СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 8](#_Toc483551226)

[**1** ОБЗОР ЛИТЕРАТУРЫ 10](#_Toc483551227)

[**1.1** Обзор используемых технологий 10](#_Toc483551228)

[**1.1.1** Выбор технологий клиентской части приложения 10](#_Toc483551229)

[**1.1.2** Выбор технологий для разработки веб-сервиса 11](#_Toc483551230)

[**1.1.3** Выбор СУБД 12](#_Toc483551231)

[**1.2** Обзор аналогов 14](#_Toc483551242)

[**1.2.1** Trello 14](#_Toc483551243)

[**1.2.2** Kanban flow 15](#_Toc483551244)

[**1.3** Выводы 17](#_Toc483551245)

[**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 18](#_Toc483551246)

[**2.1** Структура приложения 19](#_Toc483551247)

[**2.2** Структура базы данных 22](#_Toc483551248)

[**2.3** Инфраструктура приложения 25](#_Toc483551249)

[**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 26](#_Toc483551250)

[**3.1** Модуль аутентификации 26](#_Toc483551251)

[**3.2** Модуль отправки уведомлений 29](#_Toc483551252)

[**3.3** Модуль построения отчетов 30](#_Toc483551253)

[**3.4** Модуль доступа к данным 32](#_Toc483551254)

[**3.5** Модуль API веб-сервиса 36](#_Toc483551255)

[**3.6** Модуль маршрутизации фреймворка Spring 38](#_Toc483551256)

[**3.7** Модуль управления задачами 38](#_Toc483551257)

[**3.8** Модуль администрирования и авторизации 39](#_Toc483551258)

[**3.9** Модуль взаимодействия с веб-сервисом 40](#_Toc483551259)

[**3.10** Модуль рендеринга html-страниц 41](#_Toc483551260)

[**3.11** Фреймворк Angular 42](#_Toc483551261)

[**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 43](#_Toc483551262)

[**4.1** Процесс создания задачи 43](#_Toc483551263)

[**4.2** Поиск задачи 44](#_Toc483551264)

[**4.3** Организация доступа к базе данных 45](#_Toc483551265)

[**4.4** Построение отчетов о затраченном времени 46](#_Toc483551266)

[**4.5** Изменение статуса задачи 48](#_Toc483551267)

[**4.6** Добавление пользователя в проектную группу 49](#_Toc483551268)

[**4.7** Создание проектной группы 51](#_Toc483551269)

[**4.8** Добавление прав к проектной группе 53](#_Toc483551270)

[**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 54](#_Toc483551271)

[**5.1** Функциональное тестирование 54](#_Toc483551272)

[**5.2** Модульное тестирование 57](#_Toc483551273)

[**5.3** Вывод 60](#_Toc483551274)

[**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ](#_Toc483551275)

[ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ](#_Toc483551276)

[ПРОГРАММНОГО СРЕДСТВА ОТСЛЕЖИВАНИЯ ЗАДАЧ 61](#_Toc483551277)

[**7.1** Характеристика программного средства 61](#_Toc483551278)

[**7.2** Расчет сметы затрат и отпускной цены на разработку программного средства 61](#_Toc483551279)

[**7.3** Расчет экономического эффекта от продажи программного продукта 65](#_Toc483551280)

[**7.4** Расчет показателей эффективности разработки программного продукта 67](#_Toc483551281)

[ЗАКЛЮЧЕНИЕ 70](#_Toc483551282)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 71](#_Toc483551283)

[ПРИЛОЖЕНИЕ А 72](#_Toc483551284)

[ПРИЛОЖЕНИЕ Б 73](#_Toc483551285)

[ПРИЛОЖЕНИЕ В 74](#_Toc483551286)

# ВВЕДЕНИЕ

Целью дипломного проекта является разработка программного средства отслеживания задач с помощью метода управления разработкой kanban. Метод управления разработкой kanban представляет собой инструмент в виде доски с задачами, который позволяет наладить процесс работы над одним конкретным проектом. В самом простом случае kanban доска представляет собой страницу с четырьмя колонками. В каждой колонке находятся задачи с определенной степенью готовности: новая задача, задача на стадии разработки, задача в стадии тестирования и выполненная задача. В случае, если задача была выполнена некачественно, она возвращается к одному из предыдущих статусов.

