МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ТРЕКЕР ЗАДАЧ

Студент 30.05.2024 Горожанин Ярослав Александрович
(подпись, дата) (ФИО полностью)

Руководитель 30.05.2024 С.А. Зинкин
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 30.05.2024 А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от 10.06.24 № 12)

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

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«19» 04 2024г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Горожанин Ярослав Александрович гр. 20ВВП2 факультета вычислительной техники.

2. Тема работы «Трекер задач».

Тема утверждена приказом ПГУ № 1057/0 от "19" апреля 2024 г.

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

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

Разработать приложение «Трекер задач» для отслеживания задач онлайн. Приложение должно позволять пользователям создавать задачи, назначать на них других пользователей, группировать эти задачи в проекты. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметных областей. Выбрать технологии разработки приложения для отслеживания задач.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	50	19.04.2024-27.04.2024	
2	Анализ требований	10	28.04.2024-29.04.2024	
3	Выбор технологий, применяемых при проектировании и разработке	50	30.04.2024-05.05.2024	
4	Описание алгоритмов	40	06.05.2024-10.05.2024	
5	Разработка и отладка «Трекер задач»	150	11.05.2024-27.05.2024	
6	Оформление выпускной работы	24	28.05.2024-31.05.2024	

Дата выдачи задания "19" 04. 2024 г.

Руководитель бакалаврской работы 19.04.2024 С.А. Зинкин

(подпись, дата)

Задание к исполнению принял студент 19.04.2024 Я.А. Горожанин

(подпись, дата)

Работу к защите допустить

Декан факультета 06.06.2024 Л.Р. Фионова

(подпись, дата)

Реферат

Бакалаврская работа содержит 52 листа, 17 рисунков, 9 источников и 3 приложения.

POSTGRESQL, INTELLIJ IDEA, MVC, SPRING FRAMEWORK, JAVA, DBEAVER, APACHE TOMCAT, ОТСЛЕЖИВАНИЕ ЗАДАЧ

Цель работы – разработать приложение, позволяющее отслеживать выполнение задач онлайн.

Объектом разработки является приложение, которое позволяет пользователям создавать, изменять и удалять задачи, назначать на них других пользователей, объединять задачи в проекты, оставлять и удалять комментарии к задачам, а также прикреплять и удалять файлы к задачам. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя, давать возможность редактировать информацию о пользователе.

Результатом проделанной работы является проект, который удовлетворяет поставленной задаче. А именно, разработано, реализовано и протестировано приложение для отслеживания задач. При разработке было использовано большое количество технологий, к которым относятся: язык программирования Java, инструмент для создания приложений Spring Framework, WPF для написания интерфейса приложения, среда разработки IntelliJ IDEA, веб-сервер и контейнер сервлетов Apache Tomcat. Для хранения данных выбрана СУБД PostgreSQL, а для её администрирования применялся инструмент DBeaver.

Содержание

Перечень сокращений и обозначений 5

Введение 6

1 Приложение «Трекер задач» 7

1.1 Анализ предметной области 7

1.1.1 Анализ актуальности темы 7

1.1.2 Постановка задачи 12

2 Архитектура приложения «Трекер задач» 13

2.1 Клиент-серверная архитектура 13

2.2 Шаблон проектирования MVC 14

2.3 UML-диаграммы 15

3 Описание инструментов разработки приложения 22

3.1.1 Java 22

3.1.2 Spring Framework 22

3.1.3 Spring Data JPA 23

3.1.4 Spring Security 23

3.1.5 PostgreSQL 24

3.1.6 DBeaver 24

3.1.7 IntelliJ IDEA 24

3.1.8 Apache Tomcat 24

4 Разработка приложения 26

4.1 Вариант использования «Просмотр задачи». 26

4.1.1 TaskController 28

4.1.2 TaskService 31

4.2 Вариант использования «Просмотр информации об аккаунте» 35
4.2.1 UserController 36
4.2.2 UserService 39
5 Тестирование 44
6 Безопасность жизнедеятельности 46
6.1 Вред здоровью при работе с ПК 46
6.2 Снижение вреда для здоровья при работе с ПК 46
6.3 Техника безопасности при работе с компьютером 47
Заключение 48
Список использованных источников 49
Приложение А 50
Приложение А.1 - Диаграмма классов 51
Приложение А.2 - Диаграмма прецедентов 52
Приложение А.3 - Диаграммы последовательности 53
Приложение А.4 - Диаграммы деятельности 58
Приложение А.5 - Диаграмма компонентов 60
Приложение Б 61
Приложение В 92

Перечень сокращений и обозначений

БД – база данных

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

HTTP (англ. Hyper Text Transfer Protocol) – протокол передачи гипертекста

IDE (англ. Integrated Development Environment) – интегрированная среда разработки

MVC (англ. Model-View-Controller) – это схема разделения данных приложения и управляющей логики

REST (англ. REpresentational State Transfer) – это архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SOAP (англ. Simple Object Access Protocol) – это протокол, по которому веб-приложения взаимодействуют между собой или с клиентом

UML (англ. Unified Modeling Language) – унифицированный язык моделирования

Введение

В современном мире, где время играет ключевую роль, эффективное управление задачами становится всё более актуальным. Трекер задач — это инструмент, позволяющий систематизировать и контролировать выполнение задач, отслеживать прогресс и оптимизировать рабочий процесс.

Приложение трекер задач позволит повысить производительность труда, лучше организовывать рабочее время, ускорить формирование отчетности, улучшить прозрачность и видимость задач, и отслеживать их статус и прогресс выполнения.

Таким образом, разработка клиент-серверного приложения трекер задач является актуальной и важной задачей для современного общества, поскольку способствует более эффективному управлению задачами, что в свою очередь повышает производительность работников.

1 Приложение «Трекер задач»

1.1 Анализ предметной области

1.1.1 Анализ актуальности темы

В контексте рабочей среды планирование задач играет ключевую роль в продуктивной организации труда. Приложения для отслеживания задач помогают добиться этого. Они позволяют быстро подобрать задачу самому и приступить к её выполнению или руководству назначить задачу на работника, основываясь на нынешних приоритетах команды. Тем самым оптимизируя рабочий процесс, путем эффективного распределения времени пользователей. В ходе выполнения сотрудник может документировать свой прогресс, что позволит анализировать его нагрузку.

Одно из ключевых преимуществ использования трекеров задач — это улучшение прозрачности и видимости задач, пользователи всегда будут знать, какие задачи активны и завершены, что позволяет распределять ресурсы с учётом приоритетности.

Таким образом, использование приложения для планирования задач актуально для всех пользователей, которым необходимо улучшить продуктивность своей работы.

Анализ аналогов клиент-серверных трекеров задач

Чтобы создать приложение для отслеживания задач, нужно проанализировать аналоги. Это поможет выявить плюсы и минусы существующих решений, а также определить требования к программе. В этом пункте рассмотрены три популярных сервиса для отслеживания задач:

– онлайн-сервис «Todoist»;

– онлайн-сервис «Asana»;

– онлайн-сервис «TickTick».

Онлайн-сервис «Todoist».

Особенностями данного аналога является возможность оформления платной подписки, которая позволяет добавить к задачам напоминания по времени и по местоположению.

Интерфейс онлайн-сервиса «Todoist» представлен на рисунках 1 и 2.

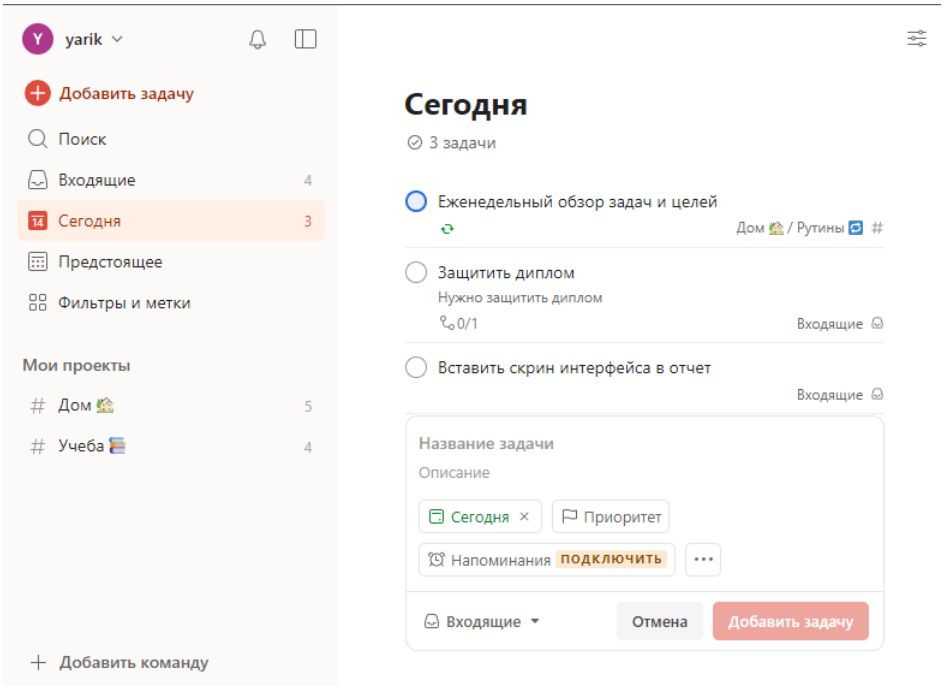


Рисунок 1 – Главная страница сервиса

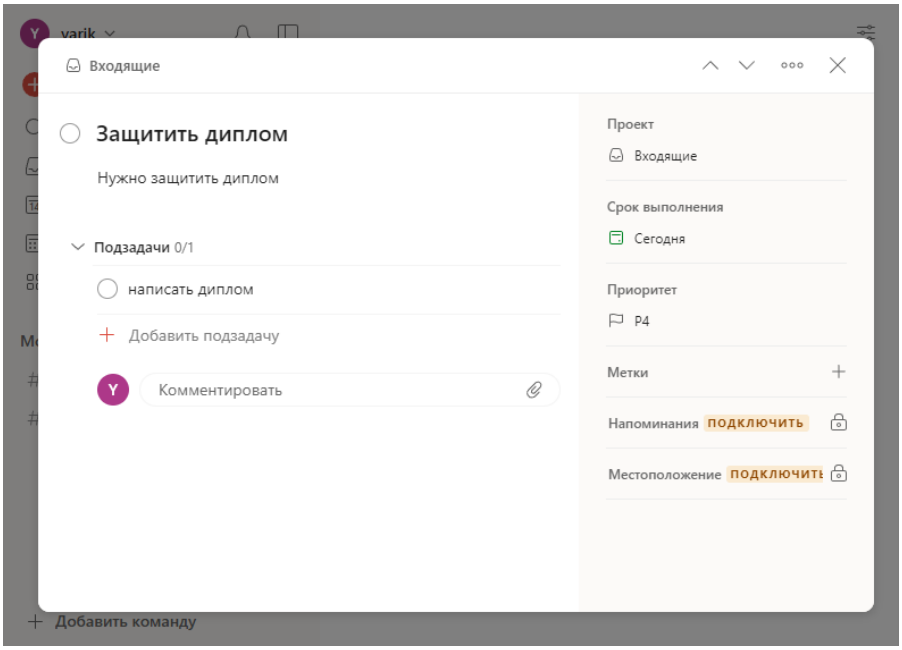


Рисунок 2 – Окно отдельной задачи

Сервис трекера задач «Todoist» предоставляет своим пользователям широкий спектр услуг для отслеживания задач. Ниже перечислены основные преимущества и недостатки данного сайта.

Преимущества:

– создание и настройка задач. Пользователь может создать задачу для себя, добавить её описание и при необходимости расписать подзадачи. Также доступен выбор приоритета и установка напоминания;

– поиск задач. Пользователь может найти задачи по их названию и приоритету;

– группировка задач в проекты. Пользователь может сгруппировать нужные задачи в проект или напрямую создавать их в проекте.

Недостатком является наличие платной подписки, без которой ограничивается функционал приложения.

Онлайн-сервис «Asana».

Сервис трекера задач «Asana» предоставляет своим пользователям широкий спектр услуг для отслеживания задач.

Интерфейс онлайн-сервиса «Asana» представлен на рисунках 3 и 4.

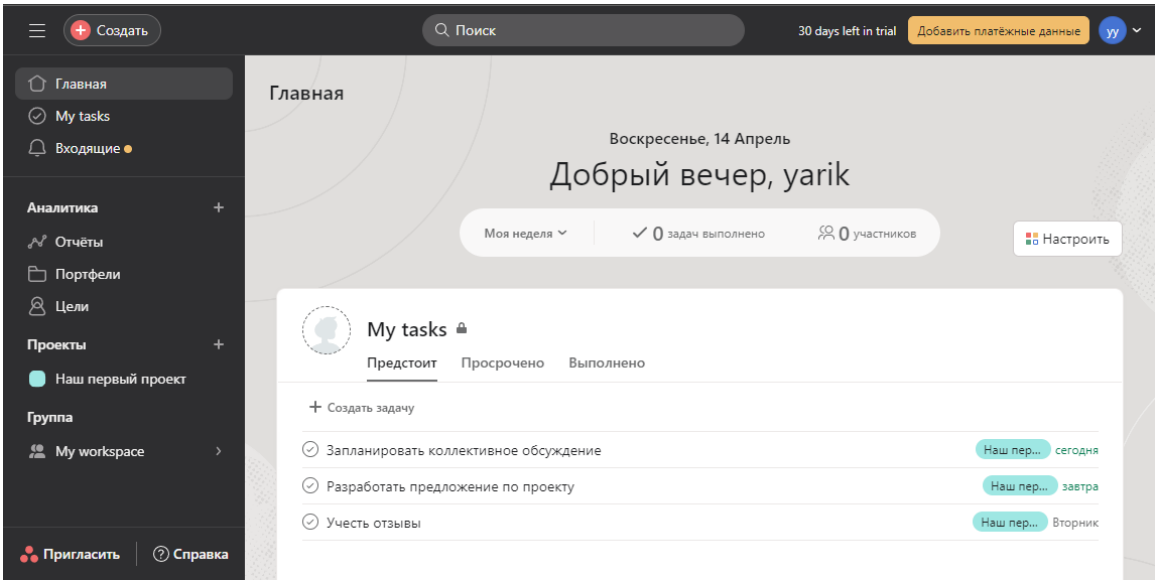


Рисунок 3 – Главная страница сервиса

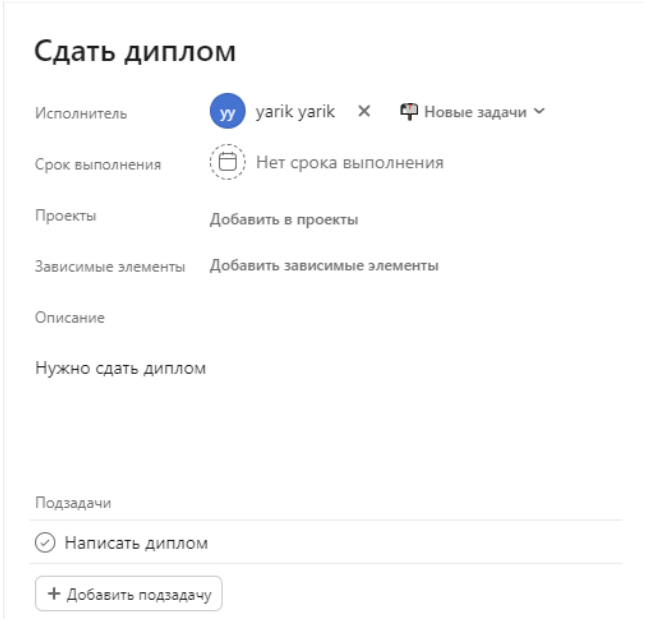


Рисунок 4 – Отдельная задача

Ниже перечислены основные преимущества и недостатки данного сайта.

Преимущества:

– создание и настройка задач. Пользователь может создать задачу, добавить её описание и при необходимости расписать подзадачи. Также доступно добавление файлов к задаче;

– поиск задач. Пользователь может найти задачи по их названию, исполнителю или наличию в проекте;

– группировка задач в проекты. Пользователь может сгруппировать нужные задачи в проект или напрямую создавать их в проекте.

Недостатком является наличие платной подписки, без которой ограничивается функционал приложения.

Онлайн-сервис «TickTick».

Сервис трекера задач «TickTick» предоставляет своим пользователям широкий спектр услуг для отслеживания задач.

Интерфейс онлайн-сервиса «TickTick» представлен на рисунках 5 и 6.

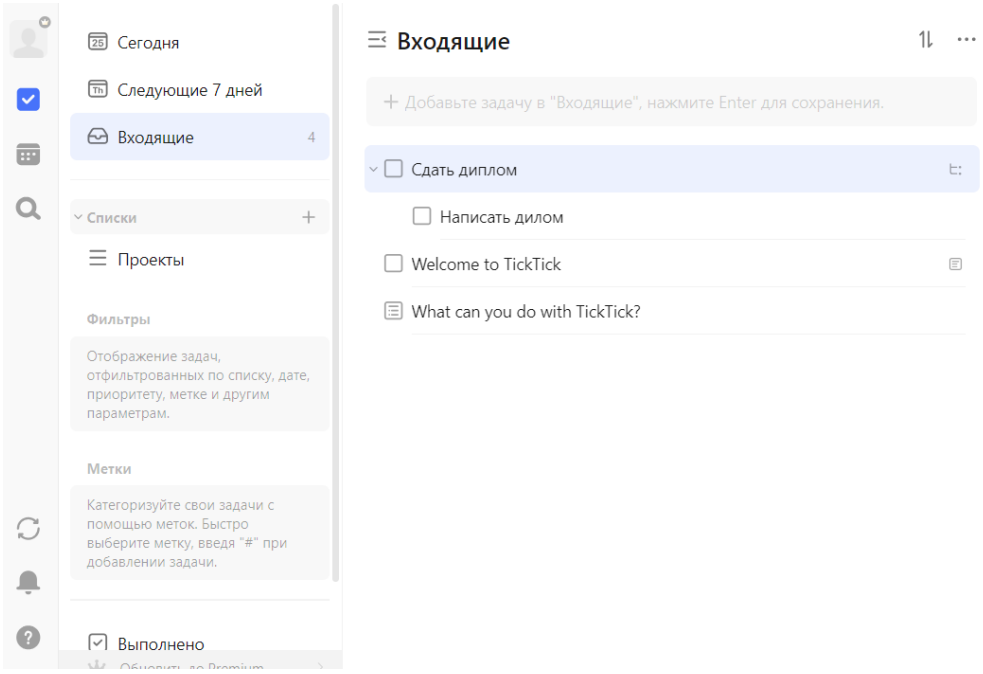


Рисунок 5 – Главная страница сервиса

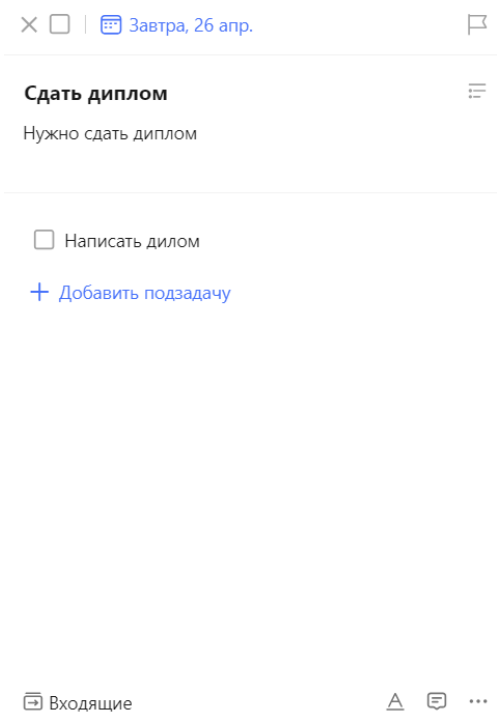


Рисунок 6 – Отдельная задача

Ниже перечислены основные преимущества и недостатки данного сайта.

Преимущества:

– создание и настройка задач. Пользователь может создать задачу, добавить её описание и при необходимости расписать подзадачи. Также доступно добавление файлов к задаче;

– поиск задач. Пользователь может найти задачи по их названию, исполнителю или наличию в проекте;

– группировка задач в проекты. Пользователь может сгруппировать нужные задачи в проект или напрямую создавать их в проекте.

Недостатком является наличие платной подписки, без которой ограничивается функционал приложения.

Вывод: на основе полученных результатов были сформированы требования к приложению, описанные в следующем пункте.

1.1.2 Постановка задачи

Необходимо разработать приложение, для отслеживания задач. Оно должно предоставлять возможность создания задач, документирование прогресса их выполнения с фиксированием затраченного времени для сотрудника [1].

Приложение должно соответствовать следующим требованиям:

– предоставлять данные о пользователе, задачах, проектах, затраченного времени;

– проверять корректность вводимой информации в базу данных;

– редактировать, добавлять и удалять данные о пользователе, задачах, проектах, затраченном времени;

– обеспечивать безопасность данных, позволяя взаимодействовать с задачами и проектами, только авторизованным пользователям.

Приложение должно предоставлять пользователям следующий функционал:

– регистрация и авторизация;

– создание задачи, проекта и записи о времени, затраченном на выполнение;

– редактирование, добавление и удаление данных о пользователе, задаче, проекте и записи о затраченном времени;

2 Архитектура приложения «Трекер задач»

Исходя из требований к приложению «Трекер задач» было принято решение адаптировать в проекте клиент-серверную архитектуру.

2.1 Клиент-серверная архитектура

Клиент-серверная архитектура – это модель распределённых систем, состоящая из двух типов узлов: клиент и сервер.

