException;
import
ru.panyaev.server.exception.project.ProjectNameEmptyException;
import
ru.panyaev.server.exception.project.ProjectNotFoundException;
import ru.panyaev.server.exception.user.UserIdEmptyException;
import ru.panyaev.server.model.Project;
import ru.panyaev.server.service.UserService;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

@Log4j2
@Service
public class ProjectServiceImpl implements
ru.panyaev.server.service.ProjectService {

    private final ProjectDao projectDao;
    private final UserService userService;

    @Autowired
    public ProjectServiceImpl(ProjectDao projectDao, UserService
userService) {
        this.projectDao = projectDao;
        this.userService = userService;
    }

    @Override
    public void save(Project project) {
        if (project == null) throw new
ProjectNotFoundException();
        projectDao.save(project);
    }

    @Override
    public void save(String name, String description,
LocalDateTime finishDate, String userId) {
        if (name == null || name.isEmpty()) throw new
ProjectNameEmptyException();
        if (projectDao.findByNameAndAuthorId(name,
userId).isPresent()) throw new ProjectAlreadyExistsException();
        if (userId == null) throw new UserIdEmptyException();
    }

```

```

        var project = new Project();
        project.setAuthor(userService.findById(userId));
        project.setName(name);
        project.setStartTime(LocalDateDateTime.now());
        project.setExecutor(userService.findByUsername("1"));
        project.setIsOpen(true);
        if (description != null && !description.isEmpty())
project.setDescription(description);
        if (finishDate != null)
project.setFinishDate(finishDate);
        projectDao.save(project);
        log.info("Project created: name: {}, description: {},
finishDate: {}, userId: {}", name, description,
        finishDate, userId);
    }

    @Override
    public Project findById(String id) {
        if (id == null) throw new ProjectIdEmptyException();
        final Optional<Project> project =
projectDao.findById(id);
        if (project.isEmpty()) throw new
ProjectNotFoundException();
        return project.get();
    }

    @Override
    public void deleteById(String id) {
        if (id == null) throw new ProjectIdEmptyException();
        projectDao.deleteById(id);
    }

    @Override
    public List<Project> findAllByUserId(String userId) {
        if (userId == null) throw new UserIdEmptyException();
        List<Project> allByAuthorId =
projectDao.findAllByAuthorId(userId);
        return allByAuthorId;
    }
}

package ru.panyaev.server.service.impl;

import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```

import ru.panyaev.server.dao.TaskDao;
import
ru.panyaev.server.exception.project.ProjectIdEmptyException;
import
ru.panyaev.server.exception.task.TaskAlreadyExistsException;
import ru.panyaev.server.exception.task.TaskIdEmptyException;
import ru.panyaev.server.exception.task.TaskNameEmptyException;
import ru.panyaev.server.exception.task.TaskNotFoundException;
import ru.panyaev.server.exception.user.UserIdEmptyException;
import ru.panyaev.server.model.Task;
import ru.panyaev.server.service.ProjectService;
import ru.panyaev.server.service.UserService;

import java.time.LocalDateTime;
import java.util.List;

@Log4j2
@Service
public class TaskServiceImpl implements
ru.panyaev.server.service.TaskService {

    private final TaskDao taskDao;
    private final ProjectService projectService;
    private final UserService userService;

    @Autowired
    public TaskServiceImpl(TaskDao taskDao, ProjectService
projectService, UserService userService) {
        this.taskDao = taskDao;
        this.projectService = projectService;
        this.userService = userService;
    }

    @Override
    public void save(Task task) {
        if (task == null) throw new TaskNotFoundException();
        taskDao.save(task);
    }

    @Override
    public void save(String name, String description,
LocalDateTime finishDate, String userId, String projectId) {
        if (name == null || name.isEmpty()) throw new
TaskNameEmptyException();
        if (taskDao.findByNameAndAuthorId(name,
userId).isPresent()) throw new TaskAlreadyExistsException();
        if (userId == null) throw new UserIdEmptyException();
        if (projectId == null) throw new

```

```

ProjectIdEmptyException();
        var task = new Task();
        task.setName(name);
        if (description != null && !description.isEmpty())
task.setDescription(description);
        if (finishDate != null) task.setFinishDate(finishDate);
        task.setAuthor(userService.findById(userId));
        task.setProject(projectService.findById(projectId));
        taskDao.save(task);
        log.info("Task created: name: {}, description: {},
finishDate: {}, userId: {}, projectId: {}", name,
            description, finishDate, userId, projectId);
    }

    @Override
    public List<Task> findAllByProjectId(String projectId) {
        if (projectId == null) throw new
ProjectIdEmptyException();
        return taskDao.findAllByProjectId(projectId);
    }

    @Override
    public List<Task> findAllByUserId(String userId) {
        if (userId == null) throw new UserIdEmptyException();
        return taskDao.findAllByAuthorId(userId);
    }

    @Override
    public void deleteById(String id) {
        if (id == null) throw new TaskIdEmptyException();
        taskDao.deleteById(id);
    }
}

package ru.panyaev.server.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService
;
import
org.springframework.security.core.userdetails.UsernameNotFoundEx
ception;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder

```

```

;
import org.springframework.stereotype.Service;
import ru.panyaev.server.dao.UserDao;
import ru.panyaev.server.exception.user.*;
import ru.panyaev.server.model.Role;
import ru.panyaev.server.model.User;
import ru.panyaev.server.service.UserService;

import java.util.Collections;
import java.util.List;
import java.util.Optional;

@Service
public class UserServiceImpl implements UserDetailsService,
UserService {

    private final UserDao userDao;
    private final BCryptPasswordEncoder passwordEncoder;

    @Autowired
    public UserServiceImpl(UserDao userDao,
BCryptPasswordEncoder passwordEncoder) {
        this.userDao = userDao;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
        var user = userDao.findByUsername(username);
        if (user.isEmpty()) {
            throw new UsernameNotFoundException("User not
found");
        }
        return user.get();
    }