В последнее время, Kanban набирает большую популярность в производстве программного обеспечения. Многие компании-разработчики программного обеспечения считают данный подход исключительно полезным. И хотя Kanban не достаточно эффективен для продуктовых команд, он очень эффективен для команд следующего вида:

– группы сопровождения программного обеспечения;

– группы тестирования, работающие отдельно от групп разработки;

– службы поддержки.

Также стоит отметить, что методология Kanban эффективна для различного рода стартапов, когда у команды нет четкого плана, но ведется активная работа над программным продуктом.

Однако настоящий программный продукт не является направленным исключительно на применение в IT-сфере. Данный проект может быть полезен в том числе для организации труда в сферах торговли, строительства или же просто может быть использован людьми в качестве планировщика.

В силу популярности методологий разработки с использованием kanban у настоящего программного продукта существует ряд аналогов. Наиболее популярными среди них являются Trello и Kanban flow.

На рисунке 1 представлен внешний вид рабочего стола проекта в Trello. Отсюда можно увидеть следующие особенности:

– статусы, которые может иметь задача, создаются пользователем;

– для задач можно указывать теги;

– для задач можно устанавливать крайний срок выполнения.

На рисунке 2 представлен внешний вид рабочего стола проекта в Kanban Flow. Данная доска имеет следующие особенности:

– пользователю предлагается четыре статуса задач;

– каждую задачу можно разбить на несколько подзадач;

– имеется возможность добавления собственных статусов задач;

– для каждой задачи назначается свой исполнитель.

Все представленные аналоги являются довольно гибкими в плане настройки и довольно простыми в использовании, в том числе и для тех пользователей, которые никогда не работали с kanban досками.



Рисунок 1 – ­ Внешний вид доски Trello



Рисунок 2 ­– Внешний вид доски Kanban flow

Задачей данного дипломного проекта является разработка программного средства отслеживания задач. Данный программный продукт будет представлен в виде веб-приложения, состоящего из сайта и веб-сервиса. Планируется реализовать следующий функционал:

* организация ведения проектов с использованием доски Kanban;
* построение отчетов о затраченном времени;
* поиск задач;
* создание проектов, ролей, управление списком пользователей, правами.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор используемых технологий

В данном разделе выберем технологии, которые будут использованы для реализации настоящего проекта.

### **1.1.1** Выбор технологий клиентской части приложения

Веб–сайт может быть представлен как одностраничное веб-приложение (в этом случае рендеринг html-страницы производится на стороне клиента) или как многостраничное приложение (рендеринг html-страницы производится на стороне веб-сервиса).

На сегодняшний день в области разработки веб-приложений широко распространена разработка одностраничных приложений (SPA – Single Page Application) [1]. Веб-сайт представлен единственной веб-страницей-оболочкой, организующей взаимодействие с пользователем посредством динамической подгрузки. При последующих запросах сервер и клиент осуществляют обмен данными (например, в формате JSON), а непосредственно рендеринг страницы производится на стороне клиента. У SPA веб-сайтов есть два неоспоримых преимущества: экономия сетевого трафика и уменьшение нагрузки на серверную часть приложения. Рассмотрим технологии разработки SPA веб-приложений.

В настоящее время наиболее пополярными фреймворками разработки SPA веб-приложени являются Angular и ReactJS.

Angular является мощным структурным фреймворком для динамических веб-приложений [2]. В фреймворке используется шаблон проектирования MVW (Model-View-Whatever) (Модель-Отображение-Что угодно) и расширение традиционного HTML синтаксиса директивами. Другая отличительная черта фреймворка — автоматическая синхронизации модели и представления. Кроме того, Angular поддерживает внедрение зависимостей и предлагает множество других полезных функций, среди которых можно выделить маршрутизацию, асинхронные запросы, валидацию форм, шаблоны и анимацию. Angular имеет достаточно низкий порог вхождения благодаря обширной документации и многочисленным учебным материалам, как из официальных источников, так и созданным крупным сообществом разработчиков. Разработчик Angular – компания Google, предоставляет официальное руководство по стилю написания кода на данном фреймворке. Разработка на данном фреймфорке может осуществляться на двух языках программирования: JavaScript и TypeScript.