Клиент — это программа, работающая на компьютере пользователя и предоставляющая интерфейс для взаимодействия с сервером. Они могут быть приложениями или веб-браузерами, взаимодействующими с веб-серверами.

Сервер — это программа, которые предоставляют запрашиваемую информацию или услуги клиентам. Основная задача сервера — бесперебойная работа и возможность обрабатывать миллионы запросов от пользователей.

Взаимодействие клиента и сервера осуществляется по определённому протоколу, определяющему соглашение между ними. Клиент отправляет запросы на сервер, а сервер возвращает ответы на эти запросы [2].

Преимущества клиент-серверной архитектуры:

– разделение нагрузки на сервера позволяет эффективно обрабатывать большое количество запросов от клиентов;

– сервер контролирует всю систему, обеспечивая безопасность и управление доступом к данным;

– отказ одного компонента системы не влияет на работу остальных компонентов;

– централизованное управление сервером обеспечивает защиту данных.

Недостатки клиент-серверной архитектуры:

– если сервер недоступен или имеет проблемы, клиенты будут неработоспособны или столкнутся с проблемами функциональности;

– требуется поддержка серверного и сетевого оборудования;

– необходимость обеспечения безопасности и защиты данных на сервере.

Клиент и сервер взаимодействуют согласно определенному протоколу, определяющему соглашение между ними. Обычно протоколы взаимодействия бывают стандартизированы и поддерживаются множеством различных программ и устройств. Примерами могут служить такие протоколы и технологии как HTTP, SOAP, REST и т.д [2].

REST — это стиль архитектуры, используемый для создания распределённых систем, основанный на обмене представлениями ресурсов, хранящихся на сервере, между клиентом и сервером. В данной работе будет использоваться REST стиль, так как он является простым, гибким и эффективным стилем архитектуры.

С учетом выбранной архитектуры, были определены минимальные системные требования для клиента:

– операционная система: Windows (7, 8, 10);

– браузер: Google Chrome, Mozilla Firefox, Opera (с поддержкой HTML5, CSS3);

– минимальная скорость интернета: 2-3 Мбит/сек;

– минимальный объем оперативной памяти: 1 ГБ (рекомендуется 2 ГБ и выше);

– минимальный объем свободного места на жестком диске: 100 МБ.

2.2 Шаблон проектирования MVC

Model-View-Controller (MVC) — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо [3].

Модель (Model) предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние [3].

Представление (View) отвечает за отображение данных предметной области (модели) пользователю, реагируя на изменения модели [3].

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений [3].

MVC шаблон, был выбран, поскольку обладает рядом достоинств:

– чёткое разделение ответственности между компонентами приложения;

– упрощение отладки и тестирования благодаря разделению на слои;

– улучшение безопасности и защиты данных благодаря отделению пользовательского интерфейса от бизнес-логики.

Но также MVC шаблон обладает и некоторыми недостатками:

– сложность настройки и интеграции компонентов;

– возможное увеличение времени разработки из-за необходимости дополнительной настройки и конфигурации;

– необходимость знания нескольких технологий для создания полноценного приложения.

2.3 UML-диаграммы

Основываясь на установленных требованиях к приложению, а также на выбранных стиле архитектуры и шаблоне проектирования была определена структура приложения.

На рисунке 7 представлена диаграмма вариантов использования приложения.

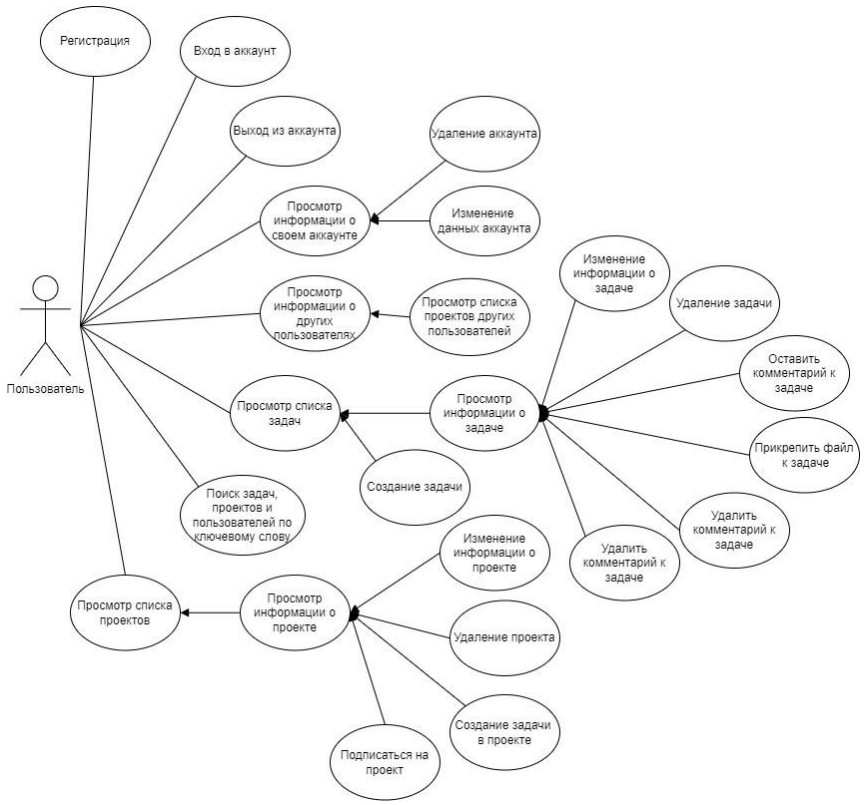


Рисунок 7 – UML-диаграмма вариантов использования приложения

На рисунке 8 представлена диаграмма компонентов приложения.

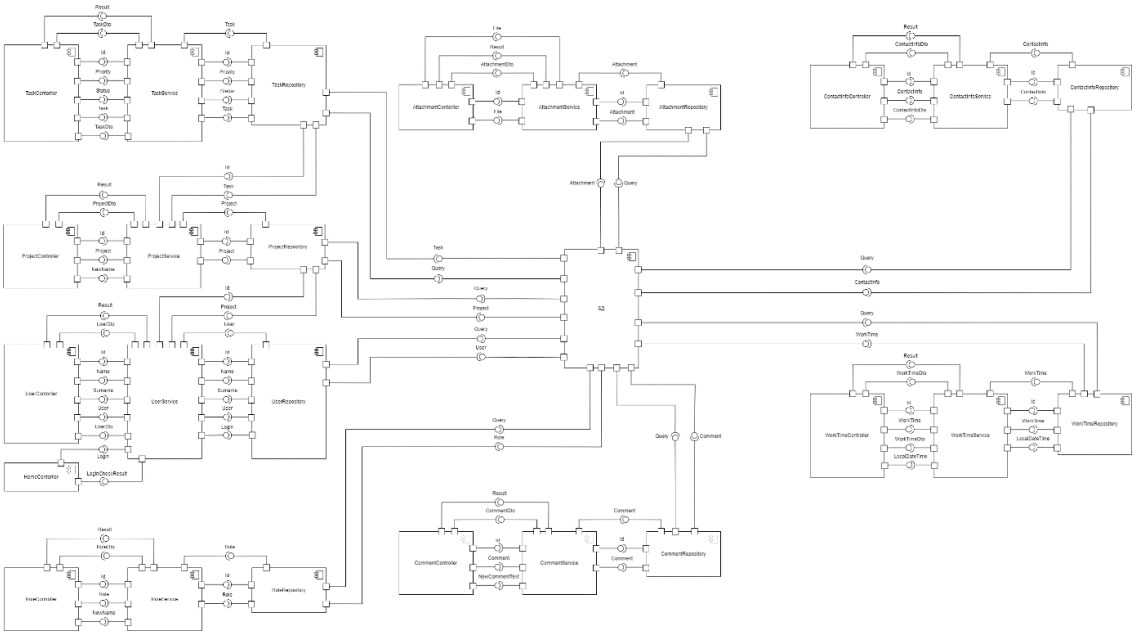


Рисунок 8 – UML-диаграмма компонентов приложения

На рисунке 9 представлена диаграмма классов приложения.

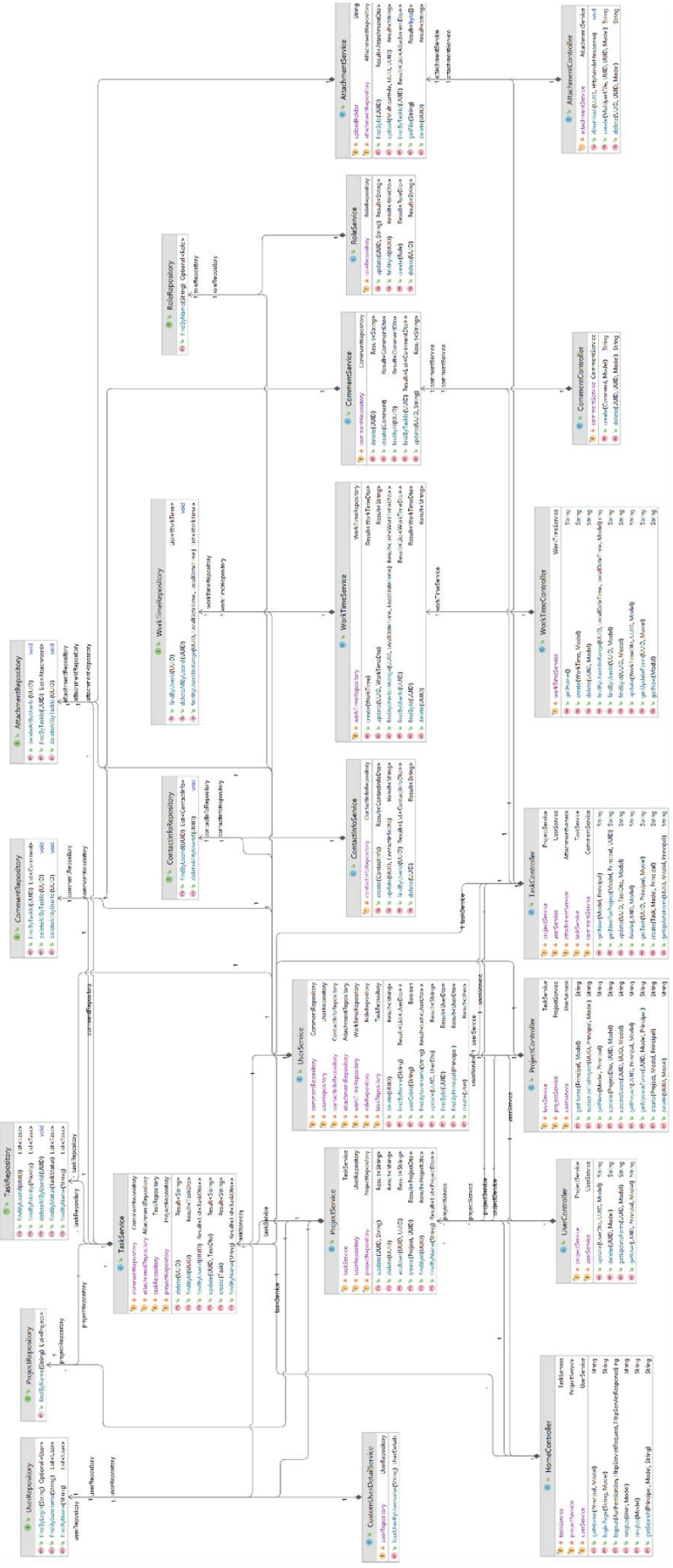


Рисунок 9 – UML-диаграмма классов приложения

На рисунках 10–14, представлены последовательные диаграммы основных взаимодействий пользователя и приложения.

При входе на страницу с задачами, от лица пользователя отправляется запрос на получение всех задач, на которые он назначен исполнителем. TaskController обрабатывает этот запрос, отправляя ID пользователя в TaskService, который в свою очередь делает запрос в БД через TaskRepository, получая на выходе список Task, который преобразуется в список TaskDto. Этот список в итоге и возвращается в TaskController, после чего последний передает его пользователю.

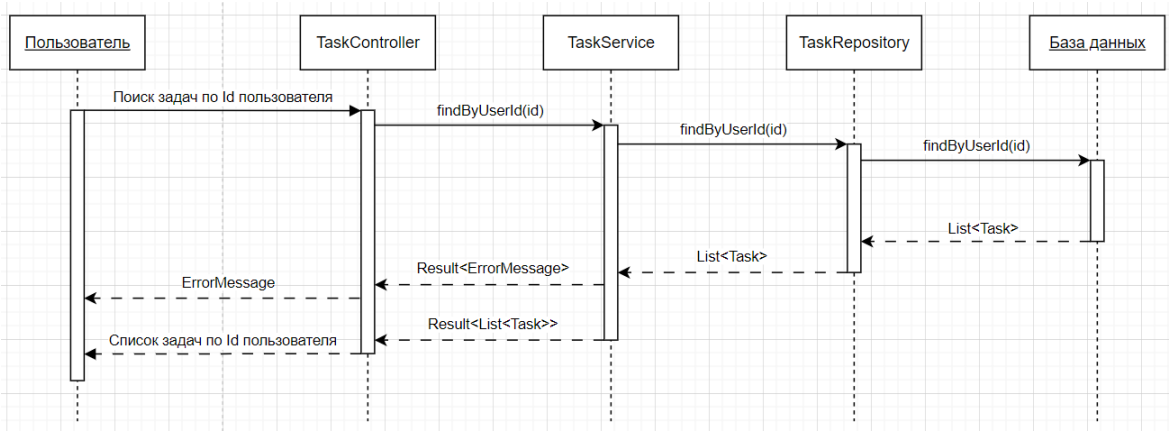


Рисунок 10 – UML-диаграмма последовательности поиска задач по Id пользователя

При создании задачи пользователь заполняет необходимую информацию о ней и отправляет соответствующий запрос. TaskController обрабатывает этот запрос, отправляя Task в TaskService, который в свою очередь делает запрос в БД на создание задачи через TaskRepository, получая на выходе информацию о созданной задаче, которая преобразуется в TaskDto. В таком виде информация о задаче возвращается в TaskController, после чего последний передает её пользователю.

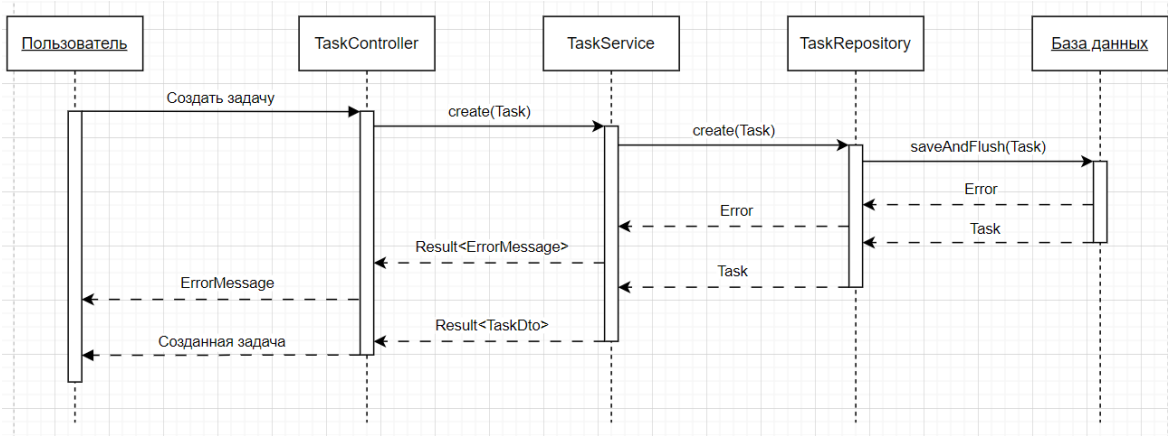


Рисунок 11 – UML-диаграмма последовательности создания задачи

При оставлении комментария к задаче пользователь отправляет соответствующий запрос. CommentController обрабатывает этот запрос, отправляя Comment в CommentService, который в свою очередь делает запрос в БД на добавление записи о комментарии через CommentRepository, получая на выходе информацию о созданном комментарии, которая преобразуется в CommentDto. В таком виде информация возвращается в CommentController, после чего последний передает её пользователю.

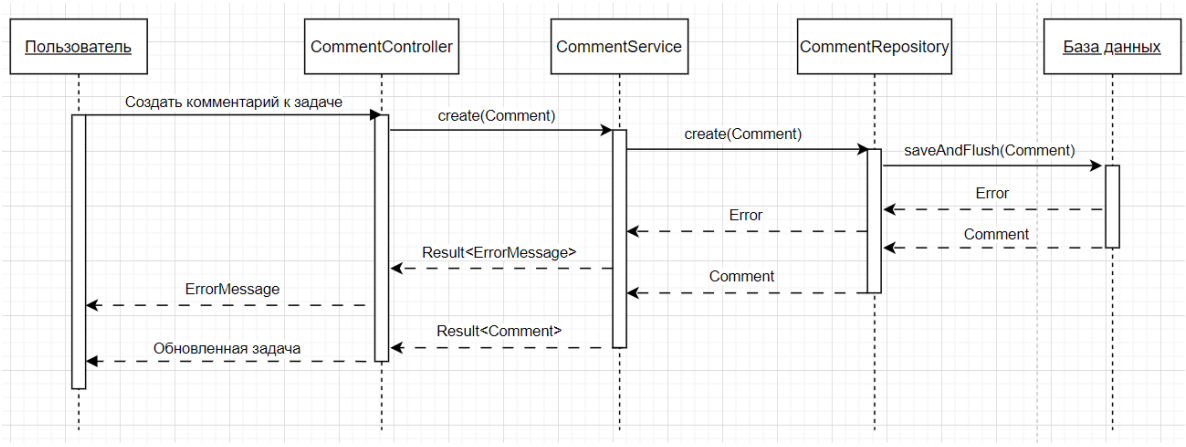


Рисунок 12 – UML-диаграмма последовательности создания комментария к задаче

При добавлении к задаче файла пользователь загружает этот файл и отправляет соответствующий запрос. AttachmentController обрабатывает этот

запрос, отправляя Attachment в AttachmentService, который в свою очередь делает запрос в БД на создание записи о файле через AttachmentRepository, получая на выходе информацию о добавленном файле, которая преобразуется в AttachmentDto. В таком виде информация возвращается в AttachmentController, после чего последний передает её пользователю.

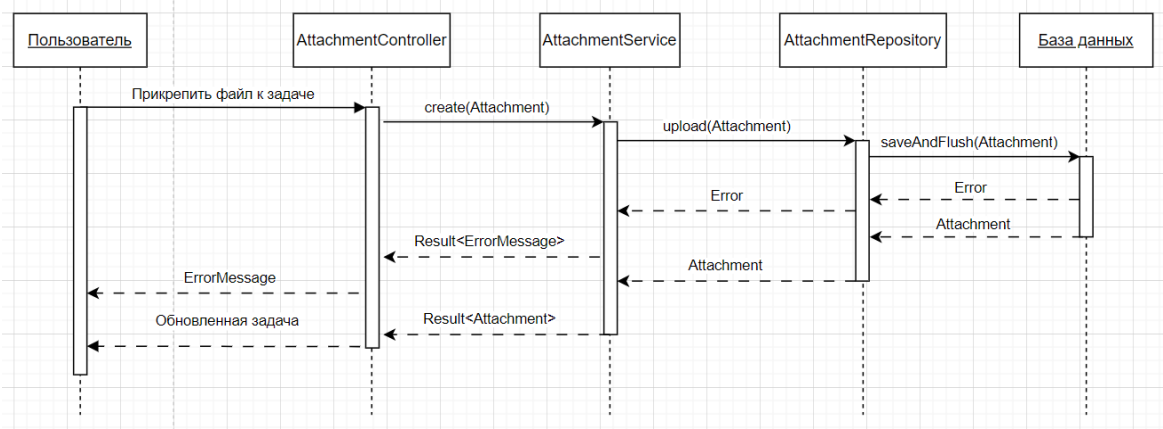


Рисунок 13 – UML-диаграмма последовательности прикрепления файла к задаче

При изменении контактных данных пользователя им отправляется соответствующий запрос. ContactInfoController обрабатывает этот запрос, отправляя ContactInfo в ContactInfoService, который в свою очередь делает запрос в БД на изменение контактных данных через ContactInfoRepository, получая на выходе информацию о добавленном файле, которая преобразуется в ContactInfoDto. В таком виде информация возвращается в ContactInfoController, после чего последний передает её пользователю.

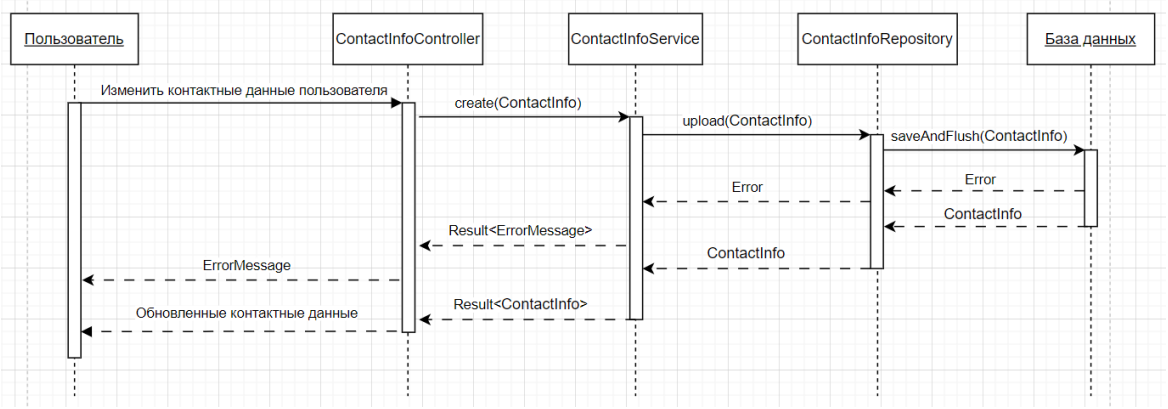


Рисунок 14 – UML-диаграмма последовательности изменения контактных данных пользователя

3 Описание инструментов разработки приложения

Руководствуясь выбранными архитектурой приложения и подходом к его разработке, а также опираясь на требования, определенные после анализа, были выбраны следующие инструменты:

– Java;

– Spring Framework;

– Spring Data JPA;

– Spring Security;

– PostgreSQL;

– DBeaver;

– Apache Tomcat.