    @Override
    public void save(User user) {
        if (user == null) return;
        userDao.save(user);
    }

    @Override
    public void save(String username, String password) {
        if (username == null || username.isEmpty()) return;
        if (password == null || password.isEmpty()) return;
        if (userDao.findByUsername(username).isPresent()) throw

```

```

new UsernameAlreadyExistsRegistrationException();
    var user = new User(username, password);
    user.setRoles(Collections.singleton(new
Role("ROLE_USER")));

user.setPassword(passwordEncoder.encode(user.getPassword()));
    userDao.save(user);
}

@Override
public List<User> findAll() {
    return userDao.findAll();
}

@Override
public User findByUsername(String username) {
    if (username == null || username.isEmpty()) throw new
UserNameEmptyException();
    var user = userDao.findByUsername(username);
    if (user.isEmpty()) throw new UserNotFoundException();
    return user.get();
}

@Override
public User findById(String userId) {
    if (userId == null) throw new UserIdEmptyException();
    Optional<User> userFromDb = userDao.findById(userId);
    if (userFromDb.isEmpty()) throw new
UserNotFoundException();
    return userFromDb.get();
}

@Override
public void deleteById(String id) {
    if (id == null) return;
    userDao.deleteById(id);
}

@Override
public void changePassword(String id, String oldPassword,
String newPassword) {
    if (id == null || id.isEmpty()) throw new
UserIdEmptyException();
    Optional<User> user = userDao.findById(id);
    if (user.isEmpty()) throw new UserNotFoundException();
    if (!passwordEncoder.matches(oldPassword,
user.get().getPassword())) throw new
PasswordNotMatchingException();

```

```

user.get().setPassword(passwordEncoder.encode(newPassword));
    userDao.save(user.get());
}
}

package ru.panyaev.server.service.validation;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.panyaev.server.dao.UserDao;
import
ru.panyaev.server.exception.user.UsernameAlreadyExistsException;

@Service
public class UserValidationService {

    private final UserDao userDao;

    @Autowired
    public UserValidationService(UserDao userDao) {
        this.userDao = userDao;
    }

    public void validateUsername(String username) {
        if (userDao.findByUsername(username).isPresent()) throw
new UsernameAlreadyExistsException();
    }

}

package ru.panyaev.server.service;

import ru.panyaev.server.model.Project;

import java.time.LocalDateTime;
import java.util.Date;
import java.util.List;

public interface ProjectService {
    void save(Project project);

    void save(String name, String description, LocalDateTime
finishDate, String userId);

    Project findById(String id);

    void deleteById(String id);
}

```

```

        List<Project> findAllByUserId(String userId);
    }

package ru.panyaev.server.service;

import ru.panyaev.server.model.Task;

import java.time.LocalDateTime;
import java.util.Date;
import java.util.List;

public interface TaskService {
    void save(Task task);

    void save(String name, String description, LocalDateTime
finishDate, String userId, String projectId);

    List<Task> findAllByProjectId(String projectId);

    List<Task> findAllByUserId(String userId);

    void deleteById(String id);
}

package ru.panyaev.server.service;

import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import ru.panyaev.server.model.User;

import java.util.List;

public interface UserService {
    UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException;

    void save(User user);

    void save(String username, String password);

    List<User> findAll();

    User findByUsername(String username);

    User findById(String userId);
}

```

```

        void deleteById(String id);

        void changePassword(String id, String oldPassword, String
newPassword);
    }

server:
    port: 8090

spring:
    security:
        oauth2:
            resourceserver:
                jwt:
                    issuer-uri: http://localhost:9000
    datasource:
        url: jdbc:postgresql://localhost:5432/postgres
        username: postgres
        password: passWord
    jpa:
        hibernate:
            ddl-auto: update

package ru.panyaev.client.config;

import nz.net.ultraq.thymeleaf.LayoutDialect;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ThymeleafConfig {

    @Bean
    public LayoutDialect layoutDialect() {
        return new LayoutDialect();
    }

}

package ru.panyaev.client.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import ru.panyaev.client.service.UserService;

```

```

import
ru.panyaev.client.service.validation.UserValidationService;

@Controller
@RequestMapping("/admin")
public class AdminController {

    private static final String ADMIN_USERS = "admin-users";
    private static final String ADMIN_MANAGE_USER = "admin-
manage-user";
    private static final String REDIRECT_USERS =
"redirect:/admin/users";

    private final UserService userService;
    private final UserValidationService userValidationService;

    @Autowired
    public AdminController(UserService userService,
UserValidationService userValidationService) {
        this.userService = userService;
        this.userValidationService = userValidationService;
    }

    @GetMapping("/users")
    public String getUserList(Model model) {
        model.addAttribute("users", userService.findAll());
        return ADMIN_USERS;
    }

    @GetMapping("/users/delete/{id}")
    public String deleteUserById(@PathVariable String id) {
        userService.deleteById(id);
        return REDIRECT_USERS;
    }

    @GetMapping("/users/manage/{id}")
    public String manageUser(
        @PathVariable String id,
        Model model
    ) {
        model.addAttribute("user", userService.findById(id));
        return ADMIN_MANAGE_USER;
    }

    @PostMapping("/users/manage")
    public String manageUserPost(
        @RequestParam(required = false) String email,
        @RequestParam String username,

```

```

        @RequestParam String userId
    ) {
        var user = userService.findById(userId);
        if (!user.getUsername().equals(username))
            userService.validateUsername(username);
        user.setUsername(username);
        user.setEmail(email);
        userService.save(user);
        return REDIRECT_USERS;
    }

}

package ru.panyaev.client.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class IndexController {

    @GetMapping
    public String index() {
        return "index";
    }

}

package ru.panyaev.client.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.format.annotation.DateTimeFormat;
import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import ru.panyaev.client.model.User;
import ru.panyaev.client.service.ProjectService;

import java.time.LocalDateTime;
import java.util.Date;