ReactJS — это JavaScript фреймворк разработанный компанией Facebook, который используется для построения пользовательского интерфейса [3]. Одной из его отличительных особенностей является возможность использовать JSX язык программирования с близким к HTML синтаксисом, который транслируется в JavaScript. Разработчики могут добиваться высокой производительности приложений с помощью  объектной модели документа Virtual DOM. C ReactJS возможно создавать изоморфные приложения, в которых данные могут подгружаться с сервера в асинхронном режиме без полной перезагрузки страницы. Созданные компоненты могут быть с легкостью изменены и повторно использованы в новых проектах. Высокий процент переиспользования кода повышает покрываемость тестами, что, в свою очередь, приводит к более высокому уровню контроля качества.

Из-за низкого порога вхождения и наличия обширной документации выбор был сделан в пользу фреймворка Angular. В связи с ограниченным сроком, на реализацию настоящего проекта, эти факторы приобрели ключевое значение.

### **1.1.2** Выбор технологий для разработки веб-сервиса

Поскольку для языка Java существует множество технологий, упрощающих разработку серверной части веб-приложений [4], рассмотрим некоторые из них. Наиболее распространенными фреймворками для разработки веб-приложений являются: Spring Framework, JSF, Grails.

Spring Framework — это универсальный Java фреймворк с открытым исходным кодом [5]. Spring состоит из множества модулей, которые разработчик может подключать к своему приложению по необходимости. Подавляющее большинство этих модулей может работать независимо друг от друга. По умолчанию Spring предоставляет две основных возможности: инверсия управления (каждый объект получает свои зависимости, а не создает их самостоятельно) и внедрение зависимостей. Spring имеет низкий порог вхождения благодаря обширной документации и большому количеству различных статей в сети интернет.

Фреймворк JSF (Java Server Faces) был разработан компанией Oracle в рамках спецификации J2EE [6]. Данный фреймворк не подходит для быстрой разработки веб-приложений, однако он легок в использовании благодаря наличию качественной документации и отсутствию внешних зависимостей, при использовании данного фреймворка не происходит выход за рамки спецификации J2EE. JSF славится своей богатой инфраструктурой и различными модулями, которые подходят для решения широкого круга задач: разработки пользовательского интерфейса, взаимодействия с базами данных и др. Главным недостатком данного фреймворка является сложность его внутреннего устройства.

Фреймворк Grails, в отличие от предыдущих, написан не на языке Java, а на языке Groovy, однако он является полностью совместимым с языком программирования Java [7]. Вообще, по своей сути, Grails можно считать не фреймворком, а своеобразной платформой для интеграрации иных фреймворков. Это позволяет Grails взаимодействовать со многими фреймворками, такими как Spring, Hibernate, Struts 2 и др. Вместе с вышеупомянутыми фреймворками Grails содержит в себе различные плагины для интеграции с контейнерами сервлетов, базами данных и средами разработки.

Среди рассмотренных фреймворков в качестве инструмента для разработки настоящего проекта будет использован Spring Framework, поскольку на сегодняшний день этот фреймворк является самым популярным среди Java разработчиков. Также Spring содержит набор стандартных конфигураций Spring Boot, воспользовавшись которыми отпадает необходимость настройки проекта вручную.

### **1.1.3** Выбор СУБД

Выберем стредство управления базой данных. По модели данных базы данных бывают: иерархические, объектные, объектно-реляционными, реляционными, сетевыми, функциональными. В настоящее время наиболее популярными моделями баз данных являются иерархические и реляционные.

Однако, в настоящей предметной области данные хорошо структурированы (задача, проект, пользователь и др), что хорошо проецируется на реляционные базы данных. В связи с этим фактом, а также в связи с имеющимся опытом работы с реляционными СУБД, выбор пал в их сторону.