3.1.1 Java

Java — это многоплатформенный, объектно-ориентированный и сетецентрический язык, который сам по себе может служить платформой. Java — быстрый, безопасный и надёжный язык программирования, подходящий для разработки мобильных приложений, корпоративного ПО, приложений для работы с большими данными и серверных технологий [4].

Java обладает строгой типизацией. Это значит, что она требует, чтобы был определён тип каждой переменной. Также предусмотрено автоматическое управление памятью, так называемый «сборщик мусора», благодаря которому программа самостоятельно выделяет и освобождает память. Стандартная библиотека содержит обширное количество классов и методов для работы с разными аспектами разработки, включая сети, базы данных и другое.

3.1.2 Spring Framework

Spring Framework — популярный и широко используемый фреймворк для создания Java приложений. Он включает в себя множество модулей и функций, которые упрощают и ускоряют процесс разработки программного обеспечения, а также улучшают его качество и надёжность [5].

Основные возможности Spring Framework:

– управление зависимостями. Spring предоставляет инструменты для управления связями между элементами приложения, что облегчает процесс разработки и улучшает его качество;

– внедрение зависимостей. Spring обладает функционалом для внедрения зависимостей между элементами приложения. Это делает разработку гибкой и расширяемой;

– управление транзакциями. Spring предоставляет инструменты для управления транзакциями в приложении, упрощая работу с БД и повышая надёжность и безопасность ПО.

Spring также поддерживает различные стандарты и спецификации, такие как Java EE, JPA и другие. Таким образом, этот инструмент можно использовать в разных средах и с различными технологиями [5].

3.1.3 Spring Data JPA

Spring JPA — это один из модулей фреймворка Spring. Он отвечает за работу с базами данных в Java-приложениях. Он позволяет работать с различными системами управления базами данных (СУБД) и облегчает процесс работы с данными [6]. Spring Data JPA содержит аннотации для сопоставления Java-объектов с таблицами в базе данных, уменьшая объём кода и упрощая соотношение таблиц и классов. Также JPA предоставляет методы для организации классов репозиториев, обеспечивающих работу с данными, что позволяет, облегчить и ускорить процесс работы с ними;

3.1.4 Spring Security

Spring Security — это модуль Spring Framework ориентированный на обеспечение безопасности веб приложений. Он предоставляет инструменты для аутентификации и авторизации пользователей, такие как базовая аутентификация по логину и паролю, автоматически сгенерированную форму входа и методы для работы с OAuth. Модуль также поддерживает ролевую авторизацию и правила

авторизации, обеспечивает защиту приложения от различных видов атак, таких как атаки на сессии, CSRF-атаки и XSS-атаки.

3.1.5 PostgreSQL

PostgreSQL — это объектно-реляционная СУБД, с обширными возможностями для хранения, управления и анализа данных. Она представляет собой клиент-серверную систему, в которой клиентское приложение подключается к серверу PostgreSQL для выполнения операций с базой данных. PostgreSQL поддерживает множество языков запросов, включая SQL. Данные хранятся в таблицах, состоящих из строк, столбцов и индексов. Также PostgreSQL поддерживает хранение данных в формате JSON.

3.1.6 DBeaver

DBeaver – это бесплатный кросс-платформенный инструмент для работы с базами данных, который поддерживает все популярные SQL-базы данных, в частности PostgreSQL [7]. Он позволяет создавать новые базы, изменять и удалять данные и выполнять SQL-запросы.

3.1.7 IntelliJ IDEA

IntelliJ IDEA — это IDE, созданная компанией JetBrains, предназначенная для языков программирования Java, Kotlin и других. В ней доступны инструменты и функции для комфортной и эффективной разработки ПО [8].

Эта IDE предлагает удобный рефакторинг кода, позволяющий быстро менять его структуру, переименовывать переменные и методы и многое другое. Кроме того, она интегрирована с популярными сборщиками проектов, такими как Maven, Gradle и Ant, что облегчает сборку и управление зависимостями проектов.

Также IntelliJ IDEA интегрируема с популярными серверами приложений, в частности Apache Tomcat. Такие интеграции позволяют упростить процесс разработки и развёртывания приложений.

3.1.8 Apache Tomcat

Apache Tomcat — это веб-сервер и контейнер сервлетов Java, созданный Apache Software Foundation. И представляет собой бесплатное ПО, используемое

для запуска Java-сервлетов и JSP-страниц веб приложений. Tomcat поддерживает стандарты Java Servlet, JavaServer Pages и другие. Может быть использован как самостоятельный веб-сервер. Поддерживает работу на разных ОС, включая Windows, Linux и macOS.

4 Разработка приложения

В этом разделе описан процесс разработки приложения «Трекер задач». Рассмотрены основные варианты использования и компоненты проекта, задействованные в них. Выполнено тестирование приложения.

4.1 Вариант использования «Просмотр задачи».

На рисунке 15 представлена диаграмма деятельности на вариант использования «Просмотр задачи». Пользователю предоставляются следующие возможные действия:

– изменение данных задачи;

– удаление задачи;

– создание комментария;

– удаление комментария;

– прикрепление файла к задаче;

– удаление прикрепленного к задаче файла.

Каждое возможное действие, запускает соответствующий метод, который делает запрос в базу данных и возвращает либо сообщение об ошибке, либо результат выполнения, после чего этот метод подает на выход полученные данные, которые затем отображаются пользователю.

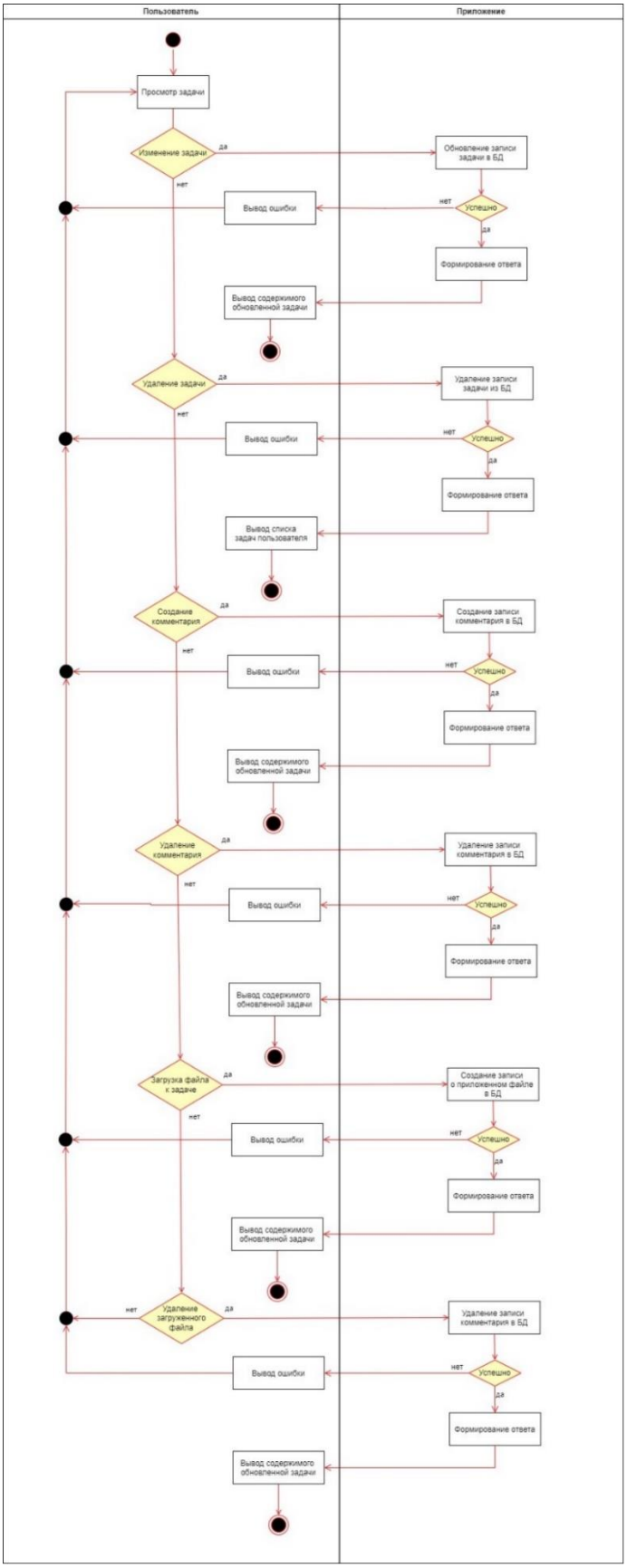


Рисунок 15 – UML-диаграмма деятельности на вариант использования «Просмотр задачи»

Далее представлены основные классы, задействованные в рассматриваемом варианте использования.

4.1.1 TaskController

Этот класс является контроллером и отвечает за обработку данных из приходящих к серверу запросов. Контроллер отвечает за все операции с данными о задачах и определяет, как и в каком виде они будут отображены.

Метод getTask() срабатывает когда на сервер приходит get-запрос по адресу «/task/{id}», где id – это идентификатор записи искомой задачи. Среди передаваемых параметров также есть объект Principal, с помощью которого определяется пользователь, от которого поступил запрос. Это нужно чтобы подтвердить его права на просмотр или редактирование данных. Параметр id передается в метод findById() класса TaskService. Результатом его работы объект класса TaskDto, содержащий нужную информацию о задаче. По цепочке за задачей выполняется поиск проекта, если она принадлежит к проекту, а также всех комментариев и приложенных к ней файлов.

Ниже представлен листинг метода getTask().

```
    @GetMapping("/{id}")
public String getTask(@PathVariable("id") UUID taskId, Principal
principal, Model model) {
    Result<UserDto> userRead =
userService.findByPrincipal(principal);
    if (userRead.isError()) {
        model.addAttribute("message", userRead.getMessage());
        return userRead.getCode();
    }
    UserDto userDto = userRead.getObject();
    Result<TaskDto> taskRead = taskService.findById(taskId);
    if (taskRead.isError()) {
        model.addAttribute("message", taskRead.getMessage());
        return taskRead.getCode();
    }

    TaskDto taskDto = taskRead.getObject();
    if (taskDto.getProjectId() != null) {
        Result<ProjectDto> projectRead =
projectService.findById(taskDto.getProjectId());
        if (projectRead.isError()) {
            model.addAttribute("message", projectRead.getMessage());
            return projectRead.getCode();
```

```
        }
        ProjectDto projectDto = projectRead.getObject();
        model.addAttribute("project", projectDto);
    }

    Result<List<CommentDto>> taskComments =
commentService.findByTaskId(taskId);
    if (taskComments.isOk()) {
        List<CommentDto> comments = taskComments.getObject();
        model.addAttribute("comments", comments);
    }

    Result<List<AttachmentDto>> attachmentRead =
attachmentService.findByTaskId(taskId);
    if (attachmentRead.isOk()) {
        List<AttachmentDto> attachmentDto =
attachmentRead.getObject();
        model.addAttribute("attachments", attachmentDto);
    }

    model.addAttribute("taskDto", taskDto);
    model.addAttribute("userDto", userDto);
    String searchQuery = "";
    model.addAttribute("searchQuery", searchQuery);
    model.addAttribute("newComment", new Comment());
    model.addAttribute("newAttachment", new Attachment());
    model.addAttribute("priorities", Priority.values());
    return "task/index";
}
```

Метод create() срабатывает когда на сервер приходит post-запрос по адресу «/task/». Среди передаваемых параметров есть объект Principal, и объект Task, содержащий все данные создаваемой задачи. Сперва в Task записывается информация о пользователе, создающем задачу, после чего объект передается в метод create() класса TaskService. Результатом его работы является информация об успешном создании задачи, либо информация об ошибке возникшей при создании задачи. После успешного создания задачи происходит перенаправление на домашнюю страницу со всеми задачами пользователя.

Ниже представлен листинг метода create().

```
     @PostMapping("/")
public String create(@ModelAttribute("task") Task task, Model model,
Principal principal) {
    Result<UserDto> userRead =
userService.findByPrincipal(principal);
    if (userRead.isError()) {
        model.addAttribute("message", userRead.getMessage());
```

```
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();
        User user = new User();
        user.setId(userDto.getId());
        task.setUser(user);
        Result<String> savedTask = taskService.create(task);
        if (savedTask.isError()) {
            model.addAttribute("message", savedTask.getMessage());
            return savedTask.getCode();
        } else {
            model.addAttribute("message", "Task create ok");
            if (task.getProjectId() != null) {
                return "redirect:/project/" + task.getProjectId();
            }
            return "redirect:/";
        }
}
```

Метод update() срабатывает когда на сервер приходит put-запрос по адресу «/task/{id}», где id – это идентификатор записи обновляемой задачи. Также среди передаваемых параметров есть объект TaskDto, содержащий новые данные для задачи. Объекты TaskDto и id передаются в метод update() класса TaskService. Результатом его работы является информация об успешном изменении задачи, либо информация о возникшей ошибке. После успешного изменения происходит перенаправление на страницу с обновленной задачей пользователя.

Ниже представлен листинг метода update().

```
     @PutMapping("/{id}")
public String update(@PathVariable("id") UUID id,
@ModelAttribute("task") TaskDto task, Model model) {
    Result<String> upload = taskService.update(id, task);
    if (upload.isError()) {
        model.addAttribute("message", upload.getMessage());
        return upload.getCode();
    } else
        return "redirect:/task/" + id;
}
```

Метод delete() срабатывает когда на сервер приходит delete-запрос по адресу «/task/{id}», где id – это идентификатор записи обновляемой задачи. Объект id передается в метод delete() класса TaskService. Результатом его работы является информация об успешном удалении задачи, либо информация о возникшей

ошибке. После успешного удаления задачи происходит перенаправление на домашнюю страницу со всеми задачами пользователя.

Ниже представлен листинг метода delete().

```
     @DeleteMapping("/{id}")
public String delete(@PathVariable("id") UUID id, Model model) {
    Result<String> delete = taskService.delete(id);
    if (delete.isError()) {
        model.addAttribute("message", delete.getMessage());
        return delete.getCode();
    } else {
        model.addAttribute("message", delete.getObject());
        return "redirect:/";
    }
}
```

4.1.2 TaskService

Этот класс принадлежит к сервисному слою и выступает посредником между классами TaskContorller и TaskRepository, он передает данные из первого во второй, обрабатывает результат и возвращает его на контроллерный слой.

Метод findById() принимает на вход параметр id типа UUID, представляющий собой идентификатор записи искомой задачи. Этот параметр передается в метод findById() класса TaskRepository для поиска записи задачи в базе данных, используя предоставленный идентификатор. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо объект TaskDto, содержащий всю информацию об искомой задаче.

Ниже представлен листинг метода findById().

```
     public Result<TaskDto> findById(UUID id) {
    try {
        Optional<Task> task = taskRepository.findById(id);
        if (task.isEmpty())
            return Result.error("Task was not found", "404");
        else
            return Result.ok(task.get().mapToDto());
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error finding task", "500");
    }
}
```

Метод findByUser() принимает на вход параметр id типа UUID, представляющий собой идентификатор пользователя, для которого требуется

найти все назначенные на него задачи. Этот параметр передается в метод findByUser() класса TaskRepository для поиска записи задачи в базе данных, используя предоставленный идентификатор. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо список объектов List<TaskDto>, содержащий все задачи, для нужного пользователя.

Ниже представлен листинг метода findByUser().

```
     public Result<List<TaskDto>> findByUserId(UUID id) {
    try {
        List<Task> tasks = taskRepository.findByUserId(id);
        return
Result.ok(tasks.stream().map(Task::mapToDto).collect(Collectors.toLi
st()));
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error finding tasks by employee",
"500");
    }
}
```

Метод findByName() принимает на вход параметр name типа String, представляющий собой название искомой задачи или его часть. Этот параметр передается в метод findByName() класса TaskRepository для поиска в базе данных всех записей задач, в названии которых присутствует строка name. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо список объектов List<TaskDto>, содержащий все найденные задачи, соответствующие условиям поиска.

Ниже представлен листинг метода findByName().

```
     public Result<List<TaskDto>> findByName(String name) {
    try {
        List<Task> tasks = taskRepository.findByName(name + '%');
        return
Result.ok(tasks.stream().map(Task::mapToDto).collect(Collectors.toLi
st()));
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error finding Tasks by first name",
"500");
    }
}
```

Метод create() принимает на вход объект Task, содержащий всю информацию о создаваемой задаче. Статус задачи устанавливается, как не начата, а дата создания устанавливается как текущее время. После чего Task передается в метод saveAndFlush() класса TaskRepository для сохранения в базе данных записи задачи. Результат выполнения вызванного метода обрабатывается, если у сохраненной задачи задан идентификатор проекта, к которому она принадлежит, то она записывается как часть проекта, по средствам методов findById() и saveAndFlush() класса ProjectRepository. На выход подается, либо сообщение об ошибке, либо сообщение об успешном сохранении задачи.

Ниже представлен листинг метода create().

```
     public Result<String> create(Task task) {
    try {
        task.setStatus(TaskStatus.NotStarted);
        task.setDateCreated(LocalDateTime.now());
        Task savedTask = taskRepository.saveAndFlush(task);
        if (savedTask.getProjectId() != null) {
            Project project =
projectRepository.findById(savedTask.getProjectId()).get();
            HashSet tasks = new HashSet<>();
            if (project.getTasks() != null) {
                tasks = ((HashSet) project.getTasks());
            }
            tasks.add(savedTask.getId());
            project.setTasks(tasks);
            projectRepository.saveAndFlush(project);
        }
        return Result.ok("OK");
    } catch (Exception e) {
        return Result.error("Failed to create task", "500");
    }
}
```

Метод update() принимает на вход объект TaskDto, содержащий новую информацию обновляемой задачи, и параметр id типа UUID. Последний передается в метод findById() класса TaskRepository для поиска записи изменяемой задачи в базе данных. Если таковая была найдена, то ее название, описание, приоритет, статус и конечная дата выполнения устанавливаются в значения содержащиеся в TaskDto. После чего полученный объект Task передается в метод saveAndFlush() класса TaskRepository для сохранения в базе

данных. Результат выполнения вызванного метода обрабатывается. На выход подается, либо сообщение об ошибке, либо сообщение об успешном сохранении задачи.

Ниже представлен листинг метода update().

```
 public Result<String> update(UUID id, TaskDto taskDto) {
    try {
        Optional<Task> taskRead = taskRepository.findById(id);
        if (taskRead.isEmpty())
            return Result.error("Task was not found", "404");
        Task task = taskRead.get();
        task.setName(taskDto.getName());
        task.setDescription(taskDto.getDescription());
        task.setPriority(taskDto.getPriority());
        task.setStatus(taskDto.getStatus());
        task.setDeadline(taskDto.getDeadline());
        taskRepository.saveAndFlush(task);
        return Result.ok("Update ok");
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Failed to update task", "500");
    }
}
```

Метод delete() принимает на вход параметр id типа UUID представляющий собой идентификатор удаляемой задачи. Перед тем как удалить задачу, нужно удалить ее из проекта, если она в таковом числится. Для этого id передается в метод findById() класса TaskRepository для поиска записи удаляемой задачи, из которой при наличии берется projectId. Он для поиска проекта передается в метод findById() класса ProjectRepository. Из найденного проекта удаляемая задача исключается, и измененная запись проекта сохраняется в базе данных. Далее вызываются методы deleteAllByTaskId() из классов AttachmentRepository и CommentRepository, удаления всех комментариев и приложенных файлов, принадлежащих к удаляемой задаче. После чего id передается в метод deleteById() класса TaskRepository предназначенный для удаления задач. Результат выполнения его обрабатывается. На выход подается, либо сообщение об ошибке, либо сообщение об успешном удалении задачи.

Ниже представлен листинг метода delete().

```
 public Result<String> delete(UUID id) {
    try {
```

```
        Task task = taskRepository.findById(id).get();
        if (task.getProjectId() != null) {
            Project project =
projectRepository.findById(task.getProjectId()).get();
            if (project.getTasks().size() == 1) {
                project.setTasks(null);
            } else {
                HashSet tasks = (HashSet) project.getTasks();
                tasks.remove(task.getId());
                project.setTasks(tasks);
            }
            projectRepository.saveAndFlush(project);
        }
        commentRepository.deleteAllByTaskId(id);
        attachmentRepository.deleteAllByTaskId(id);
        taskRepository.deleteById(id);
        return Result.ok("Delete ok");
    } catch (Exception e) {
        return Result.error("Failed to delete task", "500");
    }
}
```

4.2 Вариант использования «Просмотр информации об аккаунте»

На рисунке 16 представлена диаграмма деятельности на вариант использования «Просмотр информации об аккаунте». Пользователю предоставляются следующие возможные действия:

– изменение данных аккаунта;

– удаление аккаунта;

– просмотр одного из проектов, на которые подписан пользователь, из выводимого списка.