@Controller
@RequestMapping("/projects")
public class ProjectController {

    private static final String PROJECTS = "projects";

```

```

    private static final String REDIRECT_PROJECTS =
"redirect:/projects";

    private final ProjectService projectService;

    @Autowired
    public ProjectController(ProjectService projectService) {
        this.projectService = projectService;
    }

    @GetMapping
    public String getProjects(
        @AuthenticationPrincipal User user,
        Model model
    ) {
        model.addAttribute("projects",
projectService.findAllByUserId(user.getId()));
        return PROJECTS;
    }

    @PostMapping
    public String createProject(
        @AuthenticationPrincipal User user,
        @RequestParam String name,
        @RequestParam(required = false) String description,
        @RequestParam(required = false) @DateTimeFormat(iso
= DateTimeFormat.ISO.DATE_TIME) LocalDateTime finishDate
    ) {
        projectService.save(name, description, finishDate,
user.getId());
        return REDIRECT_PROJECTS;
    }

    @GetMapping("/delete/{id}")
    public String deleteById(@PathVariable("id") String id) {
        projectService.deleteById(id);
        return REDIRECT_PROJECTS;
    }

    @GetMapping("/{id}")
    public String getProject(@PathVariable("id") String id) {
        return null;
    }
}

package ru.panyaev.client.controller;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.format.annotation.DateTimeFormat;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import ru.panyaev.client.model.User;
import ru.panyaev.client.service.ProjectService;
import ru.panyaev.client.service.TaskService;

import javax.servlet.http.HttpServletRequest;
import java.time.LocalDateTime;
import java.util.Date;

@Controller
@RequestMapping ("/tasks")
public class TaskController {

    private static final String TASKS = "tasks";
    private static final String REDIRECT_TASKS =
"redirect:/tasks";

    private final TaskService taskService;
    private final ProjectService projectService;

    @Autowired
    public TaskController(TaskService taskService,
ProjectService projectService) {
        this.taskService = taskService;
        this.projectService = projectService;
    }

    @GetMapping
    public String getAllTasks(
        @AuthenticationPrincipal User user,
        Model modelTask,
        Model modelProject
    ) {
        modelTask.addAttribute("tasks",
taskService.findAllByUserId(user.getId()));
        modelProject.addAttribute("projects",
projectService.findAllByUserId(user.getId()));
        return TASKS;
    }

    @GetMapping("/{id}")

```

```

        public String getAllTasksByProjectId(
            @AuthenticationPrincipal User user,
            Model modelTask,
            Model modelProject,
            @PathVariable("id") String id
        ) {
            modelTask.addAttribute("tasks",
taskService.findAllByProjectId(id));
            modelProject.addAttribute("projects",
projectService.findAllByUserId(user.getId()));
            return TASKS;
        }

        @PostMapping(value = {"", "/*"})
        public String createTask(
            @AuthenticationPrincipal User user,
            @RequestParam String name,
            @RequestParam(required = false) String description,
            @RequestParam(required = false) @DateTimeFormat(iso
= DateTimeFormat.ISO.DATE_TIME) LocalDateTime finishDate,
            @RequestParam String projectId
        ) {
            taskService.save(name, description, finishDate,
user.getId(), projectId);
            return REDIRECT_TASKS;
        }

        @GetMapping("/delete/{id}")
        public String deleteById(@PathVariable("id") String id,
HttpServletRequest request) {
            taskService.deleteById(id);
            String referer = request.getHeader("Referer");
            return "redirect:"+ referer;
        }
    }

    package ru.panyaev.client.controller;

    import org.springframework.beans.factory.annotation.Autowired;
    import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
    import org.springframework.stereotype.Controller;
    import org.springframework.ui.Model;
    import org.springframework.web.bind.annotation.GetMapping;
    import org.springframework.web.bind.annotation.ModelAttribute;
    import org.springframework.web.bind.annotation.PostMapping;

```

```

import org.springframework.web.bind.annotation.RequestParam;
import ru.panyaev.client.model.User;
import ru.panyaev.client.service.UserService;
import
ru.panyaev.client.service.validation.UserValidationService;

@Controller
public class UserController {

    private static final String REDIRECT_LOGIN =
"redirect:/login";
    private static final String REDIRECT = "redirect:";
    private static final String USER_PROFILE = "user-profile";
    private static final String REGISTRATION = "registration";
    private static final String CHANGE_PASSWORD = "change-
password";

    private final UserService userService;
    private final UserValidationService userValidationService;

    @Autowired
    public UserController(UserService userService,
UserValidationService userValidationService) {
        this.userService = userService;
        this.userValidationService = userValidationService;
    }

    @GetMapping("/registration")
    public String registrationPage() {
        return REGISTRATION;
    }

    @PostMapping("/registration")
    public String registration(
        @ModelAttribute("username") String username,
        @ModelAttribute("password") String password
    ) {
        userService.save(username, password);
        return REDIRECT_LOGIN;
    }

    @GetMapping("/profile")
    public String profile(
        @AuthenticationPrincipal User user,
        Model model
    ) {
        model.addAttribute("user", user);
        return USER_PROFILE;
    }

```

```

    }

    @PostMapping("/profile")
    public String changeProfile(
        @AuthenticationPrincipal User user,
        @RequestParam(required = false) String email,
        @RequestParam String username,
        @RequestParam String userId
    ) {
        if
(!userService.findById(userId).getUsername().equals(username))
        userService.validateUsername(username);
        user.setEmail(email);
        user.setUsername(username);
        userService.save(user);
        return REDIRECT;
    }

    @GetMapping("/change-password")
    public String changePasswordPage() {
        return CHANGE_PASSWORD;
    }

    @PostMapping("/change-password")
    public String changePassword(
        @AuthenticationPrincipal User user,
        @RequestParam String currentPassword,
        @RequestParam String newPassword
    ) {
        userService.changePassword(user.getId(),
currentPassword, newPassword);
        return REDIRECT;
    }
}

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <!-- The above 3 meta tags *must* come first in the head;
any other head content must come *after* these tags -->

```