SQLite по своей сути является файловой базой данных: все таблицы хранятся в виде файла на локальном компьютере [8]. Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не интерфейса, что повышает скорость и производительность операций.

Типы данных, которые возможно хранить в базе данных под управлением SQLite: NULL, INTEGER (числовое значение в целочисленном формате), REAL (числовое значение с плавающей точкой, хранится в формате 8-байтного числа IEEE с плавающей точкой), TEXT (значение строки текста, хранится с использованием кодировки базы данных (UTF-8, UTF-16BE или UTF-16LE)), BLOB (значение бинарных данных, хранящихся точно в том же виде, в каком были введены).

Преимущества SQLite:

– файловая структура;

– использование стандартов SQL;

– масштабируемость.

Недостатки SQLite:

– отсутствие системы пользователей;

### – отсутствие возможности увеличения производительности;

### – вероятность повреждения данных при асинхронном доступе.

### MySQL в настоящее время является одной из самых популярных СУБД, используемых при разработке высоконагруженных приложений [9]. Для нее существует огромное количество документации и различных статей, помогающих начать работу с данной СУБД.

### MySQL поддерживает большое количество типов данных: TINYINT (очень малые целочисленные значения), SMALLINT (малые целочисленные значения), MEDIUMINT (средние целочисленные значения), INTEGER, BIGINT (большие целочисленные значения), DOUBLE, DECIMAL (распакованное значение с плавающей точкой, всегда знаковое), DATE (дата), DATETIME (дата и время в одном значении), TIMESTAMP (временная отметка timestamp), TIME, YEAR (год, 2 или 4 числа), CHAR, VARCHAR (строковое значение переменной длины), FLOAT, SET (множество), ENUM (перечисление), BLOB (значение типа BLOB 65535 (216 - 1) символов - максимальная длина), TINYBLOB, MEDIUMBLOB, LONGBLOB.

### Преимущества MySQL:

– простота использования;

– поддержка большинства функций языка SQL;

– реализовано большое количество средств, для безопасности;

– высокая производительность.

### Недостатки MySQL:

### – не все возможности SQL реализованы;

### – невысокая надежность, связанная со способами обработки данных;

### – медленная разработка: новые версии СУБД выходят с задержкой.

От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного [10]. Например, полной поддержкой транзакций. Акцент разработчиков был уделен производительности СУБД. Параллельность достигается не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом. PostgreSQL легко дополняется хранимыми процедурами.

Типы данных поддерживаемые PostgreSQL: bigint (знаковое 8-ми битное целочисленное значение), bigserial (автоматически инкрементируемое 8-ми битное целочисленное значение), bit (строка постоянной длины), bit varying (строка переменной длины), boolean (логическое значение, принимает значения true или false), box (прямоугольник на плоскости), bytea (массив байт), character varying (строковое значение переменной длины), character (строковое значение постоянной длины), cidr (IPv4/IPv6 сетевой адрес), circle (круг на плоскости), date (календарная дата), double, inet (IPv4/IPv6 адрес хоста), integer, interval (отрезок времени), macaddr (MAC адрес), mony (валютное значение), numeric (точное численное значение с выбранной точностью), path (геометрическая кривая на плоскости), point (геометрическая точка на плоскости), polygony (прямоугольик на плоскости), real (число с плавающей точкой одинарной точности), smallint (знаковое целочисленное значение), serial (автоматические инкрементируемое целочисленное значение), text (строковое значение переменной длины), time (время суток (без часового пояса)), time with time zone (время суток (включая часовой пояс)), timestamp (дата и время (без часового пояса)), timestamp with time zone (дата и время (с часовым поясом)), tsquery (текстовый поисковый запрос), tsvector (документ текстового поиска), txid\_snapshot (пользовательский снимок транзакции с ID), uuid (универсальный уникальный идентификатор).