Каждое возможное действие, запускает соответствующий метод, который делает запрос в базу данных и возвращает либо сообщение об ошибке, либо результат выполнения, после чего этот метод подает на выход полученные данные, которые затем отображаются пользователю.

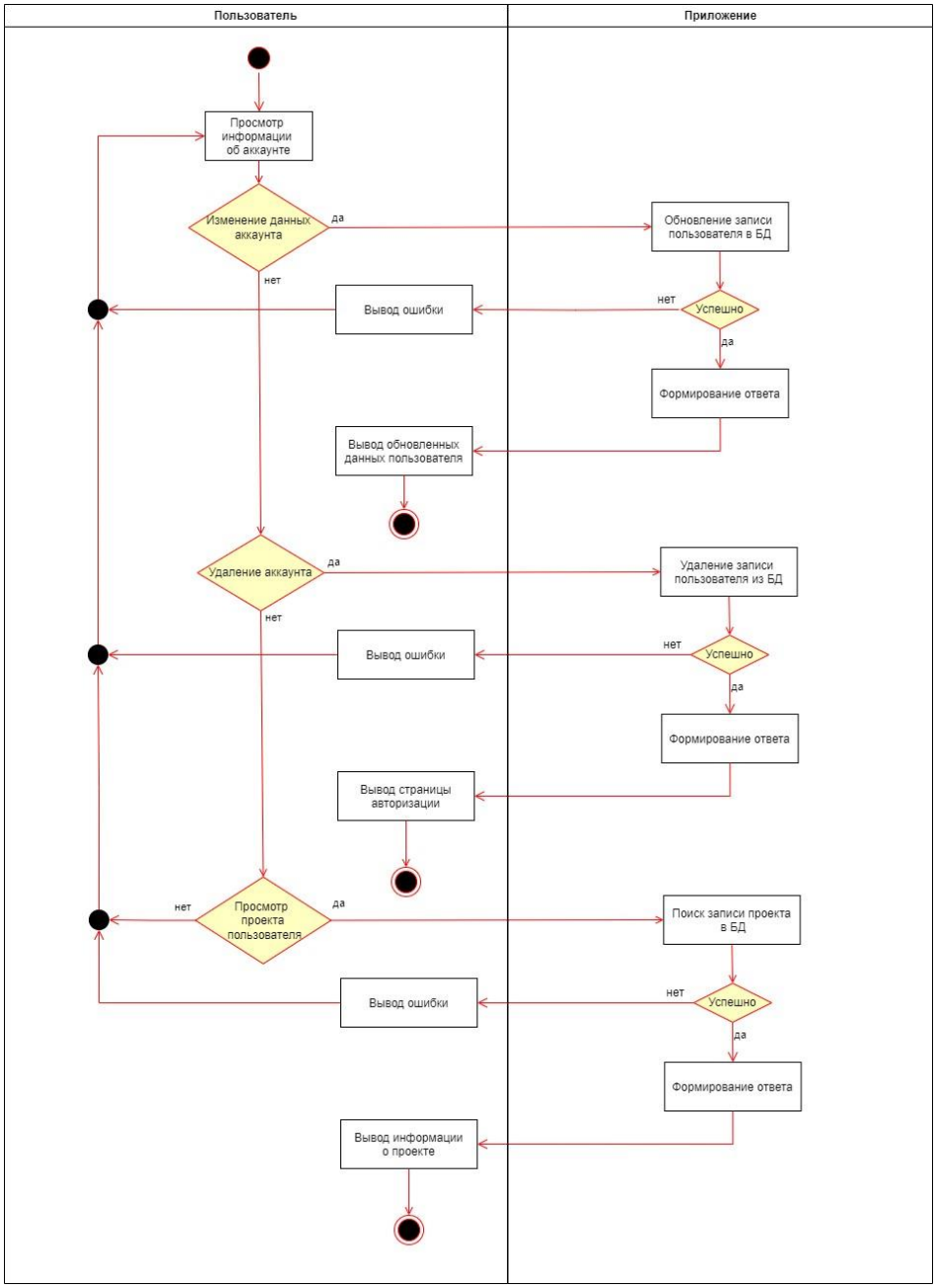


Рисунок 16 – UML-диаграмма деятельности на вариант использования «Просмотр информации об аккаунте»

Далее представлены основные классы, задействованные в рассматриваемом варианте использования.

4.2.1 UserController

Этот класс является контроллером и отвечает за обработку данных из приходящих к серверу запросов. Контроллер отвечает за все операции с данными пользователей и определяет, как и в каком виде они будут отображены.

Метод getUser() срабатывает когда на сервер приходит get-запрос по адресу «/user/{id}», где id – это идентификатор записи, содержащей данные искомого пользователя. Среди передаваемых параметров также есть объект Principal. С его помощью определяется пользователь, отправивший запрос. Если его идентификатор совпадает с тем что передан как параметр, то данные этого пользователя считаются как искомые. В противном случае, параметр id передается в метод findById() класса UserService. Результатом его работы объект класса UserDto, содержащий нужную информацию об искомом пользователе. Для искомого пользователя выполняется поиск всех проектов, в которых он участвует. Все найденные данные возвращаются методом, для отображения на веб-страницах.

Ниже представлен листинг метода getUser().

```
     @GetMapping("/{id}")
public String getUser(@PathVariable("id") UUID userId, Principal
principal, Model model) {
    Result<UserDto> userRead =
userService.findByPrincipal(principal);
    if (userRead.isError()) {
        model.addAttribute("message", userRead.getMessage());
        return userRead.getCode();
    }
    UserDto userDto = userRead.getObject();

    model.addAttribute("userDto", userDto);
    if (userDto.getId() == userId) {
        Set<UUID> projects = userDto.getProjects();
        List<ProjectDto> projectList = Collections.emptyList();
        if (projects != null) {
            projectList = projects.stream()
                    .map(id ->
projectService.findById(id).getObject())
                    .toList();
        }
        model.addAttribute("projectList", projectList);
        model.addAttribute("userDtoFound", userDto);
    } else {
        Result<UserDto> userDtoRead = userService.findById(userId);
        if (userDtoRead.isError()) {
            model.addAttribute("message", userDtoRead.getMessage());
            return userDtoRead.getCode();
        }
        UserDto userDtoFound = userDtoRead.getObject();
```

```
        Set<UUID> projects = userDtoFound.getProjects();
        List<ProjectDto> projectList = Collections.emptyList();
        if (projects != null) {
            projectList = projects.stream()
                    .map(id ->
projectService.findById(id).getObject())
                    .toList();
        }
        model.addAttribute("projectList", projectList);
        model.addAttribute("userDtoFound", userDtoFound);

        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
    }
    return "user/index";
}
```

Метод update() срабатывает когда на сервер приходит put-запрос по адресу «/user/{id}», где id – это идентификатор записи обновляемой задачи. Также среди передаваемых параметров есть объект UserDto, содержащий новые пользовательские данные. Объекты UserDto и id передаются в метод update() класса UserService. Результатом его работы является информация об успешном изменении задачи, либо информация о возникшей ошибке. После успешного изменения происходит перенаправление на страницу с обновленными данными пользователя.

Ниже представлен листинг метода update().

```
     @PutMapping("/{id}")
public String update(@ModelAttribute("user") UserDto user,
@PathVariable("id") UUID id, Model model) {
    Result<String> upload = userService.update(id, user);
    if (upload.isError()) {
        model.addAttribute("message", upload.getMessage());
        return upload.getCode();
    } else
        return "redirect:/user/" + id;
}
```

Метод delete() срабатывает когда на сервер приходит delete-запрос по адресу «/task/{id}», где id – это идентификатор записи обновляемой задачи. Объект id передается в метод delete() класса TaskService. Результатом его работы является информация об успешном удалении задачи, либо информация о возникшей ошибке. После успешного удаления пользователя происходит перенаправление на

домашнюю страницу, если удаленный пользователь не являлся инициатором операции, иначе перенаправление идет на страницу авторизации.

Ниже представлен листинг метода delete().

```
     @DeleteMapping("/{id}")
public String delete(@PathVariable("id") UUID id, Model model) {
    Result<String> delete = userService.delete(id);
    if (delete.isError()) {
        model.addAttribute("message", delete.getMessage());
        return delete.getCode();
    } else {
        model.addAttribute("message", delete.getObject());
        return "redirect:/";
    }
}
```

4.2.2 UserService

Этот класс принадлежит к сервисному слою и выступает посредником между классами UserContorller и UserRepository, он передает данные из первого во второй, обрабатывает результат и возвращает его на контроллерный слой.

Метод findById() принимает на вход параметр id типа UUID, представляющий собой идентификатор записи об искомом пользователе. Этот параметр передается в метод findById() класса UserRepository для поиска записи о пользователе в базе данных, используя предоставленный идентификатор. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо объект UserDto, содержащий всю информацию об искомом пользователе.

Ниже представлен листинг метода findById().

```
     public Result<UserDto> findById(UUID id) {
    try {
        Optional<User> user = userRepository.findById(id);
        if (user.isEmpty())
            return Result.error("User was not found", "404");
        else
            return Result.ok(user.get().mapToDto());
    } catch (Exception e) {
        return Result.error("Error finding user", "500");
    }
}
```

Метод findByPrincipal() принимает на вход объект Principal, содержащий информацию о пользователе, личность которого нужно подтвердить и получить

информацию о нем. Это производится путем передачи параметра name, из входного объекта, в метод findByLogin() класса UserRepository предназначенный для поиска в базе данных записи о пользователе по его логину. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо объект UserDto, содержащий всю информацию о нужном пользователе.

Ниже представлен листинг метода findByPrincipal().

```
     public Result<UserDto> findByPrincipal(Principal principal) {
    try {
        Optional<User> user =
userRepository.findByLogin(principal.getName());
        if (user.isEmpty())
            return Result.error("User was not found", "404");
        else
            return Result.ok(user.get().mapToDto());
    } catch (Exception e) {
        return Result.error("Error finding user", "500");
    }
}
```

Метод userExists() принимает на вход строку login, в которую записан логин пользователя, существование которого нужно подтвердить. В метод findByLogin() класса UserRepository передается login. Результат выполнения вызванного метода обрабатывается и на выход передается Boolean, значение true которого означает существование пользователя, а false обратное.

Ниже представлен листинг метода userExists().

```
     public Boolean userExists(String login) {
    return userRepository.findByLogin(login).isPresent();
}
```

Метод findByName() принимает на вход параметр name типа String, представляющий собой имя искомого пользователя или его часть. Этот параметр передается в метод findByName() класса UserRepository для поиска в базе данных всех записей о пользователях, имена которых содержат строку name. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо список объектов List<UserDto>, содержащий всех найденных пользователей, соответствующих условиям поиска.

Ниже представлен листинг метода findByName().

```
     public Result<List<UserDto>> findByName(String name) {
    try {
        List<User> users = userRepository.findByName(name + '%');
        return
Result.ok(users.stream().map(User::mapToDto).collect(Collectors.toLi
st()));
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error finding Users by first name",
"500");
    }
}
```

Метод findBySurename() принимает на вход параметр surename типа String, представляющий собой фамилию искомого пользователя или ее часть. Этот параметр передается в метод findBySurename() класса UserRepository для поиска в базе данных всех записей о пользователях, фамилии которых содержат строку surename. Результат выполнения вызванного метода обрабатывается и на выход подается, либо сообщение об ошибке, либо список объектов List<UserDto>, содержащий всех найденных пользователей, соответствующих условиям поиска.

Ниже представлен листинг метода findBySurename().

```
     public Result<List<UserDto>> findBySurename(String surename) {
    try {
        List<User> users = userRepository.findBySurename(surename +
'%');
        return
Result.ok(users.stream().map(User::mapToDto).collect(Collectors.toLi
st()));
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error finding Users by middle name",
"500");
    }
}
```

Метод create() принимает на вход объект User, содержащий всю информацию о создаваемом пользователе. Создаваемой записи присваивается роль «ROLE_USER», переданный пароль шифруется. После чего User передается в метод saveAndFlush() класса UserRepository для сохранения в базе данных. В случае успешного создания записи о пользователе, будет создан и записан в базу данных пустой объект ContactInfo, предназначенный для хранения контактных

данных пользователя. На выход метода подается, либо сообщение об ошибке, либо объект, содержащий данные созданного пользователя, User.

Ниже представлен листинг метода create().

```
     @Transactional
public Result<User> create(User user) {
    try {
        Optional<Role> result =
roleRepository.findByName("ROLE_USER");
        Role role = result.get();
        user.setRole(role);
        user.setPassword(new
BCryptPasswordEncoder().encode(user.getPassword()));
        User savedUser = userRepository.saveAndFlush(user);
        ContactInfo contactInfo = new ContactInfo();
        contactInfo.setUserr(savedUser);
        ContactInfo savedContactInfo =
contactInfoRepository.saveAndFlush(contactInfo);
        return Result.ok(savedUser);
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Error creating User", "500");
    }
}
```

Метод update() принимает на вход объект UserDto, содержащий новые пользовательские данные, и параметр id типа UUID. Последний передается в метод findById() класса UserRepository для поиска записи изменяемого пользователя в базе данных. Если таковая была найдена, то имя, фамилия, логин и пароль устанавливаются в значения содержащиеся в UserDto. После чего полученный объект User передается в метод saveAndFlush() класса UserRepository для сохранения в базе данных. Результат выполнения вызванного метода обрабатывается. На выход подается, либо сообщение об ошибке, либо сообщение об успешном сохранении пользователя.

Ниже представлен листинг метода update().

```
     public Result<String> update(UUID id, UserDto userDto) {
    try {
        Optional<User> userRead = userRepository.findById(id);
        if (userRead.isEmpty())
            return Result.error("User was not found", "404");
        User user = userRead.get();
        user.setName(userDto.getName());
        if (!user.getPassword().equals(""))
```

```
            user.setPassword(new
BCryptPasswordEncoder().encode(userDto.getPassword()));
        user.setSurename(userDto.getSurename());
        user.setLogin(userDto.getLogin());
        userRepository.saveAndFlush(user);
        return Result.ok("Update ok");
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return Result.error("Failed to update user", "500");
    }
}
```

Метод delete() принимает на вход параметр id типа UUID представляющий собой идентификатор удаляемой задачи. Перед тем как удалить пользователя, нужно удалить все задачи, созданные им, его контактные данные и все комментарии оставленные им. Это осуществляется вызовом методом deleteAllByUserId() из классов AttachmentRepository, CommentRepository, TaskRepository, ContactInfoRepository. После чего id передается в метод deleteById() класса UserRepository предназначенный для удаления задач. Результат его выполнения обрабатывается. На выход подается, либо сообщение об ошибке, либо сообщение об успешном удалении пользователя.

Ниже представлен листинг метода delete().

```
 public Result<String> delete(UUID id) {
    try {
        attachmentRepository.deleteAllByUserId(id);
        commentRepository.deleteAllByUserId(id);
        taskRepository.deleteAllByUserId(id);
        workTimeRepository.deleteAllByUserId(id);
        contactInfoRepository.deleteAllByUserId(id);
        userRepository.deleteById(id);
        return Result.ok("Delete ok");
    } catch (Exception e) {
        return Result.error("Failed to delete user", "500");
    }
}
```

5 Тестирование

Для тестирования работоспособности приложения были написаны юнит-тесты, для основных сервисных классов. На рисунке 17 представлены результаты выполнения всех методов тестирования.

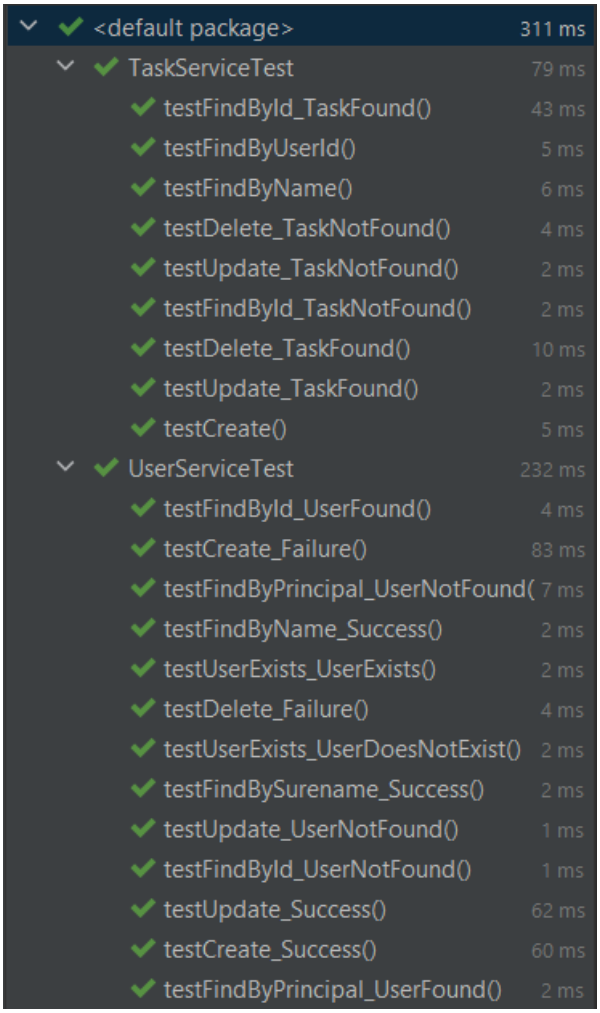


Рисунок 17 – Результаты выполнения методов тестирования из соответствующих классов

Тесты были написаны с использованием Mockito — фреймворка для тестирования приложений, который позволяет легко и быстро подменять реальные объекты программы «пустышками» [9]. При их написании сначала подготавливались тестовые данные, затем с помощью функции when(), из Mockito, устанавливалось соответствие между вызовами из сервисного метода,

обращавшихся на слой репозиториев, и подготовленными тестовыми данными. Т.е. при вызове метода findById(), класса TaskRepository, из TaskService, туда вернется заранее созданный в тесте объект TaskDto. После чего сервисный метод вернет свой результат, который будет проверен на соответствие с изначальными данными.

Ниже представлен листинг одного из методов класса TaskServiceTest — testFindById_TaskFound().

```
     @Test
public void testFindById_TaskFound() {
    UUID taskId = UUID.randomUUID();
    Task task = new Task();
    task.setId(taskId);
    User user = new User();
    user.setContactInfo(new ContactInfo());
    task.setUser(user);

when(taskRepository.findById(taskId)).thenReturn(Optional.of(task));

    Result<TaskDto> result = taskService.findById(taskId);

    Assertions.assertEquals(taskId, result.getObject().getId());
}
```

6 Безопасность жизнедеятельности

При взаимодействии с компьютером необходимо соблюдать меры безопасности, чтобы не было неприятных последствий для здоровья после использования. Если пользователь ПК будет пренебрегать техникой безопасности, то проблемы со здоровьем будут неизбежны.

6.1 Вред здоровью при работе с ПК

Проблемы со здоровьем в процессе взаимодействия с компьютером имеют свойство накапливаться, то есть здоровье ухудшается не сразу, а постепенно. Наиболее опасными факторами являются:

– нагрузка на суставы. При неправильной посадке за компьютером, на суставы будет идти большая нагрузка, а именно кистей рук, из-за чего может произойти даже онемение рук;

– нагрузка на спину. Опять же, если принять неправильно положение осанки при работе за компьютером, может появиться искривление позвоночника и даже грыжа межпозвоночного диска;

– нагрузка на органы зрения. При долгой работе с мониторами будет ухудшаться здоровья глаз, впоследствии могут появиться синдромы «красного глаза» и «сухого глаза»;

– нагрузка на психику и нервную. При долгом взаимодействии с ПК, у пользователя могут возникать проблемы с памятью, а также может появиться депрессия и даже бессонница.

6.2 Снижение вреда для здоровья при работе с ПК

При соблюдении простых правил можно избежать серьезных проблем со здоровьем, для этого нужно:

– соблюдать необходимое расстояние до монитора. Правильным расстоянием до дисплея является расстояние, при котором пользователь может коснуться указательным пальцем до верхнего края монитора;

– соблюдать оптимальный угол экрана. Оптимальным углом является наклон монитора в 30 градусов;

– клавиатуру ставить в нужное положение. Клавиатура должна находиться на расстоянии 20-30 сантиметров от края стола;

– делать перерыв. Примерно каждый час необходимо делать разминку для органов зрения, а именно оторваться от монитора и делать гимнастику для глаз: рисовать различные фигуры глазами, фокусировать внимание на дальние объекты и переводить фокус с одних предметов на другие. Также примерно каждые полтора-два часа необходимо выполнять зарядку;

– выбрать правильный стул или кресло. При выборе данной мебели необходимо приобрести такую, которая будет обеспечивать прямую осанку, при которой спина будет опираться в спинку.

6.3 Техника безопасности при работе с компьютером

При работе с компьютером необходимо соблюдать следующие правила, которые связаны с техникой безопасности:

– не оставлять компьютер без присмотра на длительное время;

– не использовать компьютер, который работает от питания без заземления;

– не работать за ПК, у которого есть повреждение в системном блоке или в его периферийных устройствах;

– не включать компьютер, если в помещении слишком высокая влажность;

– не класть на корпус компьютера или монитора посторонние предметы.

Заключение

В ходе выполнения данной выпускной квалификационной работы были получены навыки написания приложения для отслеживания задач, изучен стек технологий и библиотек Java и Spring Framework. Результатом работы является разработанное приложение «Трекер задач», которое называется TaskTracker, а также отчёт, в котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор его основных функциональных особенностей.