```

<title>Page not found!</title>

<!-- Google font -->
<link
href="https://fonts.googleapis.com/css?family=Cabin:400,700"
rel="stylesheet">
<link
href="https://fonts.googleapis.com/css?family=Montserrat:900"
rel="stylesheet">

<!-- Custom stylesheet -->
<link rel="stylesheet" type="text/css"
th:href="@{/css/notFound404.css}">

<!-- HTML5 shim and Respond.js for IE8 support of HTML5
elements and media queries -->
<!-- WARNING: Respond.js doesn't work if you view the page
via file:// -->
<!--[if lt IE 9]>
<script
src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></
script>
<script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></scri
pt>
<![endif]-->

</head>

<body>
<main layout:fragment="content">
  <div id="notfound">
    <div class="notfound">
      <div class="notfound-404">
        <h3>Oops! Page not found</h3>

<h1><span>4</span><span>0</span><span>4</span></h1>
      </div>
      <h2>we are sorry, but the page you requested was not
found</h2>
    </div>
  </div>
</main>
</body><!-- This templates was made by Colorlib
(https://colorlib.com) -->

```

[illegible]

```

        integrity="sha384-
eOJMYsd53iii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6
" crossorigin="anonymous">
    <link rel="stylesheet" type="text/css"
th:href="@{/css/style.css}">
</head>
<body>
<div th:replace="fragments/header :: header"></div>
<div class="col" layout:fragment="content">
    <h1>This is the Layout's main section</h1>
    <p>This is the main content placeholder. This content will
be replaced if the page
        using the layout also defines a
layout:fragment="content" segment. </p>
</div>
</body>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta3/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-
JEW9xMcG8R+ph31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf
"
        crossorigin="anonymous"></script>
<script type="text/javascript"
th:src="@{/js/index.js}"></script>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Projects</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <div class="row justify-content-md-center">
            <button id="addProject" type="button" class="btn
btn-primary col-5" data-bs-toggle="modal"
                data-bs-target="#addProjectModal">
                Add project
            </button>
            <div class="modal fade" id="addProjectModal"
tabindex="-1" aria-labelledby="addProjectModalLabel"
                aria-hidden="true">
                <div class="modal-dialog">
                    <div class="modal-content">

```

```

        <div class="modal-header">
            <h5 class="modal-title"
id="addProjectModalLabel">Add project</h5>
            <button type="button" class="btn-
close" data-bs-dismiss="modal" aria-label="Close"></button>
        </div>
        <form method="post">
            <div class="modal-body">
                <div class="col">
                    <div class="mb-3">
                        <label for="name"
class="form-label">Name</label>
                        <input type="text"
class="form-control" name="name" id="name" placeholder="Name"
required>
                        <span
id="errorSpanProject" th:if="{error != null}"
th:text="{error}"
class="text-
danger">
                            </span>
                        </div>
                    </div>
                    <div class="mb-3">
                        <label for="description"
class="form-label">Description</label>
                        <textarea
name="description" class="form-control" id="description"
placeholder="Description"></textarea>
                    </div>
                    <div class="mb-3">
                        <label for="finishDate"
class="form-label">Deadline</label>
                        <input type="datetime-
local" name="finishDate" class="form-control" id="finishDate">
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn
btn-secondary" data-bs-dismiss="modal">Close</button>
                    <button type="submit" class="btn
btn-primary">Add</button>
                </div>
            </div>
        </form>
    </div>
</div>
</div>

```

```

        </div>
    </div>
    <div class="container mt-15">
        <div class="row">
            <div data-th-each="project : ${projects}"
class="col-sm-3 mt-15">
                <div class="card">
                    <div class="card-body">
                        <h5 data-th-text="${project.name}"
class="card-title"></h5>
                        <hr>
                        Deadline: <div data-th-
text="${#temporals.format(project.finishDate, 'dd-MM-yyyy
HH:mm')}" class="card-text">
                        </div>
                        <div class="collapse"
th:id="${project.id}">
                            <hr th:if="${project.finishDate}">
                            Description: <div data-th-
text="${project.description}">
                            </div>
                            <hr th:if="${project.name}">
                            Create time: <div data-th-
text="${#temporals.format(project.createTime, 'dd-MM-yyyy
HH:mm')}">
                            </div>
                            <hr th:if="${project.name}">
                            Start time: <div data-th-
text="${#temporals.format(project.startTime, 'dd-MM-yyyy
HH:mm')}">
                            </div>
                            <hr
th:if="${project.author.username}">
                            Author: <div data-th-
text="${project.author.username}">
                            </div>
                            <hr
th:if="${project.executor.username}">
                            Executor: <div data-th-
text="${project.executor.username}">
                            </div>
                            <hr th:if="${project.isOpen}">
                            Is open: <div data-th-
text="${project.isOpen}">
                            </div>
                            <hr th:if="${project.description}">
                            <span
sec:authorize="hasRole('ROLE_MANAGER')">

```

```

        <div class="row justify-content-md-
center">
            <a class="btn btn-primary col-4"
                data-th-href="@{'/tasks/' +
${project.id}}">Tasks</a>
            <div class="col-3"></div>
            <a class="btn btn-danger col-4"
                data-th-
href="@{'/projects/delete/' + ${project.id}}">Delete</a>
            </div>
        </span>
    </div>
    <div class="row justify-content-md-
center">
        <a class="btn-link col-auto mt-10"
            data-bs-toggle="collapse"
            data-th-href="@{'#' +
${project.id}}"
            role="button" aria-
expanded="false">
            More
        </a>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
<script type="text/javascript"
th:src="@{/js/project.js}"></script>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Task Manager</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-50">
        <p class="row justify-content-md-center">Welcome to Task
Manager!</p>
    </div>

```