Преимущества PostgreSQL:

– бесплатная СУБД с качественной реализацией SQL;

– наличие большого количества дополнений;

– реализация объектно-ориентированной модели;

### Недостатки PostgreSQL:

– слабая производительность (особенно при чтении);

– низкая популярность;

– мало хостингов поддерживает данную СУБД;

В настоящем проекте будет использоваться база данных MySQL. Именно эта СУБД будет использована, потому что она является одной из самых популярных среди всех ныне существующих реляционных баз данных, а так же в связи с имеющимся опытом разработки на данной СУБД. Также выбор в сторону MySQL пал, потому что данная база является бесплатной, кроссплатформенной и хорошо подходит для хранения структурированных данных.

**1.2** Обзор аналогов

Как было упомянуто ранее, поскольку методологии с использованием Kanban досок широко распространены в настоящее время, уже существует несколько популярных продуктов в этой области. Рассмотрим некоторые из них более детально.

**1.2.1** Trello

Trello представляет собой доску с задачами, которые распределены по колонкам (см. рис. 1). В каждой колонке содержится задача принадлежащая той или иной группе.

Каждая задача представлена в виде карточки. С карточками можно проводить большое количество манипуляций (см. рис. 1.1): комментирование, прикрепление файлов, назначение меток и др.

В Trello для каждого проекта создается своя рабочая группа (см. рис. 1.2). Члены данной группы имеют возможность просматривать в режиме реального времени состояние задач, касающихся проекта. Также возможно получать уведомления по электронной почте.

В общем случае trello представляет собой web-приложение, однако имеется и приложение для мобильных устройств под управлением операционной системы IOS.

По мнению некоторых разработчиков Trello не годится для разработки крупномасштабных проектов, а больше подходит для различного рода стартапов или в качестве ежедневника. Использование всех версий Trello в настоящий момент полностью бесплатно.



Рисунок 1.1 – Экран управления задачей



Рисунок 1.2 – Экран рабочей группы проекта

В общем случае trello представляет собой web-приложение, однако имеется и приложение для мобильных устройств под управлением операционной системы IOS.

По мнению некоторых разработчиков Trello не годится для разработки крупномасштабных проектов, а больше подходит для различого рода стартапов или в качестве ежедневника. Использование всех версий Trello в настоящий момент полностью бесплатно.

**1.2.2** Kanban flow

Данный сервис является абсолютно бесплатным и позволяет создать несколько досок, при необходимости работы сразу над несколькими проектами.

Изначально пользователю предоставляется доска с четырьмя колонками (см. рис. 2), в каждой колонке задачи с определенным статусом:

– новые задачи;

– задачи на сегодня;

– задачи, которые выполняются;

– выполненные задачи.

При добавлении новой задачи в проект можно указывать: время, которое потребуется для решения задачи, комментарий к задаче и исполнителя задачи (см. рис. 1.3). При работе над задачей можно указать фактическое время ее выполнения.



Рисунок 1.3 – Создание новой задачи в kanban flow

Смена статуса задачи осуществляется методом перетаскивания, а при нажатии на задачу правой кнопкой мыши, появляется контекстное меню со списком некоторых действий над задачей.

Данная доска является полностью настраиваемой, при необходимости пользователь может изменить порядок колонок, изменить название колонок, добавить или удалить колонки (см. рис. 1.4).



Рисунок 1.4 –­ Экран редактирования колонок Kanban flow

Также существует возможность добавления участников в проект и коллективной работы над ним. Приглашенному пользователю придет уведомление на электронную почту.

В итоге можно сказать, что доска Kanban flow довольно функциональна и удобна, одним из недостатков, для некоторых пользователей, является отсутствие мобильного приложения Kanban flow.

**1.3** Выводы

Резюмируя всё вышесказанное можно сказать, что настоящий дипломный проект будет состоять из серверной и клиентской частей. Серверная часть будет написана на языке программирования Java, с использованием фреймворка Spring. Клиентская часть будет написана с использованием фреймворка Angular на языке JavaScript. В качестве СУБД была выбрана MySQL.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Перед разработкой структуры приложений необходимо более глубоко изучить требования к разрабатываемой системе.