Разработанное приложение позволяет пользователю создавать, изменять и удалять задачи, объединять их в проекты, а также оставлять и удалять к ним комментарии и загружать, скачивать и удалять файлы, прикрепленные к задачам. После проведения тестирования было выяснено, что разработанное приложение соответствует всем поставленным в начале работы требованиям и работает корректно.

Список использованных источников

1. Что такое таск-трекер. [Электронный ресурс]/Режим доступа: https://blog.rt.ru/b2c/chto-takoe-task-treker.htm, свободный. (дата обращения 20.04.2024).

2. Клиент-серверная архитектура в картинках / Хабр. [Электронный ресурс]/Режим доступа: https://habr.com/ru/articles/495698/, свободный. (дата обращения 20.04.2024).

3. Обобщенный Model-View-Controller. [Электронный ресурс]/Режим доступа: https://rsdn.org/article/patterns/generic-mvc.xml#EEGAC, свободный. (дата обращения 21.04.2024).

4. Язык Java: что это, зачем нужен, плюсы и минусы - применение и сравнение с другими языками программирования, с чего начать. [Электронный ресурс]/Режим доступа: https://practicum.yandex.ru/blog/chto-takoe-java-dlya-nachinayuschih/, свободный. (дата обращения 23.04.2024).

5. Что такое Spring Framework? От внедрения зависимостей до Web MVC / Хабр. [Электронный ресурс]/Режим доступа: https://habr.com/ru/articles/490586/, свободный. (дата обращения 24.04.2024).

6. Spring Data JPA / Хабр. [Электронный ресурс]/Режим доступа: https://habr.com/ru/articles/435114/, свободный. (дата обращения 24.04.2024).

7. DBeaver Community | Free Universal Database Tool. [Электронный ресурс]/Режим доступа: https://dbeaver.io/, свободный. (дата обращения 25.04.2024).

8. Что такое Intellij IDEA и зачем он нужен - frontend и backend-разработка в IDE Intellij Idea. [Электронный ресурс]/Режим доступа: https://workspace.ru/tools/environment/intellij-idea/, свободный. (дата обращения 25.04.2024).

9. Mockito - что это за фреймворк для Java, примеры использования. [Электронный ресурс]/Режим доступа: https://blog.skillfactory.ru/glossary/mockito/, свободный. (дата обращения 26.04.2024).

Приложение А

(Обязательное)

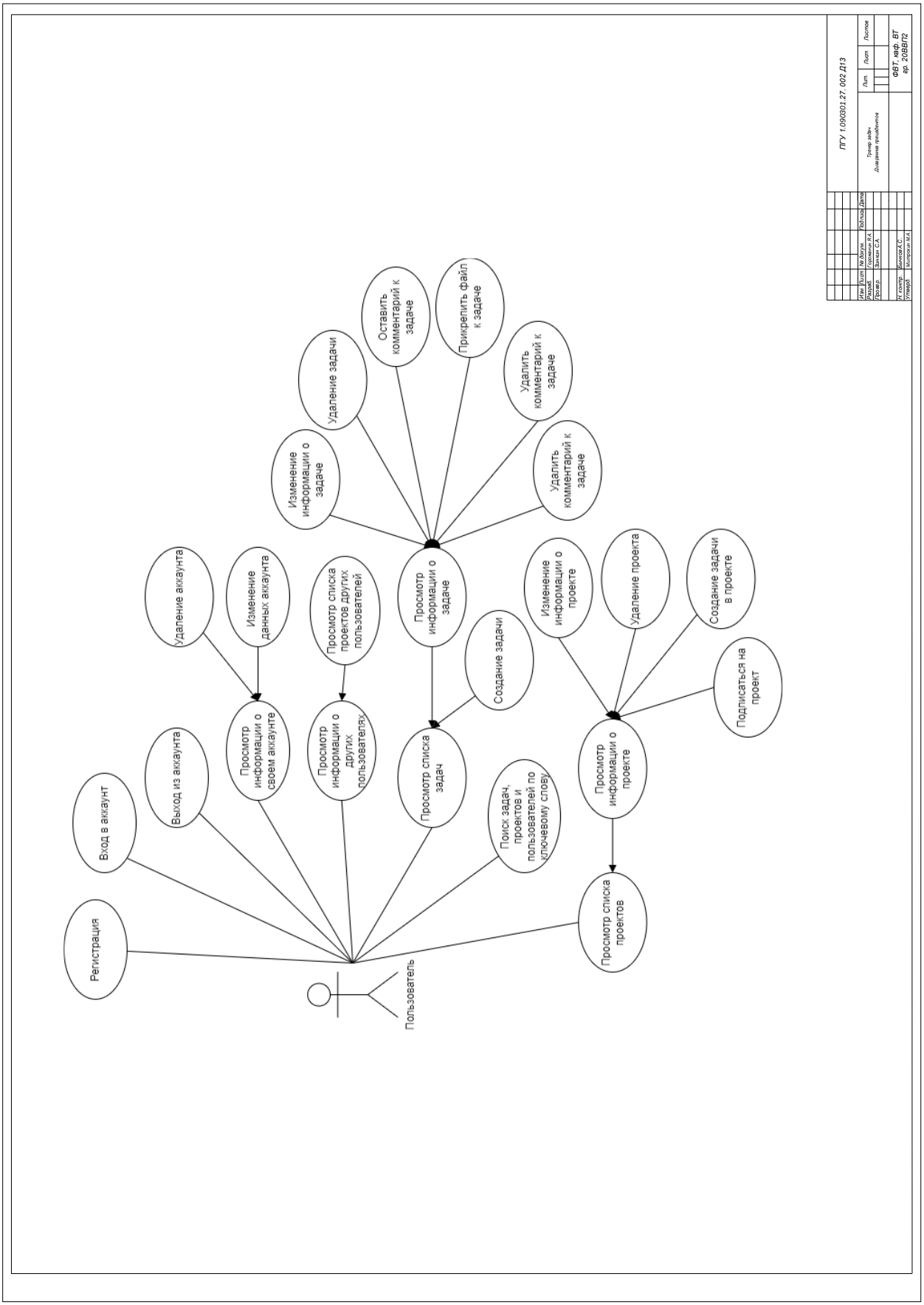
Трекер задач

UML-диаграммы

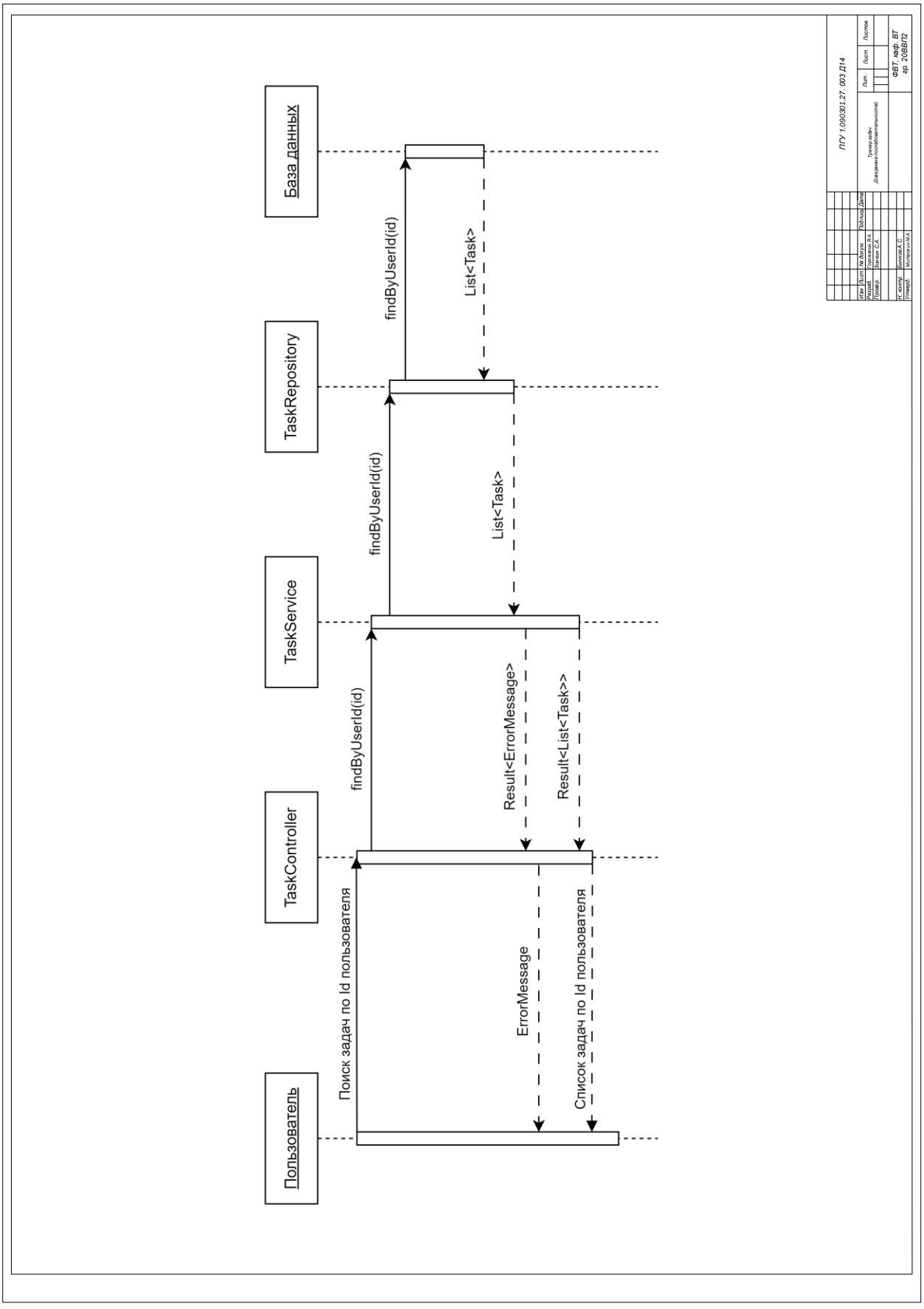
Приложение А.1 - Диаграмма классов

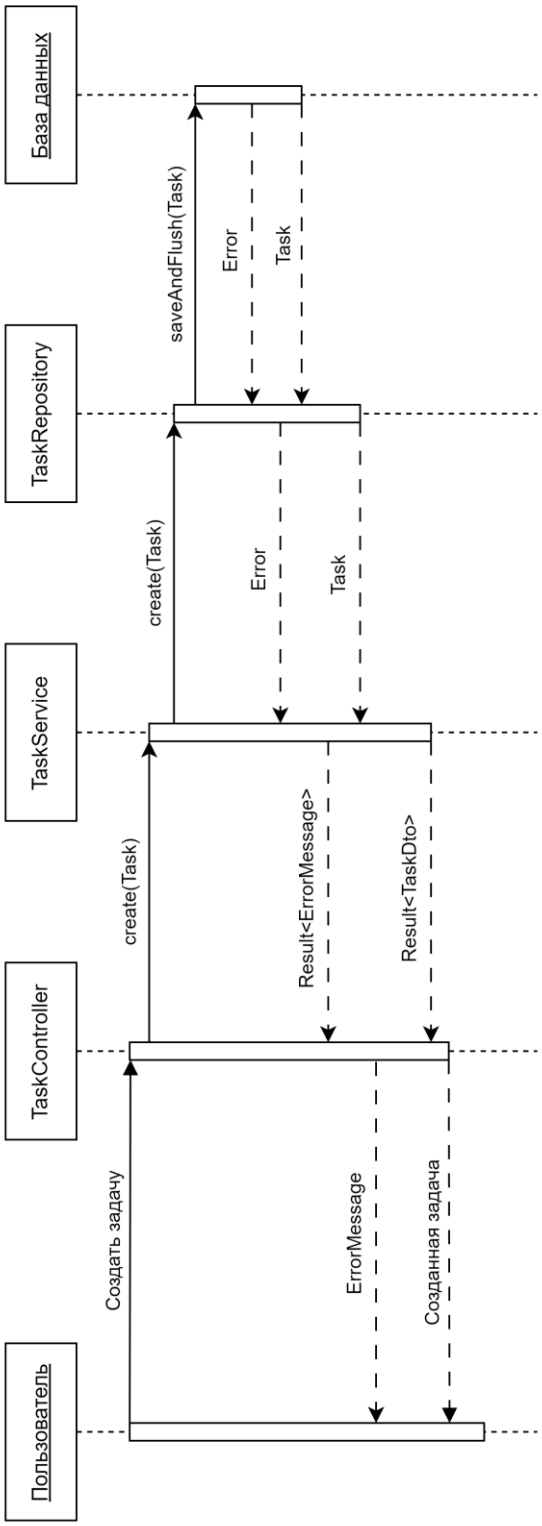
									ПГУ 1.090301.27.001 Д11		
									Тренер зала- Документ описание	Лит.	Лист	Листов
Изм.	Лист	№ докум.	Подпись	Дата							
Разраб.		Горлачев Л.А.									
Провер.		Белкин С.А.									
Т. контр.		Бакова А.С.							ФВТ, ивф. ВТ гр. 20ВВП2		
Утверд.		Матвеева М.А.									

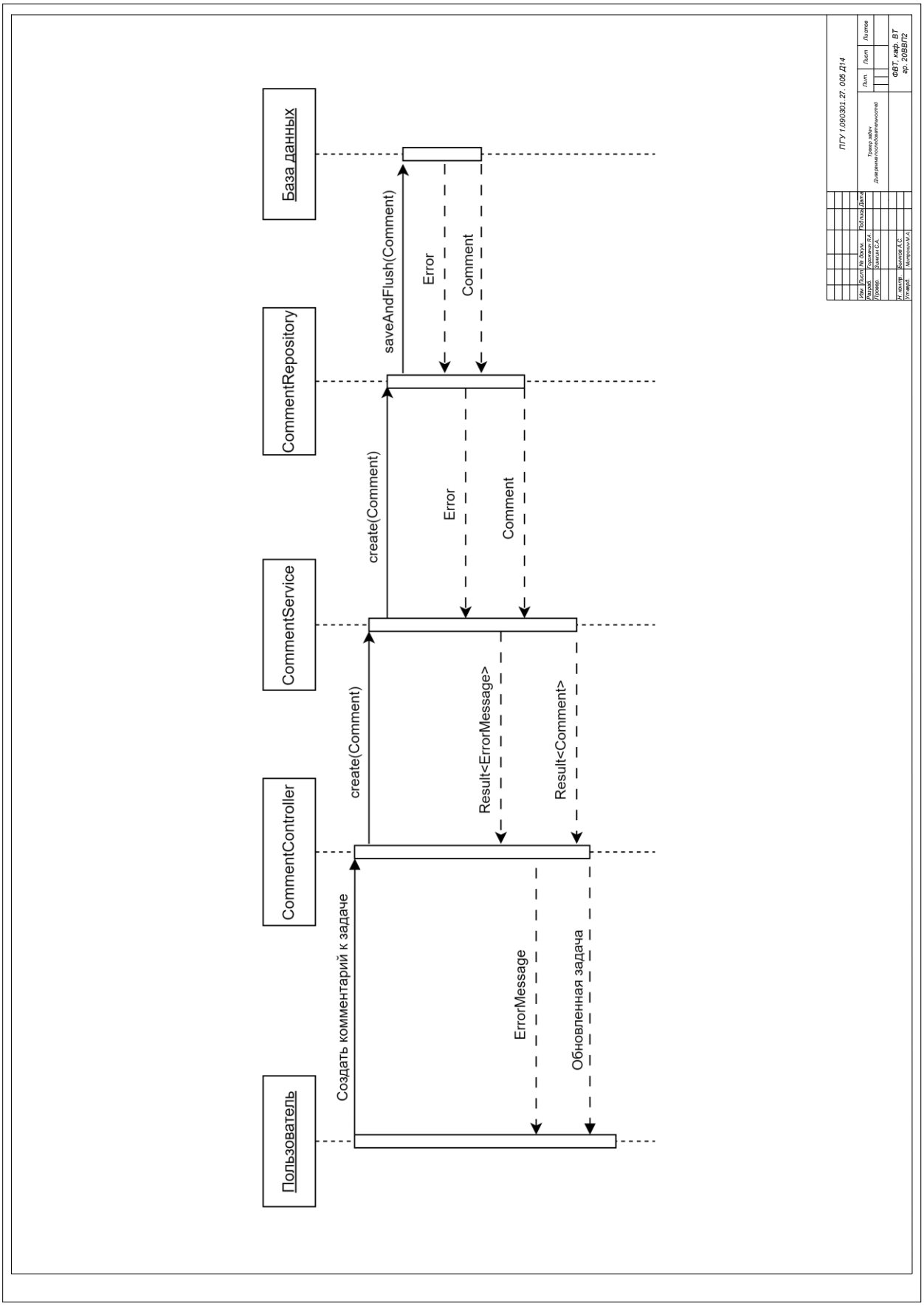
Приложение А.2 - Диаграмма прецедентов

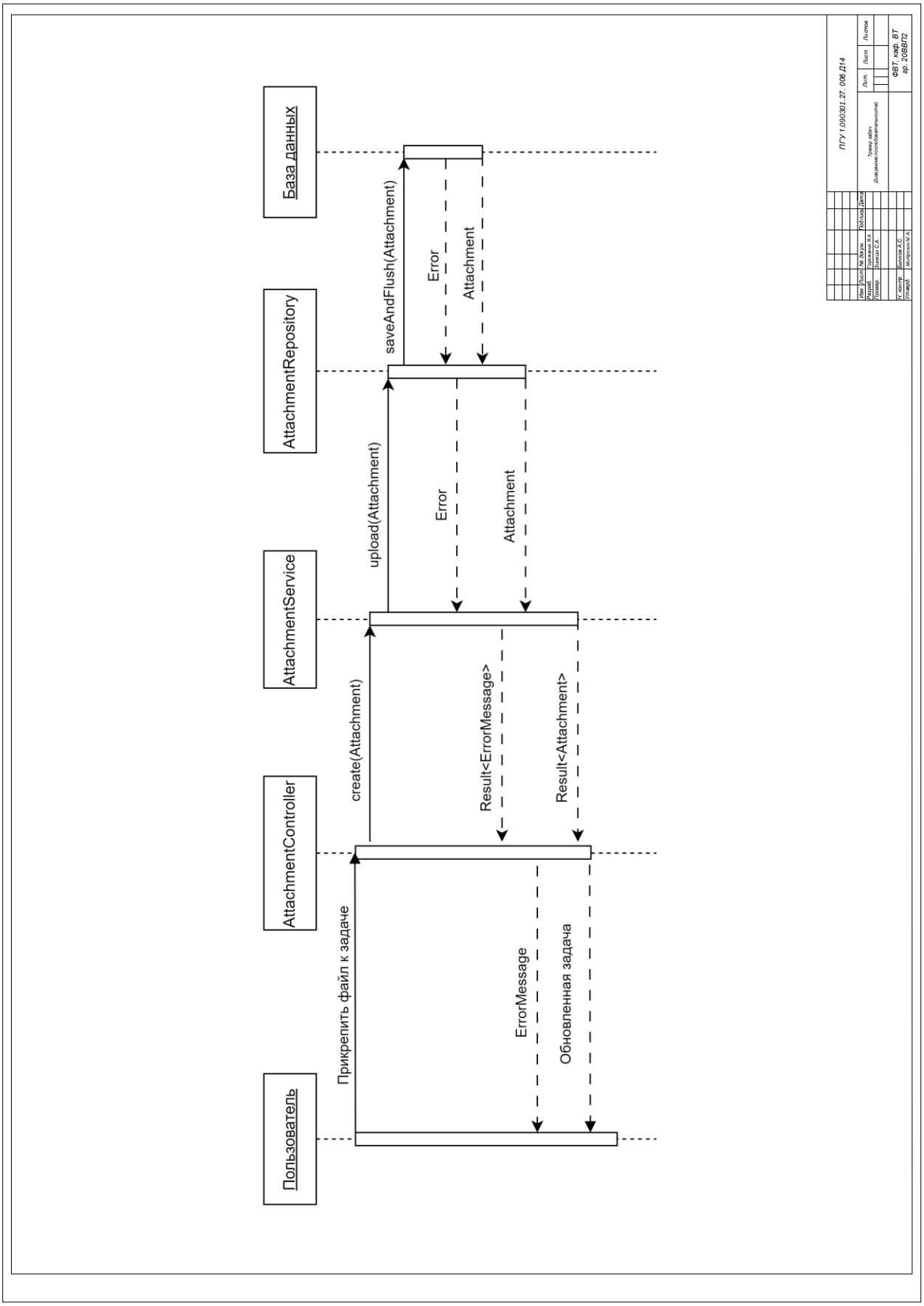


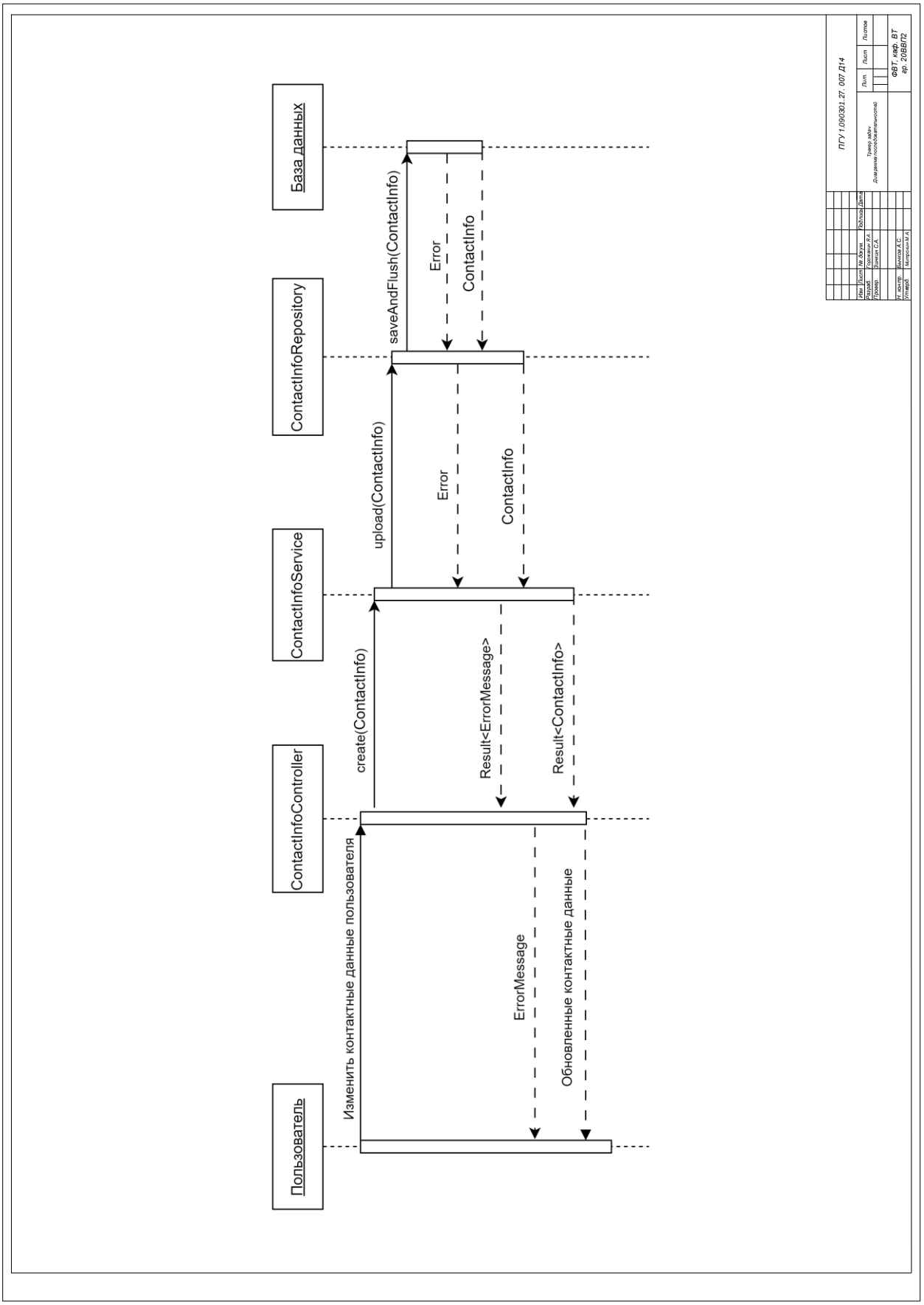
Приложение А.3 - Диаграммы последовательности