```

</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Manage user</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <form method="post" action="/admin/users/manage"
class="row justify-content-md-center">
            <div class="mb-3 col-7">
                <label for="exampleInputEmail1" class="form-
label">Email address</label>
                <input data-th-value="{user.email}"
type="email" name="email" class="form-control"
id="exampleInputEmail1">
            </div>
            <div class="mb-3 col-7">
                <label for="exampleInputEmail11" class="form-
label">First Name</label>
                <input data-th-value="{user.firstName}"
type="text" name="email" class="form-control"
id="exampleInputEmail11">
            </div>
            <div class="mb-3 col-7">
                <label for="exampleInputEmail12" class="form-
label">Second Name</label>
                <input data-th-value="{user.secondName}"
type="text" name="email" class="form-control"
id="exampleInputEmail12">
            </div>
            <div class="mb-3 col-7">
                <input type="hidden" th:value="{user.id}"
name="userId">
                <label for="username" class="form-
label">Username</label>
                <input th:value="{user.username}" type="text"
name="username" class="form-control" id="username"
required>
                <span th:if="{error != null}"
th:text="{error}">
            </div>
        </form>
    </div>
</main>
</body>
</html>

```

```

        class="text-danger">
    </span>
</div>
<div class="mb-3 col-7">
    <label for="projectSelect" class="form-
label">Role</label>
    <select class="form-select" id="projectSelect"
name="projectId">
        <option>Developer</option>
        <option>Manager</option>
        <option>Teamlead</option>
    </select>
</div>

<div class="col-7">
    <div>
        <a data-th-href="@{'/admin/users/delete/' +
${user.id}}" class="btn btn-warning col-12">Delete</a>
    </div>
    <div class="row mt-15 justify-content-md-
center">
        <button type="submit" class="btn btn-primary
col-5">Save</button>
        <div class="col-1"></div>
        <a href="/" class="btn btn-danger col-
5">Cancel</a>
    </div>
</div>
</form>
</div>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Users</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-50 margin-container">
        <table class="table">
            <thead>
                <tr>

```

```

        <th scope="col">Email</th>
        <th scope="col">Username</th>
        <th scope="col">Role</th>
        <th scope="col">Delete</th>
        <th scope="col">Manage</th>
    </tr>
</thead>
<tbody>
<tr data-th-each="user : ${users}">
    <td data-th-text="${user.email}"></td>
    <td data-th-text="${user.username}"></td>
    <td data-th-text="${user.role}"></td>
    <td><a data-th-href="@{'/admin/users/delete/' +
${user.id}}">Delete</a></td>
    <td><a data-th-href="@{'/admin/users/manage/' +
${user.id}">Manage</a></td>
</tr>
</tbody>
</table>
</div>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Profile</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <form method="post" class="row justify-content-md-
center">
            <div class="mb-3 col-7">
                <label for="currentPassword" class="form-
label">Current password</label>
                <input type="password" name="currentPassword"
class="form-control" id="currentPassword" required>
                <span th:if="${error != null}"
th:text="${error}"
                    class="text-danger">
                </span>
            </div>
            <div class="mb-3 col-7">

```

```

        <label for="newPassword" class="form-label">New
password</label>
        <input type="password" name="newPassword"
class="form-control" id="newPassword" required>
        </div>
        <div class="mb-3 col-7">
            <label for="newPassword2" class="form-
label">Password confirmation</label>
            <input type="password"
oninput="validatePassword()" name="newPassword2" class="form-
control"
                id="newPassword2" required>
            </div>
        <div class="container col-7">
            <div class="row mt-15 justify-content-md-
center">
                <button type="submit" class="btn btn-primary
col-5">Save</button>
                <div class="col-1">
                    </div>
                <a href="/profile" class="btn btn-danger
col-5">Cancel</a>
            </div>
        </div>
    </form>
</div>
</div>
<script type="text/javascript"
th:src="@{/js/user.js}"></script>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Registration</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <form method="post" class="row justify-content-md-
center">
            <div class="mb-3 col-lg-7">
                <label for="username" class="form-

```

```

label">Username</label>
        <input type="text" name="username" class="form-
control" id="username" required>
        <span th:if="{error != null}"
th:text="{error}"
            class="text-danger">
        </span>
    </div>
    <div class="mb-3 col-lg-7">
        <label for="password" class="form-
label">Password</label>
        <input type="password" name="password"
class="form-control" id="password" required>
        </div>
        <div class="row col-7 mt-15 justify-content-md-
center">
            <button type="submit" class="btn btn-primary
col-5">Registration</button>
            <div class="col-2"></div>
            <a class="btn btn-primary col-5" data-th-
href="@{'/'}" role="button">Cancel</a>
        </div>

    </form>
</div>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Tasks</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <div class="row justify-content-md-center">
            <button id="addTask" type="button" class="btn btn-
primary col-5" data-bs-toggle="modal"
                data-bs-target="#addTaskModal">
                Add task
            </button>
            <div class="modal fade" id="addTaskModal"
tabindex="-1" aria-labelledby="addTaskModalLabel"

```

```

        aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title"
id="addTaskModalLabel">Add task</h5>
                <button type="button" class="btn-
close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <form method="post">
                <div class="modal-body">
                    <div class="col">
                        <div class="mb-3">
                            <label for="name"
class="form-label">Name</label>
                            <input type="text"
class="form-control" name="name" id="name" placeholder="Name"
required>
                            <span id="errorSpanTask"
th:if="{error != null}" th:text="{error}"
                                class="text-
danger">
                                </span>
                            </div>
                        <div class="mb-3">
                            <label for="description"
class="form-label">Description</label>
                            <textarea
name="description" class="form-control" id="description"
placeholder="Description"></textarea>
                        </div>
                        <div class="mb-3">
                            <label for="finishDate"
class="form-label">Deadline</label>
                            <input type="datetime-
local" name="finishDate" class="form-control" id="finishDate">
                        </div>
                        <div class="mb-3">
                            <select class="form-
select" id="projectSelect" name="projectId">
                                <option data-th-
each="project : {projects}" th:value="{project.id}"
th:text="{project.name}"></option>
                                </select>
                            </div>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>

```