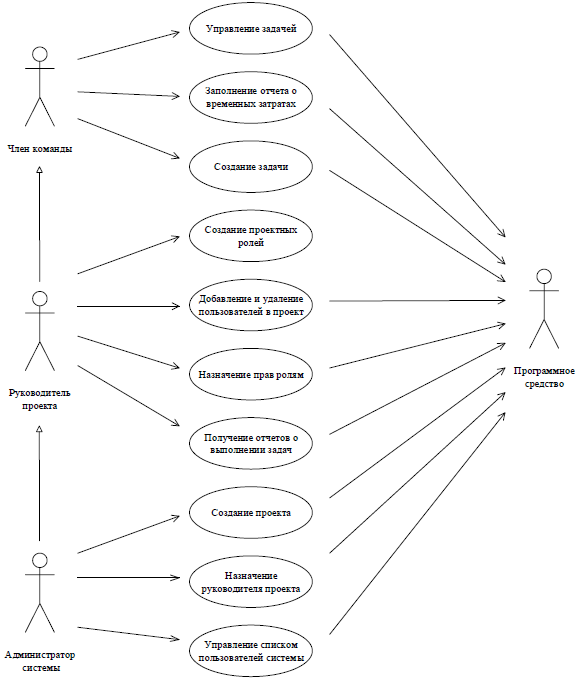
В приложении В изображена диаграмма использования разрабатываемого программного продукта. В системе можно выделить четыре актера: пользователь, руководитель проекта, администратор системы и непосредственно программное средство (см. рис. 2.1).

Рисунок 2.1 – Диаграмма использования программного средства

Все процессы, происходящие в системе сводятся к запросу к программному средству. Пользователь может управлять задачей и заполнять отчеты о затраченном времени. Руководитель кроме вышеупомянутых действий может так же: создавать задачи, назначать проектную группу, назначать права для группы, отслеживать статистику по задачам. Администратор системы обладает всеми правами.

Исходя из функций пользователей системы, была проведена декомпозиция программного средства на блоки, что позволяет реализовать концепцию слабой связанности элементов приложения.

## 2.1 Структура приложения

После изучения теоретических аспектов разрабатываемой системы и постановки требований к ней, систему необходимо разбить на функциональные блоки. Таким образом достигается гибкость архитектуры, что позволяет изменять сами модули без изменения всей системы в целом.

В настоящей системе на стороне веб-сервиса можно выделить следующие функциональные блоки (см. рис 2.2):

– модуль API веб-сервиса;

– модуль аутентификации;

– модуль отправки уведомлений;

– модуль построения отчетов;

– модуль маршрутизации фреймворка Spring;

– модуль управления задачами;

– модуль администрирования и авторизации;

– модуль доступа к данным;

– база данных веб-сервиса.

На стороне веб-сайта можно выделить следующие функциональные блоки (см. рис. 2.3):

– модуль рендеринга html-страниц;

– модуль взаимодействия с веб-сервисом;

– фреймворк Angular;

Структурная схема, где представлены все вышеперечисленные блоки приведена на чертеже ГУИР.400201.136 С1. В системе каждый модуль выполняет свою задачу и взаимодействует с другими модулями посредством интерфейсов. Рассмотрим функциональные модули разрабатываемой системы.

*Модуль API веб-сервиса.* Этот блок является своего рода «ядром» настоящего программного продукта. Данный модуль принимает запросы от модуля маршрутизации фреймворка Spring, затем обращается, по необходимости, к другим модулям веб-сервиса, производит предварительную обработку ответа и посылает ответ модулю маршрутизации. В свою очередь модуль маршрутизации посылает полученный и обработанный ответ на сторону веб-сайта.