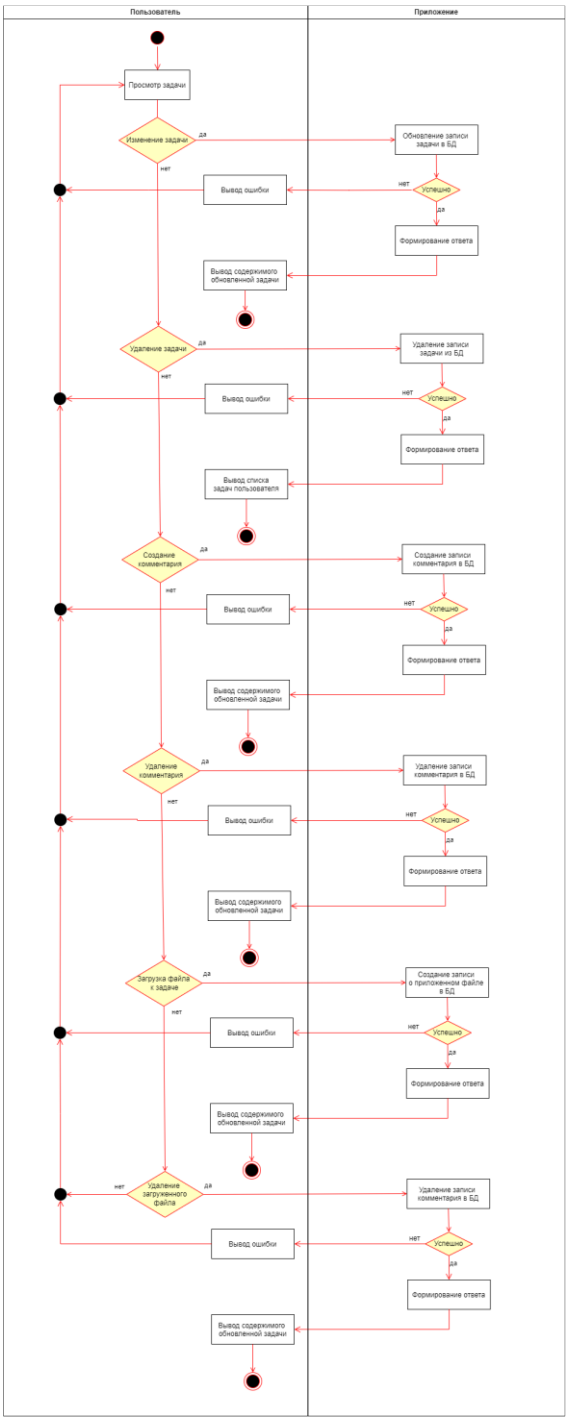

База данных
TaskRepository
TaskService
TaskController
Пользователь
saveAndFlush(Task)
Error
Task
create(Task)
Error
Task
create(Task)
Result<ErrorMessage>
Result<TaskDto>
Создать задачу
ErrorMessage
Созданная задача


ПГУ 1.090301.27. 005 Д14
Трекер задач
Дипломная пояснительная записка
ФВТ, каф. ВТ
гр. 20ВВП2
Разраб. Сорокин Р.А.
Провер. Белов С.А.
Н. контр. Белов А.С.
Утверд. Мезенцев К.Н.
База данных
CommentRepository
CommentService
CommentController
Пользователь
saveAndFlush(Comment)
Error
Comment
create(Comment)
Error
Comment
create(Comment)
Result<ErrorMessage>
Result<Comment>
Создать комментарий к задаче
ErrorMessage
Обновленная задача

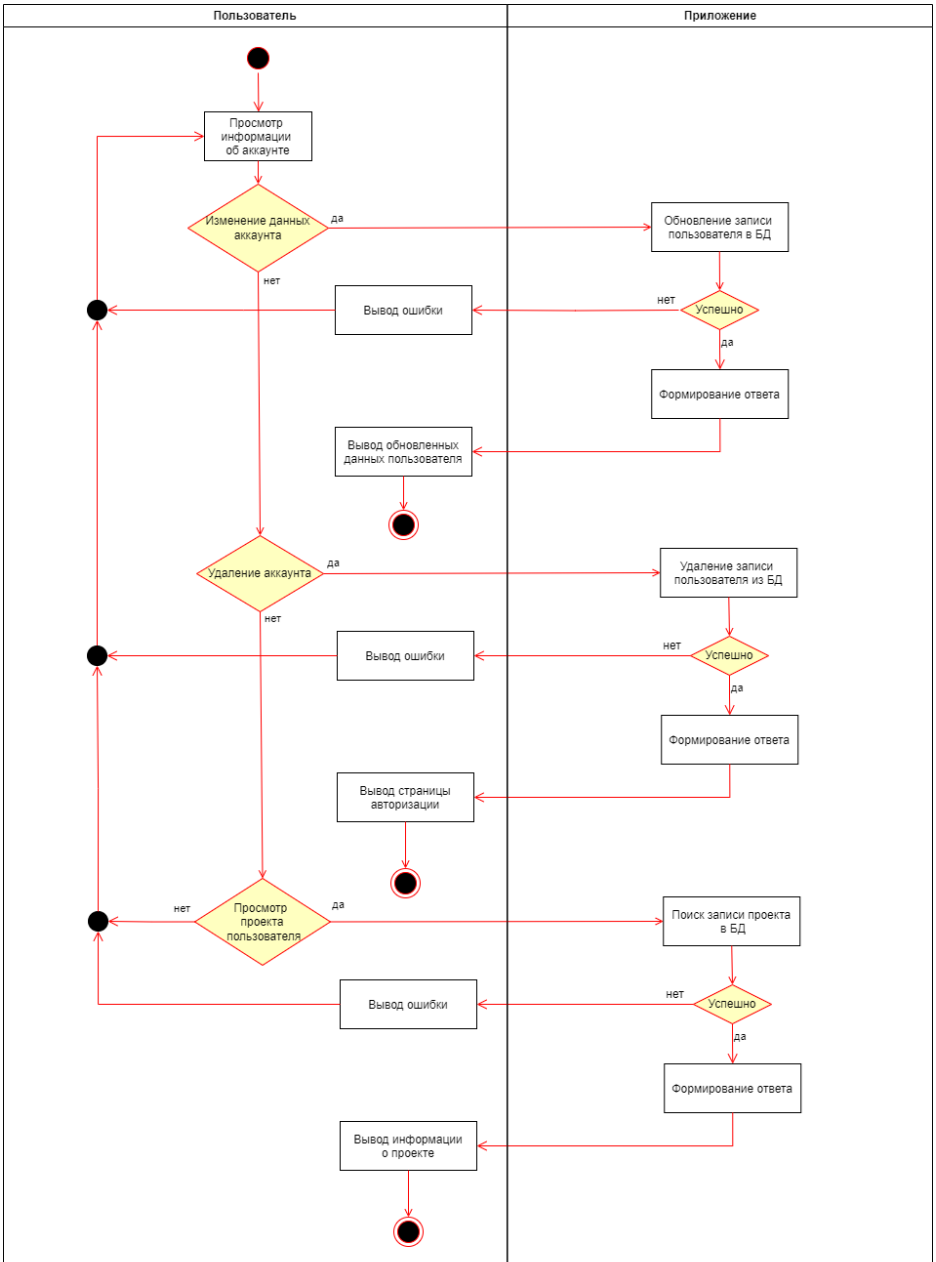

Пользователь
AttachmentController
AttachmentService
AttachmentRepository
База данных
Прикрепить файл к задаче
create(Attachment)
upload(Attachment)
saveAndFlush(Attachment)
Result<ErrorMessage>
Result<Attachment>
ErrorMessage
Обновленная задача
Error
Attachment
Error
Attachment
ПГУ 1.090301.27. 006 Д14
Лит.
Лист
Листов
ФВТ, каф. ВТ
гр. 20ВВП2
Разраб.
Провер.
Т. контр.
Н.контр.
Утверд.
Сорокин Р.А.
Белов С.А.
Волков А.С.
Мезенцев А.А.


Пользователь
ContactInfoController
ContactInfoService
ContactInfoRepository
База данных
Изменить контактные данные пользователя
create(ContactInfo)
upload(ContactInfo)
saveAndFlush(ContactInfo)
Error
ContactInfo
Result<ErrorMessage>
Result<ContactInfo>
ErrorMessage
Обновленные контактные данные

Приложение А.4 - Диаграммы деятельности

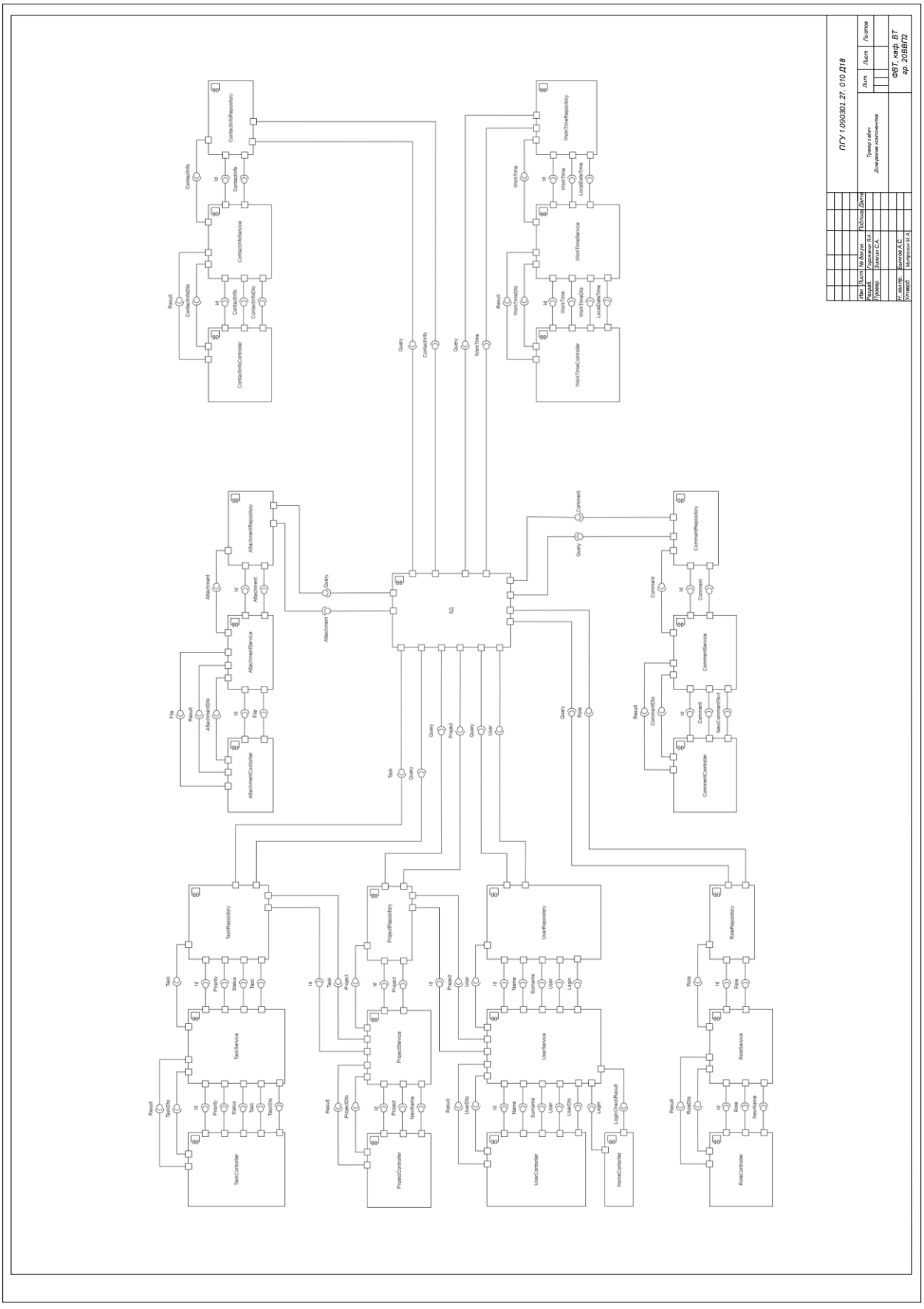


					ПГУ 1.090301.27. 008 Д17		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Гороханин Я.А.			Тренер задач Диаграмма деятельности	Лит.	Лист	Листов
Провер.		Зимкин С.А.						
Н. контр.		Быков А.С.				ФВТ, каф. ВТ гр. 20ВВП2		
Утверд.		Митрохин М.А.						



Изм.	Лист	№ докум.	Подпись	Дата	ПГУ1.090301.27.001 ПЗ	Лист 59

Приложение А.5 - Диаграмма компонентов



Приложение Б

(Обязательное)

Трекер задач

Листинг программы

Файл TaskController.java

```
package task_tracker.controller;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import task_tracker.domain.Attachment;
import task_tracker.domain.Comment;
import task_tracker.domain.Task;
import task_tracker.domain.User;
import task_tracker.dto.*;
import task_tracker.enums.Priority;
import task_tracker.enums.TaskStatus;
import task_tracker.service.*;
import task_tracker.utils.Result;

import java.security.Principal;
import java.util.List;
import java.util.UUID;

@Controller
@RequestMapping("/task")
@PreAuthorize("hasAuthority('ROLE_USER') ||
hasAuthority('ROLE_ADMIN')")
public class TaskController {
    private final TaskService taskService;
    private final UserService userService;
    private final ProjectService projectService;
    private final CommentService commentService;
    private final AttachmentService attachmentService;

    public TaskController(TaskService taskService, UserService
userService, ProjectService projectService, CommentService
commentService, AttachmentService attachmentService) {
        this.taskService = taskService;
        this.userService = userService;
        this.projectService = projectService;
        this.commentService = commentService;
        this.attachmentService = attachmentService;
    }

    @GetMapping("/{id}")
    public String getTask(@PathVariable("id") UUID taskId,
Principal principal, Model model) {
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
```

```
        }
        UserDto userDto = userRead.getObject();
        Result<TaskDto> taskRead = taskService.findById(taskId);
        if (taskRead.isError()) {
            model.addAttribute("message",
taskRead.getMessage());
            return taskRead.getCode();
        }

        TaskDto taskDto = taskRead.getObject();
        if (taskDto.getProjectId() != null) {
            Result<ProjectDto> projectRead =
projectService.findById(taskDto.getProjectId());
            if (projectRead.isError()) {
                model.addAttribute("message",
projectRead.getMessage());
                return projectRead.getCode();
            }
            ProjectDto projectDto = projectRead.getObject();
            model.addAttribute("project", projectDto);
        }

        Result<List<CommentDto>> taskComments =
commentService.findByTaskId(taskId);
        if (taskComments.isOk()) {
            List<CommentDto> comments =
taskComments.getObject();
            model.addAttribute("comments", comments);
        }

        Result<List<AttachmentDto>> attachmentRead =
attachmentService.findByTaskId(taskId);
        if (attachmentRead.isOk()) {
            List<AttachmentDto> attachmentDto =
attachmentRead.getObject();
            model.addAttribute("attachments", attachmentDto);
        }

        model.addAttribute("taskDto", taskDto);
        model.addAttribute("userDto", userDto);
        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
        model.addAttribute("newComment", new Comment());
        model.addAttribute("newAttachment", new Attachment());
        model.addAttribute("priorities", Priority.values());
        return "task/index";
    }

    @GetMapping("/new")
    public String getNew(Model model, Principal principal) {
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
```

```
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();
        model.addAttribute("userDto", userDto);
        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
        Task task = new Task();
        task.setPriority(Priority.Lowest);
        model.addAttribute(task);
        model.addAttribute("taskStatus", TaskStatus.values());
        model.addAttribute("priorities", Priority.values());
        return "task/new";
    }

    @GetMapping("/new/project/{id}")
    public String getNewForProject(Model model, Principal
principal, @PathVariable UUID id) {
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();
        model.addAttribute("userDto", userDto);
        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
        Task task = new Task();
        task.setProjectId(id);
        task.setPriority(Priority.Lowest);
        ProjectDto projectDto =
projectService.findById(id).getObject();
        List<UserDto> projectUsers = projectDto.getUsers()
                .stream()
                .map(userId ->
userService.findById(userId).getObject())
                .toList();
        model.addAttribute("projectUsers", projectUsers);
        model.addAttribute("task", task);
        model.addAttribute("taskStatus", TaskStatus.values());
        model.addAttribute("priorities", Priority.values());
        return "task/new";
    }

    @GetMapping("/update/{id}")
    public String getUpdateForm(@PathVariable("id") UUID id,
Model model, Principal principal) {
        Result<TaskDto> taskRead = taskService.findById(id);
```

```
        if (taskRead.isError()) {
            model.addAttribute("message",
taskRead.getMessage());
            return taskRead.getCode();
        }
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();
        model.addAttribute("userDto", userDto);
        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
        model.addAttribute("task", taskRead.getObject());
        model.addAttribute("taskStatus", TaskStatus.values());
        model.addAttribute("priorities", Priority.values());
        return "task/update";
    }

    @PostMapping("/")
    public String create(@ModelAttribute("task") Task task,
Model model, Principal principal) {
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        if (task.getProjectId()==null){
            UserDto userDto = userRead.getObject();
            User user = new User();
            user.setId(userDto.getId());
            task.setUser(user);
        }
        Result<String> savedTask = taskService.create(task);
        if (savedTask.isError()) {
            model.addAttribute("message",
savedTask.getMessage());
            return savedTask.getCode();
        } else {
            model.addAttribute("message", "Task create ok");
            if (task.getProjectId() != null) {
                return "redirect:/project/" +
task.getProjectId();
            }
            return "redirect:/";
        }
    }
```

```
    @PutMapping("/{id}")
    public String update(@PathVariable("id") UUID id,
@ModelAttribute("task") TaskDto task, Model model) {
        Result<String> upload = taskService.update(id, task);
        if (upload.isError()) {
            model.addAttribute("message", upload.getMessage());
            return upload.getCode();
        } else
            return "redirect:/task/" + id;
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable("id") UUID id, Model
model) {
        Result<String> delete = taskService.delete(id);
        if (delete.isError()) {
            model.addAttribute("message", delete.getMessage());
            return delete.getCode();
        } else {
            model.addAttribute("message", delete.getObject());
            return "redirect:/";
        }
    }
}
```