```

        </div>
        <div class="modal-footer">
            <button type="button" class="btn
btn-secondary" data-bs-dismiss="modal">Close</button>
            <button type="submit" class="btn
btn-primary">Add</button>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>
<div class="container mt-15">
    <div class="row">
        <div data-th-each="task : ${tasks}" class="col-sm-3
mt-15">
            <div class="card">
                <div class="card-body">
                    <h5 data-th-text="${task.name}"
class="card-title"></h5>
                    <hr>
                    <div data-th-
text="${#temporals.format(task.finishDate, 'dd-MM-yyyy HH:mm')}"
class="card-text"></div>
                    <div class="collapse"
th:id="${task.id}">
                        <hr th:if="${task.finishDate}">
                        <div data-th-
text="${task.description}">
                            </div>
                        <hr th:if="${task.description}">
                        Project: <span data-th-
text="${task.project.name}"></span>
                        <hr>
                        <div class="row justify-content-md-
center">
                            <div class="col-4"></div>
                            <div class="col-3"></div>
                            <a class="btn btn-danger col-4"
data-th-
href="@{'/tasks/delete/' + ${task.id}}">Delete</a>
                        </div>
                    </div>
                <div class="row justify-content-md-
center">
                    <a class="btn-link col-auto mt-10"
data-bs-toggle="collapse"

```

```

        data-th-href="@{'#' +
        ${task.id}}"
        role="button"
        aria-expanded="false">
        More
    </a>
</div>
</div>
</div>
</div>
</div>
</div>
<script type="text/javascript"
th:src="@{/js/task.js}"></script>
</main>
</body>
</html>

<!DOCTYPE html>
<html lang="en"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layouts/base}">
<head>
    <meta charset="UTF-8">
    <title>Profile</title>
</head>
<body>
<main layout:fragment="content">
    <div class="container w-25">
        <form method="post" class="row justify-content-md-
center">
            <div class="mb-3 col-7">
                <label for="exampleInputEmail1" class="form-
label">Email address</label>
                <input data-th-value="${user.email}"
type="email" name="email" class="form-control"
id="exampleInputEmail1">
            </div>
            <div class="mb-3 col-7">
                <input type="hidden" th:value="${user.id}"
name="userId">
                <label for="username" class="form-
label">Username</label>
                <input th:value="${user.username}" type="text"
name="username" class="form-control" id="username"
required>
                <span th:if="${error != null}"
th:text="${error}"

```

```

        class="text-danger">
    </span>
</div>

<div class="col-7">
    <div>
        <a href="/change-password" class="btn btn-
secondary col-12">Change password</a>
    </div>
    <div class="row mt-15 justify-content-md-
center">
        <button type="submit" class="btn btn-primary
col-5">Save</button>
        <div class="col-1"></div>
        <a href="/" class="btn btn-danger col-
5">Cancel</a>
    </div>
</div>
</form>
</div>
</main>
</body>
</html>

```

```

* {
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}

body {
    padding: 0;
    margin: 0;
}

#notfound {
    position: relative;
    height: 100vh;
}

#notfound .notfound {
    position: absolute;
    left: 50%;
    top: 50%;
    -webkit-transform: translate(-50%, -50%);
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
}

```

```

.notfound {
    max-width: 520px;
    width: 100%;
    line-height: 1.4;
    text-align: center;
}

.notfound .notfound-404 {
    position: relative;
    height: 240px;
}

.notfound .notfound-404 h1 {
    font-family: 'Montserrat', sans-serif;
    position: absolute;
    left: 50%;
    top: 50%;
    -webkit-transform: translate(-50%, -50%);
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
    font-size: 252px;
    font-weight: 900;
    margin: 0px;
    color: #262626;
    text-transform: uppercase;
    letter-spacing: -40px;
    margin-left: -20px;
}

.notfound .notfound-404 h1>span {
    text-shadow: -8px 0px 0px #fff;
}

.notfound .notfound-404 h3 {
    font-family: 'Cabin', sans-serif;
    position: relative;
    font-size: 16px;
    font-weight: 700;
    text-transform: uppercase;
    color: #262626;
    margin: 0px;
    letter-spacing: 3px;
    padding-left: 6px;
}

.notfound h2 {
    font-family: 'Cabin', sans-serif;
    font-size: 20px;

```

```

    font-weight: 400;
    text-transform: uppercase;
    color: #000;
    margin-top: 0px;
    margin-bottom: 25px;
}

@media only screen and (max-width: 767px) {
    .notfound .notfound-404 {
        height: 200px;
    }
    .notfound .notfound-404 h1 {
        font-size: 200px;
    }
}

@media only screen and (max-width: 480px) {
    .notfound .notfound-404 {
        height: 162px;
    }
    .notfound .notfound-404 h1 {
        font-size: 162px;
        height: 150px;
        line-height: 162px;
    }
    .notfound h2 {
        font-size: 16px;
    }
}

.mt-15 {
    margin-top: 15px;
}

.mt-10 {
    margin-top: 10px;
}

.pb-100 {
    padding-bottom: 100px;
}

let projects = document.getElementsByClassName("card-title")
let projectName = document.getElementById("name")
function validateProject() {
    projectName.setCustomValidity('');
    for (let item of projects) {
        if(item.textContent === projectName.value)
projectName.setCustomValidity("Project already exists!");
    }
}

```