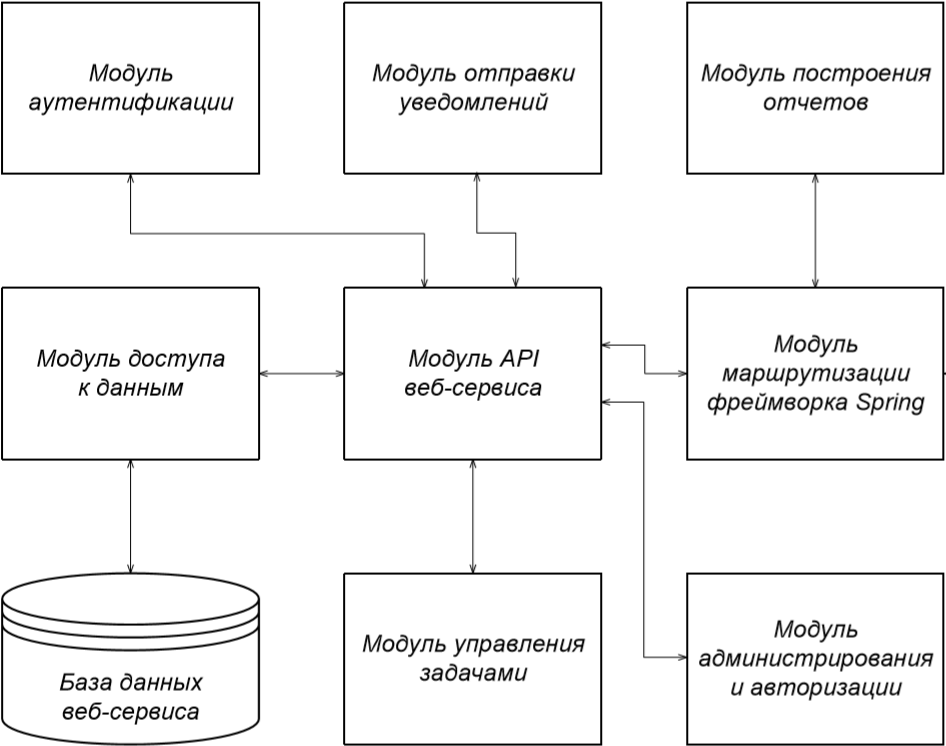


Рисунок 2.2 – Структурная схема веб-сервиса

*Модуль аутентификации.* Данный модуль отвечает за аутентификацию пользователей. Аутентификация производится с помощью логина и пароля, причем пароль хранится на стороне веб-сервиса в хешированном виде, используется метод хеширования bcrypt. Данный метод хеширования является «односторонним», это значит что не существует способа по ключу получить исходный пароль. Во время авторизации из базы данных извлекается хеш пароля, который впоследствии сравнивается с хешем пароля, который ввел пользователь. При успешной авторизации пользователь перенаправляется на главную страницу приложения, иначе происходит перенаправление на страницу авторизации. Блок аутентификации реализуется с помощью модуля Spring Security фреймворка Spring.

*Модуль отправки уведомлений.* Данный модуль предназначен для отправки сообщений по электронной почте пользователям. Модуль может использоваться для отправки уведомлений при смене пароля, восстановлении пароля, изменениях в проекте и др. Данный модуль планируется разработать с помощью библиотеки JavaMail и ее обертки spring-boot-starter-mail.

*Модуль построения отчетов.* Модуль построения отчетов необходим для получения статистики по задачам, имеющимся на проекте и для отправки отчетов пользователем о проделанной работе.

*Модуль маршрутизации фреймворка Spring.* Данный модуль является своего рода «точкой входа» в веб-сервис для веб-сайта. Модуль представляет собой контроллер, который принимает запросы по протоколу http в формате json и делегирует данный запрос модулю API веб-сервиса. Получив ответ от модуля API веб-сервиса, модуль маршрутизации отправляет клиенту ответ.

*Модуль управления задачами.* Данный модуль предназначен для создания задач, смены статуса задачи поиска задач и т.д. Модуль будет представлен в виде набора классов, позволяющих изменять статус задачи, назначать задачу пользователю и др.

*Модуль администрирования и авторизации*. В данном модуле сосредоточены функции получения прав пользователя, назначения прав пользователям а так же для администрирования системы в целом (создание проекта, назначение руководителя проекта). Авторизация производится с использованием модуля Spring Security фреймворка Spring. После создания проекта его руководитель создает проектную группу. Для каждой группы указываются её права: возможность создавать задачу, право на смену статуса задач, может ли пользователь выбирать задачу самостоятельно, может ли пользователь назначать задачи другим пользователям.