Файл UserController.java

```
package task_tracker.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import task_tracker.dto.ProjectDto;
import task_tracker.dto.UserDto;
import task_tracker.service.ProjectService;
import task_tracker.service.UserService;
import task_tracker.utils.Result;

import java.security.Principal;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import java.util.UUID;

@Controller
@RequiredArgsConstructor
@RequestMapping("/user")
public class UserController {
    private final UserService userService;
```

```
    private final ProjectService projectService;

    @GetMapping("/{id}")
    public String getUser(@PathVariable("id") UUID userId,
Principal principal, Model model) {
        Result<UserDto> userRead =
userService.findByPrincipal(principal);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();

        model.addAttribute("userDto", userDto);
        if (userDto.getId() == userId) {
            Set<UUID> projects = userDto.getProjects();
            List<ProjectDto> projectList =
Collections.emptyList();
            if (projects != null) {
                projectList = projects.stream()
                        .map(id ->
projectService.findById(id).getObject())
                        .toList();
            }
            model.addAttribute("projectList", projectList);
            model.addAttribute("userDtoFound", userDto);
        } else {
            Result<UserDto> userDtoRead =
userService.findById(userId);
            if (userDtoRead.isError()) {
                model.addAttribute("message",
userDtoRead.getMessage());
                return userDtoRead.getCode();
            }
            UserDto userDtoFound = userDtoRead.getObject();
            Set<UUID> projects = userDtoFound.getProjects();
            List<ProjectDto> projectList =
Collections.emptyList();
            if (projects != null) {
                projectList = projects.stream()
                        .map(id ->
projectService.findById(id).getObject())
                        .toList();
            }
            model.addAttribute("projectList", projectList);
            model.addAttribute("userDtoFound", userDtoFound);

            String searchQuery = "";
            model.addAttribute("searchQuery", searchQuery);
        }
```

```
        return "user/index";
    }

    @GetMapping("/update/{id}")
    public String getUpdateForm(@PathVariable("id") UUID id,
Model model) {
        Result<UserDto> userRead = userService.findById(id);
        if (userRead.isError()) {
            model.addAttribute("message",
userRead.getMessage());
            return userRead.getCode();
        }
        UserDto userDto = userRead.getObject();
        userDto.setPassword("");
        model.addAttribute("user", userDto);
        String searchQuery = "";
        model.addAttribute("searchQuery", searchQuery);
        return "user/update";
    }

    @PutMapping("/{id}")
    public String update(@ModelAttribute("user") UserDto user,
@PathVariable("id") UUID id, Model model) {
        Result<String> upload = userService.update(id, user);
        if (upload.isError()) {
            model.addAttribute("message", upload.getMessage());
            return upload.getCode();
        } else
            return "redirect:/user/" + id;
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable("id") UUID id, Model
model) {
        Result<String> delete = userService.delete(id);
        if (delete.isError()) {
            model.addAttribute("message", delete.getMessage());
            return delete.getCode();
        } else {
            model.addAttribute("message", delete.getObject());
            return "redirect:/";
        }
    }
}
```

Файл TaskService.java

```
package task_tracker.service;

import org.springframework.stereotype.Service;
import task_tracker.domain.Project;
```

```
import task_tracker.domain.Task;
import task_tracker.dto.TaskDto;
import task_tracker.enums.Priority;
import task_tracker.enums.TaskStatus;
import task_tracker.repository.AttachmentRepository;
import task_tracker.repository.CommentRepository;
import task_tracker.repository.ProjectRepository;
import task_tracker.repository.TaskRepository;
import task_tracker.utils.Result;

import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
public class TaskService {
    private final TaskRepository taskRepository;
    private final CommentRepository commentRepository;
    private final AttachmentRepository attachmentRepository;
    private final ProjectRepository projectRepository;

    public Result<TaskDto> findById(UUID id) {
        try {
            Optional<Task> task = taskRepository.findById(id);
            if (task.isEmpty())
                return Result.error("Task was not found",
"404");
            else
                return Result.ok(task.get().mapToDto());
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error finding task", "500");
        }
    }

    public Result<List<TaskDto>> findByUserId(UUID id) {
        try {
            List<Task> tasks = taskRepository.findByUserId(id);
            return
Result.ok(tasks.stream().map(Task::mapToDto).collect(Collectors.
toList()));
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error finding tasks by
employee", "500");
        }
    }

    public Result<List<TaskDto>> findByName(String name) {
```

```
        try {
            List<Task> tasks = taskRepository.findByName(name +
'%');
            return
Result.ok(tasks.stream().map(Task::mapToDto).collect(Collectors.
toList()));
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error finding Tasks by first
name", "500");
        }
    }

    public Result<String> create(Task task) {
        try {
            task.setStatus(TaskStatus.NotStarted);
            task.setDateCreated(LocalDateTime.now());
            Task savedTask = taskRepository.saveAndFlush(task);
            if (savedTask.getProjectId() != null) {
                Project project =
projectRepository.findById(savedTask.getProjectId()).get();
                HashSet tasks = new HashSet<>();
                if (project.getTasks() != null) {
                    tasks = ((HashSet) project.getTasks());
                }
                tasks.add(savedTask.getId());
                project.setTasks(tasks);
                projectRepository.saveAndFlush(project);
            }
            return Result.ok("OK");
        } catch (Exception e) {
            return Result.error("Failed to create task", "500");
        }
    }

    public Result<String> update(UUID id, TaskDto taskDto) {
        try {
            Optional<Task> taskRead =
taskRepository.findById(id);
            if (taskRead.isEmpty())
                return Result.error("Task was not found",
"404");
            Task task = taskRead.get();
            task.setName(taskDto.getName());
            task.setDescription(taskDto.getDescription());
            task.setPriority(taskDto.getPriority());
            task.setStatus(taskDto.getStatus());
            task.setDeadline(taskDto.getDeadline());
            taskRepository.saveAndFlush(task);
            return Result.ok("Update ok");
        } catch (Exception e) {
            System.out.println(e.getMessage());
```

```
            return Result.error("Failed to update task", "500");
        }
    }

    public Result<String> delete(UUID id) {
        try {
            Task task = taskRepository.findById(id).get();
            if (task.getProjectId() != null) {
                Project project =
projectRepository.findById(task.getProjectId()).get();
                if (project.getTasks().size() == 1) {
                    project.setTasks(null);
                } else {
                    HashSet tasks = (HashSet)
project.getTasks();
                    tasks.remove(task.getId());
                    project.setTasks(tasks);
                }
                projectRepository.saveAndFlush(project);
            }
            commentRepository.deleteAllByTaskId(id);
            attachmentRepository.deleteAllByTaskId(id);
            taskRepository.deleteById(id);
            return Result.ok("Delete ok");
        } catch (Exception e) {
            return Result.error("Failed to delete task", "500");
        }
    }

    public TaskService(TaskRepository taskRepository,
CommentRepository commentRepository, AttachmentRepository
attachmentRepository, ProjectRepository projectRepository) {
        this.taskRepository = taskRepository;
        this.commentRepository = commentRepository;
        this.attachmentRepository = attachmentRepository;
        this.projectRepository = projectRepository;
    }
}
```

Файл UserService.java

```
package task_tracker.service;

import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import task_tracker.domain.ContactInfo;
import task_tracker.domain.Role;
import task_tracker.domain.User;
```

```
import task_tracker.dto.UserDto;
import task_tracker.repository.*;
import task_tracker.utils.Result;

import java.security.Principal;
import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
public class UserService {
    private final UserRepository userRepository;
    private final AttachmentRepository attachmentRepository;
    private final TaskRepository taskRepository;
    private final WorkTimeRepository workTimeRepository;
    private final ContactInfoRepository contactInfoRepository;
    private final RoleRepository roleRepository;
    private final CommentRepository commentRepository;

    public UserService(UserRepository userRepository,
AttachmentRepository attachmentRepository, TaskRepository
taskRepository, WorkTimeRepository workTimeRepository,
ContactInfoRepository contactInfoRepository, RoleRepository
roleRepository, CommentRepository commentRepository) {
        this.userRepository = userRepository;
        this.attachmentRepository = attachmentRepository;
        this.taskRepository = taskRepository;
        this.workTimeRepository = workTimeRepository;
        this.contactInfoRepository = contactInfoRepository;
        this.roleRepository = roleRepository;
        this.commentRepository = commentRepository;
    }

    public Result<UserDto> findById(UUID id) {
        try {
            Optional<User> user = userRepository.findById(id);
            if (user.isEmpty())
                return Result.error("User was not found",
"404");
            else
                return Result.ok(user.get().mapToDto());
        } catch (Exception e) {
            return Result.error("Error finding user", "500");
        }
    }

    public Result<UserDto> findByPrincipal(Principal principal)
{
        try {
            Optional<User> user =
userRepository.findByLogin(principal.getName());
```

```
            if (user.isEmpty())
                return Result.error("User was not found",
"404");
            else
                return Result.ok(user.get().mapToDto());
        } catch (Exception e) {
            return Result.error("Error finding user", "500");
        }
    }

    public Boolean userExists(String login) {
        return userRepository.findByLogin(login).isPresent();
    }

    public Result<List<UserDto>> findByName(String name) {
        try {
            List<User> users = userRepository.findByName(name +
'%');
            return
Result.ok(users.stream().map(User::mapToDto).collect(Collectors.
toList()));
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error finding Users by first
name", "500");
        }
    }

    public Result<List<UserDto>> findBySurename(String surename)
{
        try {
            List<User> users =
userRepository.findBySurename(surename + '%');
            return
Result.ok(users.stream().map(User::mapToDto).collect(Collectors.
toList()));
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error finding Users by middle
name", "500");
        }
    }

    @Transactional
    public Result<User> create(User user) {
        try {
            Optional<Role> result =
roleRepository.findByName("ROLE_USER");
            Role role = result.get();
            user.setRole(role);
            user.setPassword(new
BCryptPasswordEncoder().encode(user.getPassword()));
```

```
            User savedUser = userRepository.saveAndFlush(user);
            ContactInfo contactInfo = new ContactInfo();
            contactInfo.setUserr(savedUser);
            ContactInfo savedContactInfo =
contactInfoRepository.saveAndFlush(contactInfo);
            return Result.ok(savedUser);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Error creating User", "500");
        }
    }

    public Result<String> update(UUID id, UserDto userDto) {
        try {
            Optional<User> userRead =
userRepository.findById(id);
            if (userRead.isEmpty())
                return Result.error("User was not found",
"404");
            User user = userRead.get();
            user.setName(userDto.getName());
            if (!user.getPassword().equals(""))
                user.setPassword(new
BCryptPasswordEncoder().encode(userDto.getPassword()));
            user.setSurename(userDto.getSurename());
            user.setLogin(userDto.getLogin());
            userRepository.saveAndFlush(user);
            return Result.ok("Update ok");
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return Result.error("Failed to update user", "500");
        }
    }

    public Result<String> delete(UUID id) {
        try {
            attachmentRepository.deleteAllByUserId(id);
            commentRepository.deleteAllByUserId(id);
            taskRepository.deleteAllByUserId(id);
            workTimeRepository.deleteAllByUserId(id);
            contactInfoRepository.deleteAllByUserId(id);
            userRepository.deleteById(id);
            return Result.ok("Delete ok");
        } catch (Exception e) {
            return Result.error("Failed to delete user", "500");
        }
    }
}
```

Файл TaskRepository.java

```
package task_tracker.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import task_tracker.domain.Task;
import task_tracker.enums.Priority;
import task_tracker.enums.TaskStatus;

import java.util.List;
import java.util.UUID;

@Repository
public interface TaskRepository extends JpaRepository<Task,
UUID> {

    @Query("select t from Task t where t.user.id = :id")
    List<Task> findByUserId(UUID id);

    @Query("select t from Task t where t.status = :taskStatus")
    List<Task> findByStatus(TaskStatus taskStatus);

    @Query("select t from Task t where t.priority = :priority")
    List<Task> findByPriority(Priority priority);

    @Query("select t from Task t where ?1 is null or
lower(t.name) like ?1")
    List<Task> findByName(String name);

    @Modifying
    @Transactional
    @Query("delete from Task where user.id = ?1")
    void deleteAllByUserId(UUID id);

}
```

Файл UserRepository.java

```
package task_tracker.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import task_tracker.domain.User;

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

```
import java.util.UUID;

@Repository
public interface UserRepository extends JpaRepository<User,
UUID> {

    @Query("select u from User u where ?1 is null or
lower(u.name) like ?1")
    List<User> findByName(String name);

    @Query("select u from User u where u.login = ?1")
    Optional<User> findByLogin(String login);

    @Query("select u from User u where ?1 is null or
lower(u.surename) like ?1")
    List<User> findBySurename(String surename);
}
```

Файл User.java

```
package task_tracker.domain;

import
io.hypersistence.utils.hibernate.type.json.JsonBinaryType;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;
import org.hibernate.annotations.TypeDef;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthori
ty;
import
org.springframework.security.core.userdetails.UserDetails;
import task_tracker.dto.UserDto;

import javax.persistence.*;
import java.io.Serializable;
import java.util.Collection;
import java.util.Collections;
import java.util.Set;
import java.util.UUID;

@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class)
public class User implements Serializable, UserDetails {
    @Id
```

```
    @GeneratedValue(generator = "uuid-hibernate-generator")
    @GenericGenerator(name = "uuid-hibernate-generator",
strategy = "org.hibernate.id.UUIDGenerator")
    private UUID id;

    private String name;

    private String surename;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "role", referencedColumnName = "id",
foreignKey = @ForeignKey(name = "users_fk"))
    private Role role;

    private String login;

    private String password;

    @Type(type = "jsonb")
    @Column(name = "projects", columnDefinition = "jsonb")
    Set<UUID> projects;

    @OneToMany(mappedBy = "user")
    private Set<Task> tasks;

    @OneToMany(mappedBy = "user")
    private Set<WorkTime> workTimes;

    @OneToOne(mappedBy = "userr", fetch = FetchType.LAZY)
    private ContactInfo contactInfo;

    @OneToMany(mappedBy = "user")
    private Set<Attachment> attachments;

    @OneToMany(mappedBy = "user")
    private Set<Comment> comments;

    public UserDto mapToDto() {
        return new UserDto(
                this.id,
                this.name,
                this.surename,
                this.role,
                this.login,
                this.password,
                this.projects,
                this.contactInfo.mapToDto()
        );
    }

    @Override
```

```
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return Collections.singleton(new
SimpleGrantedAuthority(getRole().getAuthority()));
    }

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

    @Override
    public String getPassword() {
        return password;
    }

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

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

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

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

Файл User.java

```
package task_tracker.domain;

import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;
import org.springframework.format.annotation.DateTimeFormat;
import task_tracker.dto.TaskDto;
import task_tracker.enums.Priority;
import task_tracker.enums.TaskStatus;

import javax.persistence.*;
import java.io.Serializable;
```

```
import java.time.LocalDateTime;
import java.util.UUID;

@Entity
@Table(name = "tasks")
@Data
@NoArgsConstructor
public class Task implements Serializable {
    @Id
    @GeneratedValue(generator = "uuid-hibernate-generator")
    @GenericGenerator(name = "uuid-hibernate-generator",
strategy = "org.hibernate.id.UUIDGenerator")
    private UUID id;

    private String name;

    private String description;

    @ManyToOne()
    @JoinColumn(name = "assignee", referencedColumnName = "id",
foreignKey = @ForeignKey(name = "tasks_fk"))
    private User user;

    @Enumerated(EnumType.STRING)
    private TaskStatus status;

    @Column(name = "date_created")
    @DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm")
    private LocalDateTime dateCreated;

    @DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm")
    private LocalDateTime deadline;

    @Column(name = "project_id")
    private UUID projectId;
    @Enumerated(EnumType.STRING)
    private Priority priority;
    public TaskDto mapToDto() {
        return new TaskDto(
                this.id,
                this.name,
                this.description,
                this.user.mapToDto(),
                this.status,
                this.dateCreated,
                this.deadline,
                this.projectId,
                this.priority
        );
    }
}
```

Файл index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.css">
</head>
<body>
<nav class="navbar navbar-expand navbar-dark bg-dark">
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample02">
        <a class="navbar-brand" href="#">Task tracker</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
                <a class="nav-link" th:href="@{/}">Задачи</a>
            </li>
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/project/}">Проекты</a>
            </li>
        </ul>
    </div>
    <form th:action="@{/search}" class="justify-content-md-
center" th:method="post">
        <input type="text" name="searchQuery" class=""
th:object="${searchQuery}" placeholder="Search">
        <button class="btn btn-primary"
type="submit">Search</button>
    </form>
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample03">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/user/{id}(id=${userDto.id})}"
th:text="${userDto.name}"></a>
            </li>
            <li class="nav-item">
                <a class="btn btn-danger"
th:href="@{/logout}">Выход</a>
            </li>
        </ul>
    </div>
</nav>
<div th:if="${project!=null}" class="mx-4 my-4">
    <p class="h1">Проект: <span
th:text="${project.name}"></span></p>
```

```
    <p th:if="${userDto.role.name==ROLE_ADMIN ||
project.users.contains(userDto.id)}"
       class="btn btn-primary my-1"
th:href="@{/project/update/{id}(id=${project.id})}">Изменить
название</p>
    <p th:if="${userDto.projects!=null}">
        <a th:if="${userDto.projects.contains(project.id)}"
class="btn btn-primary"

th:href="@{/task/new/project/{id}(id=${project.id})}">Создать
задачу в проекте</a>
    </p>
    <form class="my-1" th:if="${userDto.role.name==ROLE_ADMIN ||
project.users.contains(userDto.id)}"
          th:action="@{/project/{id}(id=${project.id})}"
th:method="delete">
        <input class="btn btn-danger" type="submit"
value="Удалить проект"/>
    </form>
    <br>
    <br>
    <span th:if="${userDto.projects!=null}">
        <a th:if="${!userDto.projects.contains(project.id)}"
class="btn btn-primary my-3"

th:href="@{/project/subscribe/{id}(id=${project.id})}">Подписать
ся на проект</a>
    </span>
    <span th:if="${userDto.projects==null}">
        <a class="btn btn-primary my-3"

th:href="@{/project/subscribe/{id}(id=${project.id})}">Подписать
ся на проект</a>
    </span>

    <div class="my-4">
        <p class="h1">Подписчики проекта: </p>
        <p th:each="projectUser: ${projectUsers}">
            <a th:href="@{/user/{id}(id=${projectUser.id})}">
                <span th:text="${projectUser.name} "></span>
                <span th:text="${projectUser.surename}"></span>
            </a>
        </p>
    </div>

</div>

<a th:if="${project==null}" class="btn btn-primary mx-4 my-4"
th:href="@{/task/new}">Создать задачу</a>

<p class="h1 mx-4 my-4">Задачи: </p>
```

```
<div th:if="${tasks!=null}" class="row row-cols-1 row-cols-md-4
mx-4 my-4">
    <div class="col mb-4" th:each="task: ${tasks}">
        <a th:href="@{/task/{id}(id=${task.id})}" class="text-
decoration-none">
            <div class="card text-white bg-dark">
                <div class="card-body">
                    <h5 class="card-title"
th:text="${task.name}">Card title</h5>
                    <p class="card-text"
th:text="${task.description}">Content</p>
                </div>
            </div>
        </a>
    </div>
</div>
</body>
</html>
```

Файл login.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>MyBooks</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
    <link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.min.css">
</head>
<body class="text-center">
<main class="mx-auto mt-3" style="width: 350px">
    <form th:action="@{/login}" method="post" class="mt-5">
        <h1 class="h3 mb-3 fw-normal">Вход</h1>
        <div class="form-floating mt-2">
            <input type="text" id="login" name="login"
class="form-control" placeholder="Login" required=""
                   autofocus="">
            <label for="login">Логин</label>
        </div>
        <div class="form-floating mt-2">
            <input type="password" class="form-control"
                   name="password" id="password">
            <label for="password">Пароль</label>
        </div>
        <div th:if="${errorMessage}" class="alert alert-danger
mt-2"
             role="alert" th:text="${errorMessage}"></div>
        <button class="w-100 btn btn-lg btn-primary mt-2"
```

```
                type="submit">Вход
        </button>
    </form>
    <form th:action="@{/signup}" method="get">
        <button class="w-100 btn btn-lg btn-primary mt-2"
                type="submit">Регистрация
        </button>
    </form>
</main>
</body>
</html>
```

Файл task/index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.css">
</head>
<body>
<nav class="navbar navbar-expand navbar-dark bg-dark">
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample02">
        <a class="navbar-brand" href="#">Task tracker</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
                <a class="nav-link" th:href="@{/}">Задачи</a>
            </li>
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/project/}">Проекты</a>
            </li>
        </ul>
    </div>
    <form th:action="@{/search}" class="justify-content-md-
center" th:method="post">
        <input type="text" name="searchQuery" class=""
th:object="${searchQuery}" placeholder="Search">
        <button class="btn btn-primary"
type="submit">Search</button>
    </form>
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample03">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
```

```
                <a class="nav-link"
th:href="@{/user/{id}(id=${userDto.id})}"
th:text="${userDto.name}">Username</a>
            </li>
            <li class="nav-item">
                <a class="btn btn-danger"
th:href="@{/logout}">Выход</a>
            </li>
        </ul>
    </div>
</nav>
<!--------------------------------------------------------------
---------------------------------------->
<div class="mx-5 my-5">
    <p class="h1" th:text="${taskDto.name}">Task name</p>
    <p class="h5" th:text="${taskDto.description}">Task
description</p>
    <p class="">
        <span th:text="${taskDto.status}"> Task status</span>
    </p>
    <p class="">
        <span th:text="${taskDto.priority}"> Task
priority</span>
    </p>
    <p class="" th:text="${taskDto.dateCreated}">Task
dateCreated</p>
    <p class="" th:if="${taskDto.deadline!=null}"
th:text="${taskDto.deadline}">Task deadline</p>
    <p class="" th:if="${taskDto.projectId!=null}"><a
th:href="@{/project/{id}(id=${taskDto.projectId})}"

class="text-decoration-none" th:text="${project.name}"></a></p>
    <p class=""><a
th:href="@{/user/{id}(id=${taskDto.user.id})}" class="text-
decoration-none">
        <span th:text="${taskDto.user.name} "></span>
        <span th:text="${taskDto.user.surename}"></span></a></p>
    <a class="" th:if="${userDto.role.name==ROLE_ADMIN ||
userDto.id==taskDto.user.id}"
       th:href="@{/task/update/{id}(id=${taskDto.id})}">
        <button class="btn btn-primary">Изменить данные</button>
    </a>
    <br>
    <br>
    <form th:if="${userDto.id==taskDto.user.id ||
userDto.role.name==ROLE_ADMIN}"
          th:action="@{/task/{id}(id=${taskDto.id})}"
th:method="delete">
        <input type="submit" class="btn btn-danger"
value="Удалить задачу"/>
    </form>
    <br>
```