```

    }
}
projectName.onkeyup = validateProject;

let tasks = document.getElementsByClassName("card-title")
let taskName = document.getElementById("name")

taskName.onkeyup = validateTaskByName;

function validateTaskByName() {
    taskName.setCustomValidity('');
    for (let item of tasks) {
        if(item.textContent === taskName.value)
taskName.setCustomValidity("Task already exists!");
    }
}

let projectSelect = document.getElementById("projectSelect");
function validateTaskByProject() {
    if (projectSelect.childElementCount < 1)
projectSelect.setCustomValidity("Please choose or create a
project!");
}
taskName.onkeyup = validateTaskByProject;

let password = document.getElementById("newPassword")
let confirm_password = document.getElementById("newPassword2");
function validatePassword() {
    if (password.value !== confirm_password.value) {
        confirm_password.setCustomValidity("Passwords don't
match");
    } else {
        confirm_password.setCustomValidity('');
    }
}
password.onchange = validatePassword;
confirm_password.onkeyup = validatePassword;

```

Приложение В

(Обязательное)

Клиент-серверная система для планирования, распределения и управления
проектами

Презентация

					ПГУ 1.090501.09.001 ПЗ	Лист
						118
Изм.	Лист	№ докум.	Подпись	Дата		

Клиент-серверная система для планирования, распределения и управления проектами

Работу выполнил:

студент гр. 18ВОЭ1 Паняев Антон Валерьевич

Научный руководитель:

д.т.н., профессор каф ВТ ПГУ Зинкин Сергей Александрович

Пенза, 2023

Цель работы

Цель работы - разработать клиент-серверную систему для планирования, распределения и управления проектами.

В итоге должно получиться несколько приложений на базе микросервисной архитектуры, позволяющие управлять проектами и задачами.