```
    <br>
    <p class="h4">Comments:</p>
    <div th:if="${comments!=null}" th:each="comment:
${comments}">
        <div class="card border-dark mb-3" style="max-width:
20rem;">
            <div class="card-header"><span
th:text="${comment.author.name} "></span> <span

th:text="${comment.author.surename}"></span></div>
            <div class="card-body">
                <p class="card-text"
th:text="${comment.getContents()}">Task comments</p>
                <p class="card-text"
th:text="${comment.getDate()}">Task comments</p>
                <form th:if="${comment.author.id==userDto.id ||
userDto.role.name==ROLE_ADMIN}"

th:action="@{/comment/{id}/{taskId}(id=${comment.id},
taskId=${taskDto.id})}" th:method="delete">
                    <input type="submit" class="btn btn-danger"
value="Удалить"/>
                </form>
            </div>
        </div>
    </div>
</div>
<!--------------------------------------------------------------
---------------------------------------->
<form th:action="@{/comment/}" th:object="${newComment}"
th:method="post" class="mx-5 my-5" style="max-width: 20rem;">
    <input type="text" th:field="*{contents}"
placeholder="Напишите комментарий" class="form-control"
id="contents"/>
    <input type="hidden" th:value="${userDto.id}" class="form-
control" id="user.id" name="user.id"/>
    <input type="hidden" th:value="${taskDto.id}" class="form-
control" id="task.id" name="task.id"/>
    <input type="submit" class="btn btn-primary"
value="Отправить"/>
</form>
<!--------------------------------------------------------------
---------------------------------------->
<div class="mx-5 my-5">
    <p class="h4">Attachments:</p>
    <div th:if="${attachments!=null}" th:each="attachment:
${attachments}">
        <a th:href="@{/attachment/{id}(id=${attachment.id})}"
class="text-decoration-none">
            <div class="card border-success mb-3" style="max-
width: 20rem;">
```

```
                <div class="card-header"><span
th:text="${attachment.user.name} "></span> <span

th:text="${attachment.user.surename}"></span></div>
                <div class="card-body">
                    <p class="card-text"
th:text="${attachment.name}">Task attachments</p>
                    <p class="card-text"
th:text="${attachment.dateUploaded}">Task attachments</p>
                    <form
th:if="${attachment.user.id==userDto.id ||
userDto.role.name==ROLE_ADMIN}"

th:action="@{/attachment/{id}/{taskId}(id=${attachment.id},
taskId=${taskDto.id})}"
                          th:method="delete">
                        <input type="submit" class="btn btn-
danger" value="Удалить"/>
                    </form>
                </div>
            </div>
        </a>
    </div>
</div>
<!--------------------------------------------------------------
---------------------------------------->
<form th:action="@{/attachment/}" th:object="${newAttachment}"
th:method="POST" class="mx-5 my-5"
      style="max-width: 20rem;" enctype="multipart/form-data">
    <input type="file" name="file"/>
    <input type="hidden" th:name="user"
th:value="${userDto.id}">
    <input type="hidden" th:name="task"
th:value="${taskDto.id}">
    <input type="submit" class="btn btn-primary"
value="Отправить"/>
</form>
<!--------------------------------------------------------------
---------------------------------------->
</body>
</html>
```

Файл user/index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
```

```
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.css">
</head>
<body>
<nav class="navbar navbar-expand navbar-dark bg-dark">
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample02">
        <a class="navbar-brand" href="#">Task tracker</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
                <a class="nav-link" th:href="@{/}">Задачи</a>
            </li>
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/project/}">Проекты</a>
            </li>
        </ul>
    </div>
    <form th:action="@{/search}" class="justify-content-md-
center" th:method="post">
        <input type="text" name="searchQuery" class=""
th:object="${searchQuery}" placeholder="Search">
        <button class="btn btn-primary"
type="submit">Search</button>
    </form>
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample03">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/user/{id}(id=${userDto.id})}"
th:text="${userDto.name}">Username</a>
            </li>
            <li class="nav-item">
                <a class="btn btn-danger"
th:href="@{/logout}">Выход</a>
            </li>
        </ul>
    </div>
</nav>
<!-----------------------------------------------------------------
---------------------------------------->
<div class="mx-5 my-5">
    <p class="h1">
        <span th:text="${userDtoFound.name}">User name</span>
        <span th:text="${userDtoFound.surename}">User
surename</span>
    </p>
    <p class="">Логин: <span
th:text="${userDtoFound.login}">User login</span></p>
```

```
    <p class=""
th:text="${userDtoFound.contactInfoDto.address}">User
address</p>
    <p class=""
th:text="${userDtoFound.contactInfoDto.phoneNumber}">User phone
mumber</p>
    <p class=""
th:text="${userDtoFound.contactInfoDto.email}">User email</p>

    <a class="" th:if="${userDto.id==userDtoFound.id}"
th:href="@{/user/update/{id}(id=${userDtoFound.id})}">
        <button class="btn btn-primary">Изменить данные</button>
    </a>
</div>
<!--------------------------------------------------------------
---------------------------------------->
<p class="h1 mx-4 my-4">Проекты</p>
<div class="row row-cols-1 row-cols-md-4 mx-4 my-4">
    <div class="col mb-4" th:each="project: ${projectList}">
        <a th:href="@{/project/{id}(id=${project.id})}"
class="text-decoration-none">
            <div class="card border-dark">
                <div class="card-body">
                    <h5 class="card-title"
th:text="${project.name}">Card title</h5>
                </div>
            </div>
        </a>
    </div>
</div>
<div class="mx-5 my-5">
    <form th:if="${userDto.id==userDtoFound.id}"
th:action="@{/user/{id}(id=${userDtoFound.id})}"
th:method="delete">
        <input type="submit" class="btn btn-danger"
value="Удалить аккаунт"/>
    </form>
    <form th:if="${userDto.id!=userDtoFound.id &&
userDto.role.name==ROLE_ADMIN}"
          th:action="@{/user/{id}(id=${userDto.id})}"
          th:method="delete">
        <input type="submit" class="btn btn-danger"
value="Удалить пользователя"/>
    </form>
</div>
</body>
</html>
```

Файл user/update.html

```
<!DOCTYPE html>
```

```
<html lang="ru" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.css">
</head>
<body>
<p class="form-control" th:text="${error}"></p>
<form th:action="@{/user/{id}(id=${user.id})}"
th:object="${user}" th:method="put">

    <p class="form-control">
        <label for="name">Имя </label>
        <input type="text" th:field="*{name}" id="name"/>
    </p>
    <p class="form-control">
        <label for="surename">Фамилия </label>
        <input type="text" th:field="*{surename}"
id="surename"/>
    </p>
    <p class="form-control">
        <label for="login">Логин </label>
        <input type="text" th:field="*{login}" id="login"/>
    </p>
    <p class="form-control">
        <label for="password">Пароль </label>
        <input type="text" th:field="*{password}"
id="password"/>
    </p>
    <p class="form-control">
        <input class="btn btn-success" type="submit"
value="Отправить"/>
    </p>
</form>
</body>
</html>
```

Файл user/update.html

```
<!DOCTYPE html>
<html lang="ru" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.css">
</head>
<body>
```

```
<nav class="navbar navbar-expand navbar-dark bg-dark">
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample02">
        <a class="navbar-brand" href="#">Task tracker</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
                <a class="nav-link" th:href="@{/}">Задачи</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Проекты</a>
            </li>
        </ul>
    </div>
    <form th:action="@{/search}" class="justify-content-md-
center" th:method="post">
        <input type="text" name="searchQuery" class=""
th:object="${searchQuery}" placeholder="Search">
        <button class="btn btn-primary"
type="submit">Search</button>
    </form>
    <div class="collapse navbar-collapse justify-content-md-
center" id="navbarsExample03">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link"
th:href="@{/user/{id}(id=${userDto.id})}"
th:text="${userDto.name}"></a>
            </li>
            <li class="nav-item">
                <a class="btn btn-danger"
th:href="@{/logout}">Выход</a>
            </li>
        </ul>
    </div>
</nav>
<form th:action="@{/task/{id}(id=${task.id})}"
th:object="${task}" th:method="put">

    <p class="form-control">
        <label for="name">Имя задачи </label>
        <input type="text" th:field="*{name}" id="name"/>
    </p>
    <p class="form-control">
        <label for="description">Описание задачи </label>
        <input type="text" th:field="*{description}"
id="description"/>
    </p>
    <p class="form-control">
        <label>Приоритет задачи </label>
        <select th:field="*{priority}" class="custom-select">
            <option th:each="priority: ${priorities}"
                    th:text="${priority}"
```

```
                    th:selected="${priority==task.priority}"
                    th:value="${priority}"></option>
        </select>
    </p>
    <p class="form-control">
        <label>Статус задачи </label>
        <select th:field="*{status}" class="custom-select">
            <option th:each="status: ${taskStatus}"
                    th:text="${status}"
                    th:selected="${status==task.status}"
                    th:value="${status}"></option>
        </select>
    </p>
    <p class="form-control">
        <label for="deadline">Конечная дата </label>
        <input type="datetime-local" th:field="*{deadline}"
id="deadline"/>
    </p>
    <p class="form-control">
        <input class="btn btn-success" type="submit"
value="Отправить"/>
    </p>
</form>
</body>
</html>
```

Приложение В

(Обязательное)

Трекер задач

Презентация

Выпускная квалификационная работа на тему «Трекер задач»

Автор работы:

студент гр. 20ВВП2 Горожанин Ярослав Александрович

Научный руководитель:

д.т.н., профессор Зинкин Сергей Александрович

Пенза 2024

Цель работы

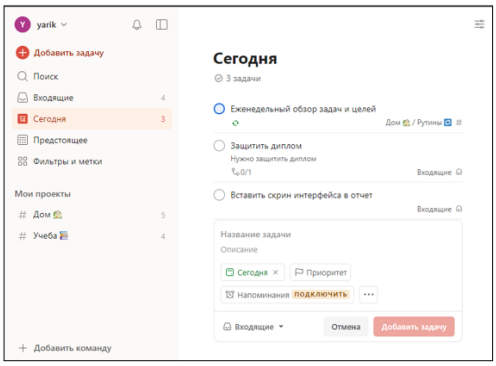
Основной целью данной работы является разработать приложение «Трекер задач» для отслеживания задач онлайн. Приложение должно позволять пользователям создавать задачи, назначать на них других пользователей, группировать эти задачи в проекты. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе.

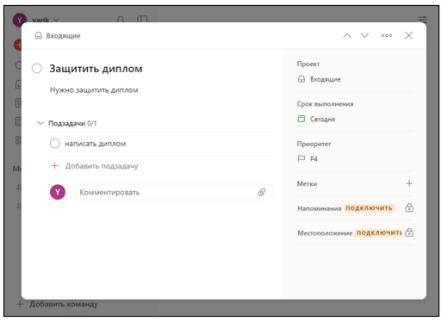
Актуальность работы

В контексте рабочей среды планирование задач играет ключевую роль в продуктивной организации труда. Приложения для отслеживания задач позволяют быстро подобрать задачу самому и приступить к её выполнению или руководству назначить задачу на работника, основываясь на нынешних приоритетах команды. Тем самым оптимизируя рабочий процесс, путем эффективного распределения времени пользователей.

Аналоги

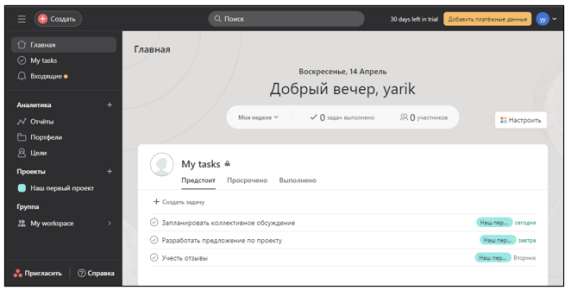
Онлайн-сервис «Todoist»

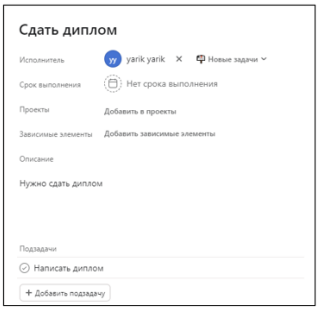




Аналоги

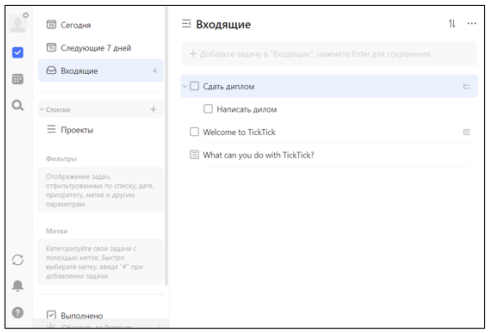
Онлайн-сервис «Asana»

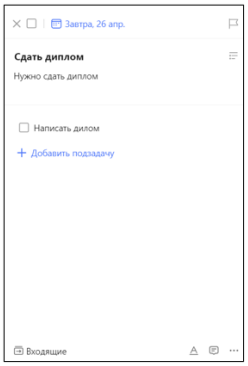




Аналоги

Онлайн-сервис «TickTick»





Постановка задачи

Приложение должно соответствовать следующим требованиям:

- предоставлять данные о пользователе, задачах и проектах;
- проверять корректность вводимой информации в базу данных;
- редактировать, добавлять и удалять данные о пользователе, задачах и проектах;
- обеспечивать безопасность данных, позволяя взаимодействовать с задачами и проектами, только авторизованным пользователям.

Приложение должно предоставлять пользователям следующий функционал:

- регистрация и авторизация;
- создание задачи и проекта;
- редактирование, добавление и удаление данных о пользователе, задаче и проекте;

Используемые технологии

Архитектура приложения

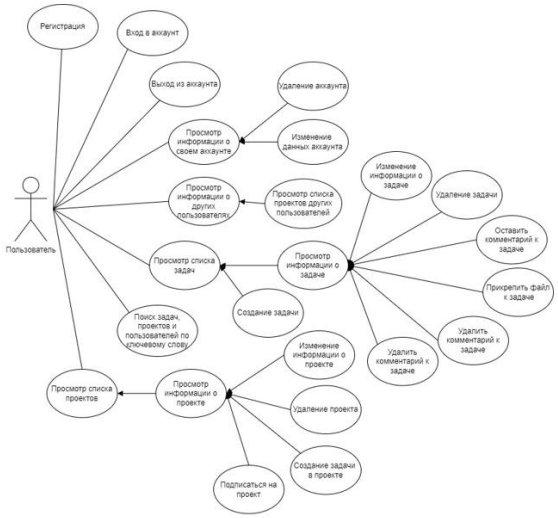
В качестве шаблона проектирования был выбран Model-View-Controller (MVC) — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние.

Представление (View) отвечает за отображение данных предметной области (модели) пользователю, реагируя на изменения модели.

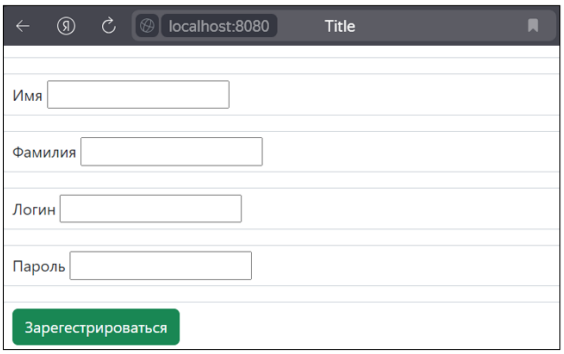
Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Архитектура приложения



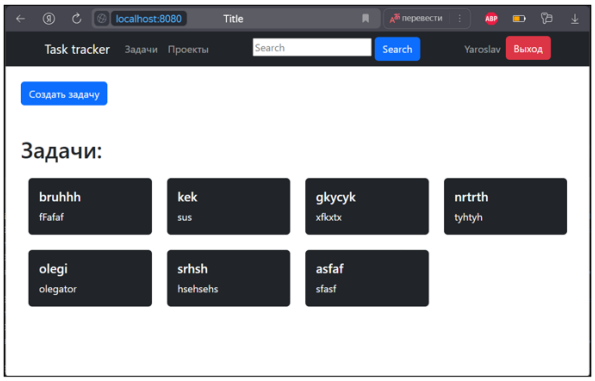
Приложение «Трекер задач»

Страницы авторизации и регистрации



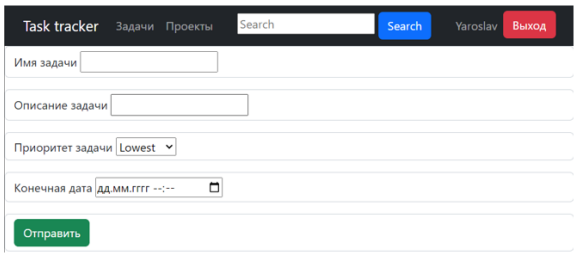
Приложение «Трекер задач»

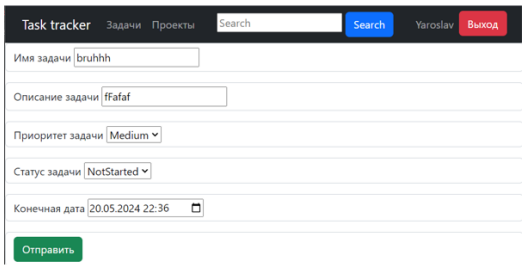
Главная страница приложения



Приложение «Трекер задач»

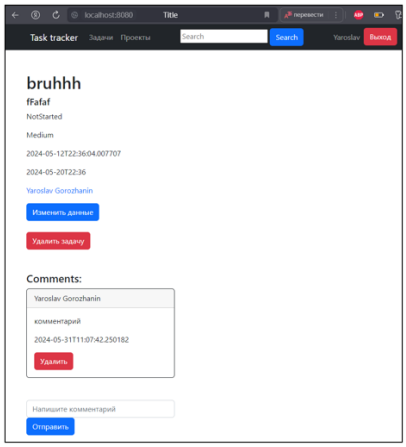
Страницы создания и изменения задачи





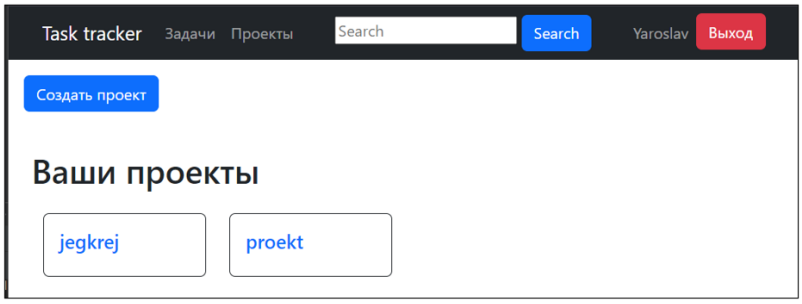
Приложение «Трекер задач»

Страница задачи



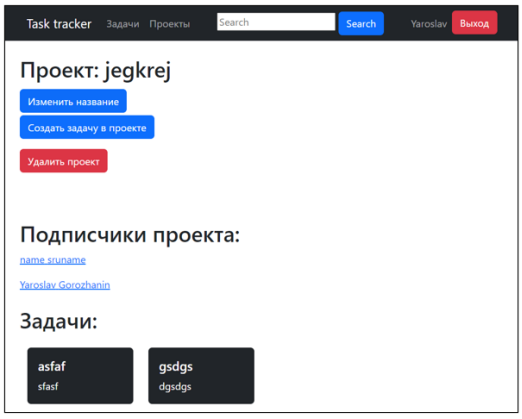
Приложение «Трекер задач»

Страница со списком проектов пользователя



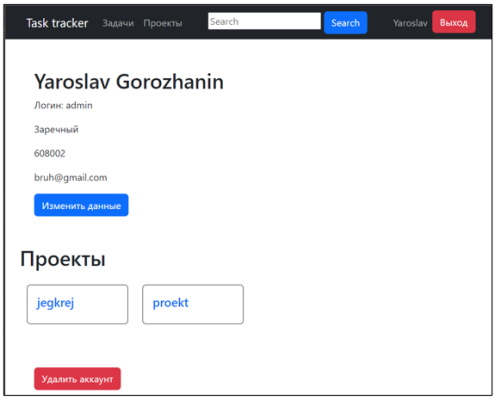
Приложение «Трекер задач»

Страница проекта



Приложение «Трекер задач»

Страница с информацией о пользователе



Результаты работы

Результатом работы является разработанное приложение «Трекер задач», которое называется TaskTracker, а также отчёт, в котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор его основных функциональных особенностей.

Разработанное приложение позволяет пользователю создавать, изменять и удалять задачи, объединять их в проекты, а также оставлять и удалять к ним комментарии и загружать, скачивать и удалять файлы, прикрепленные к задачам. После проведения тестирования было выяснено, что разработанное приложение соответствует всем поставленным в начале работы требованиям и работает корректно.

Спасибо за внимание