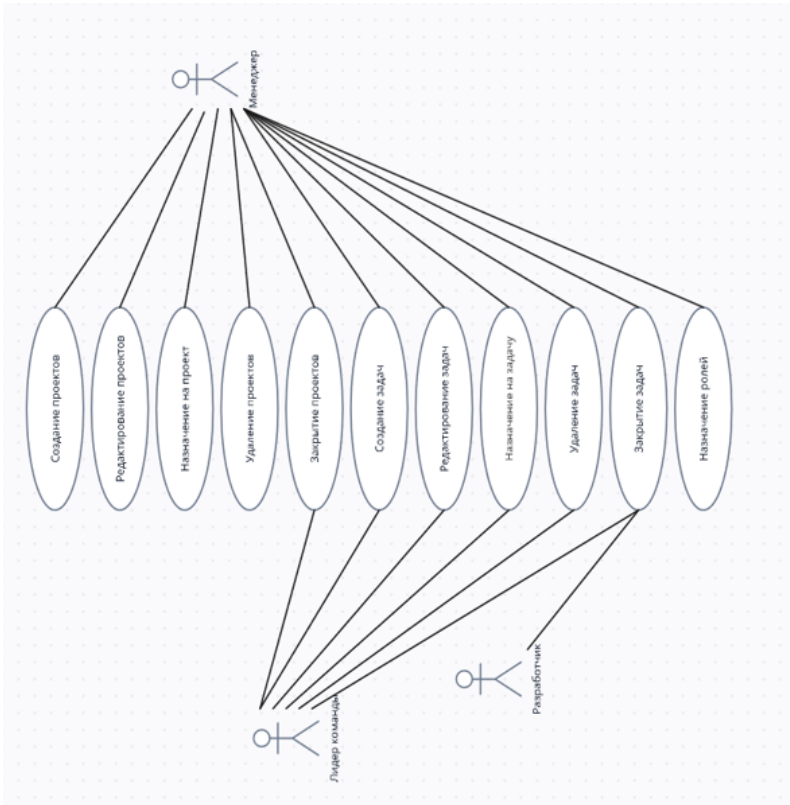
Стек технологий



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.09.001 ПЗ

Функции приложения



Архитектура приложения

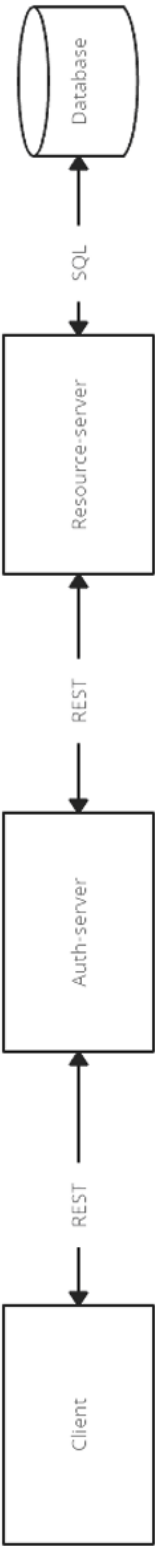
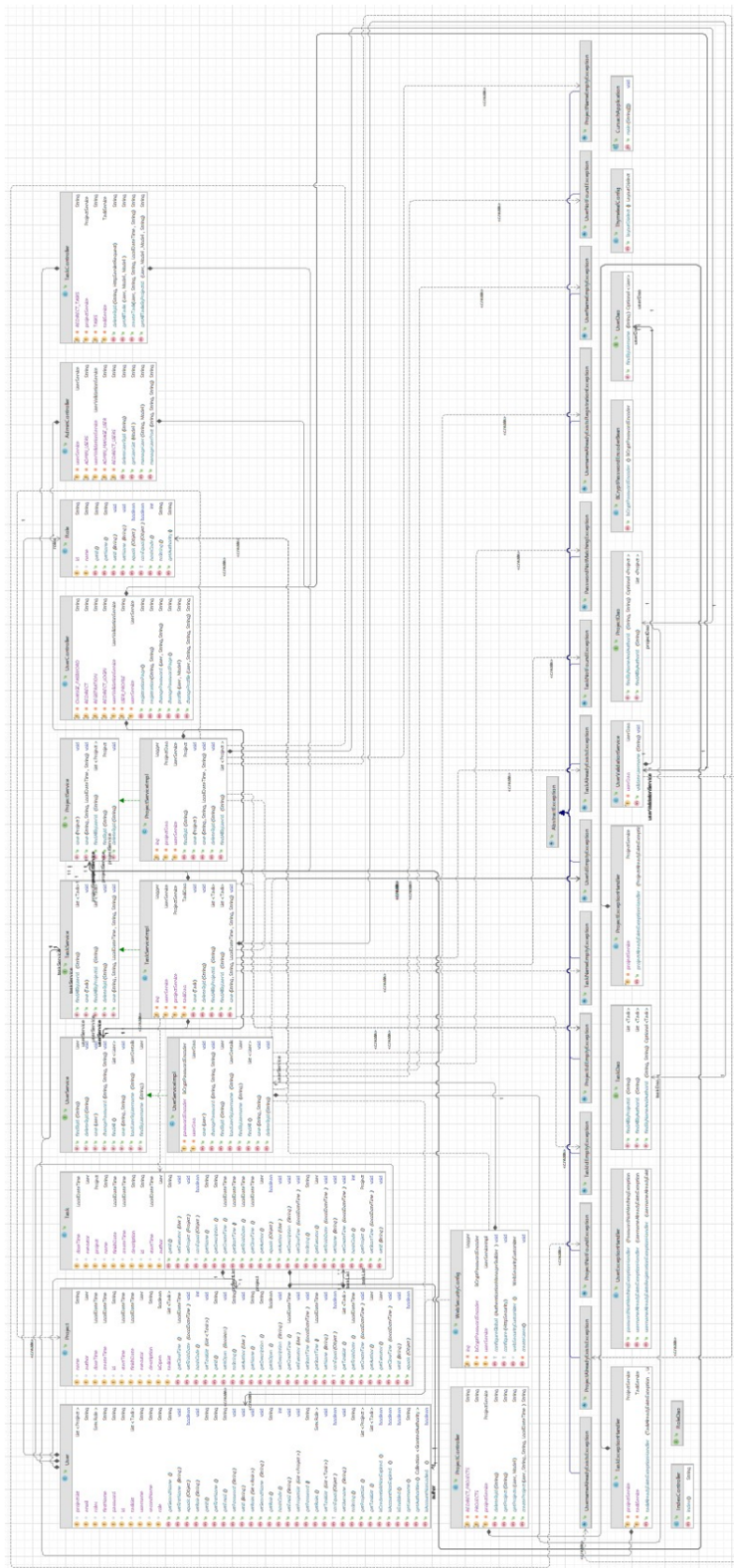
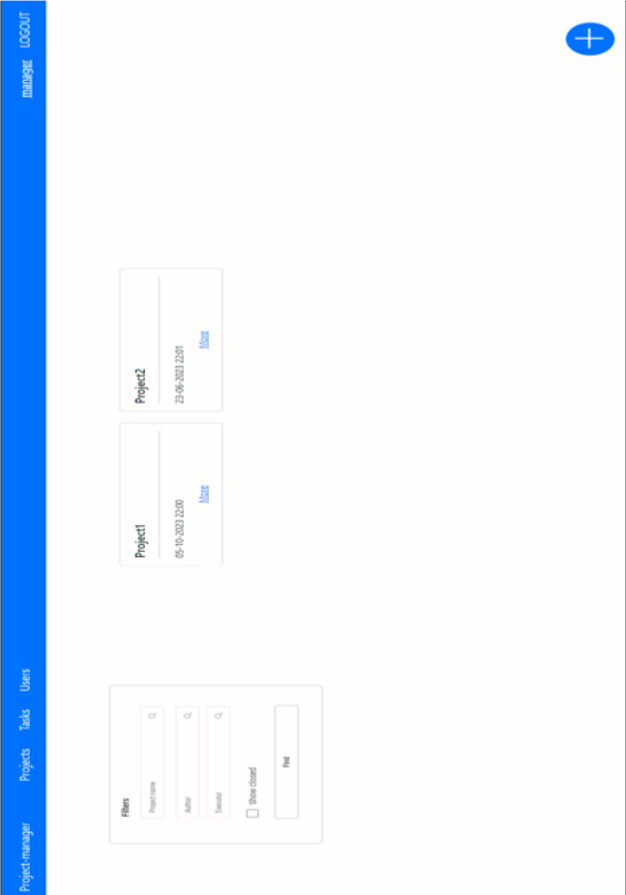
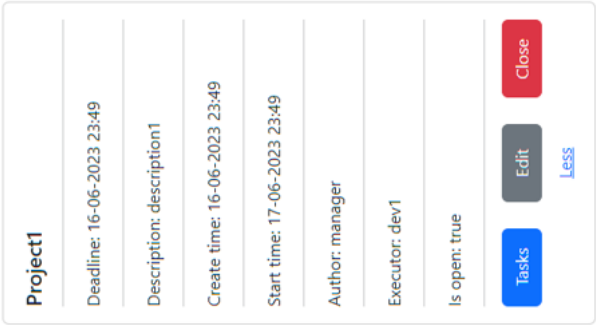


Диаграмма классов ресурсного сервера



Скриншоты приложения



Скриншоты приложения

Project manager

Projects

Tasks

Users

manage

logout

task1

23-06-2023 00:37

More

task2

17-06-2023 00:38

More

Filters

Task name

Project name

Author

Executor

Show closed

Find

+

Add task

Name

Name

Description

Description

Deadline

dd.mm.yyyy --:--

Start time

dd.mm.yyyy --:--

Executor

Executor

Project

Project1

Close

Add

task1

Deadline: 23-06-2023 00:37

Description: desc1

Create time: 16-06-2023 23:49

Start time: 17-06-2023 23:49

Author: manager

Executor: dev1

Is open: true

Project: Project1

Edit

Delete

More

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090501.09.001 ПЗ

Лист
126

Скриншоты приложения

Email	Username	Role	Delete	Manage
test@mail.ru	dev1	DEVELOPER	Delete	Manage
test@mail.ru	manager	MANAGER	Delete	Manage
test@mail.ru	teamlead1	TEAMLEAD	Delete	Manage

Email address

test@mail.ru

Username

dev1

First name

name2

Second name

name2

Role

DEVELOPER

Delete

Save

Cancel

Please sign in

Username

Password

Sign in

Заключение

В ходе выполнения дипломного проекта была разработана клиент-серверная система для планирования, распределения и управления проектами.

Разработанная система позволяет:

- создавать/удалять/редактировать проекты
- создавать/удалять/редактировать задачи для проектов
- управление всеми процессами с помощью удобного пользовательского интерфейса

Изм.	Лист	№ докум.	Подпись	Дата