*Модуль доступа к данным.* Данный модуль является своего рода адаптером между базой данных и непосредственно веб-сервисом. Модуль будут реализовывать шаблон проектирования под названием «объект для доступа к данным». Используя такой подход можно с лёгкостью поменять схему базы данных или тип хранилища данных (например с SQL на NoSQL), при этом не затронув логику самого веб-сервиса. Данный модуль реализуется с помощью модуля Spring Jdbc фреймворка Spring.

*База данных веб-сервиса.* В настоящем программном продукте хранилище данных представлено в виде базы данных SQL. В качестве базы данных была выбрана база MySQL с движком MariaDB.

*Модуль рендеринга html-страниц.* Поскольку клиентская часть приложения основана на концепции SPA, рендеринг html-страниц происходит непосредственно на стороне клиента (в браузере). В таком случае клиент получает ответ от сервера в формате json и на его основе отрисовывает html-страницу. Такой подход позволяет сократить объем трафика между веб-сайтом и веб-сервисом.

*Модуль взаимодействия с веб-сервисом.* Данный модуль работает на стороне веб-сайта и производит передачу запроса к веб-сервису по протоколу http. После обработки запроса веб-сервисом данный модуль получает ответ по тому же протоколу. Запросы отправлятся на веб-сервер в асинхронном режиме с использованием ajax.

*Фреймворк Angular.* Данный модуль по сути является фреймворком, упрощающим разработку клиентской части веб-приложения. С помощью этого фреймворка осуществляется рендеринг html-страниц и отправка http-запросов веб-сервису.

В данной подглаве были рассмотрены функциональные блоки разрабатываемой системы, установлены их предназначения и связи между ними.



Рисунок 2.3 – Структурная схема веб-сайта

## 2.2 Структура базы данных

Следующим этапом разработки программного продукта является построение базы данных (см. рис. 2.4). В приложении Г представлена структура базы данных программного средства отслеживания задач.

*User.* Данная таблица предназначена для хранения информации о пользователях системы. Первичным ключом здесь является поле *us\_username* (логин пользователя). Внешний ключ *us\_company\_id* ссылается ссылается на таблицу company и указывает в какой компании работает пользователь. Таблица *user* состоит в отношении многие ко многим с таблицей *role* и с таблицей *project*.

*Project.* В этой таблице хранится информация о проектах внутри системы. В качестве первичного ключа здесь выступает поле *pr\_id* (синтетически сгенерированное число). Внешний ключ *pr\_lead* ссылается на таблицу *user* и указывает руководителя проекта. Внешний ключ *pr\_company\_id* ссылается на таблицу *company* и указывает в рамках какой компании будет разрабатываться проект.

Таблица *task* используется для храненения проектных задач. В качестве первичного ключа используется поле *ta\_id* (искусственное число). Внешний ключ *ta\_status* ссылается на таблицу *task\_status*, тем самым указывая текущий статус задачи. Внешний ключ *ta\_creator* ссылается на таблицу user и указывает создателя задачи. Внешний ключ ta\_executor указывает текущего исполнителя задачи. Внешний ключ *ta\_project* ссылается на таблицу *project* и указывает проект, которому принадлежит задача. Таблица *task* состоит в отношении многие ко многим с таблицей *tag*.

Таблица *tag* представляет собой набор тегов для задач в рамках проекта. По тегам осуществляется поиск проектных задач. В качестве первичного ключа выступает поле *ta\_id*. Таблица состоит в отношении многие ко многим с таблицей *task*.

Таблица *task\_status* используется для хранения статуса задачи в рамках проекта. Поле *ts\_id* используется в качестве первичного ключа. Внешний ключ *ts\_project* ссылается на таблицу *project* и указывет проект в рамках которого задача может иметь данный статус. При удалении проекта соответствующие записи из *task\_status* также будут удалены.

*Comment.* Данная таблица используется для хранения комментариев к задаче. Поле *co\_id* здесь является первичным ключом. Внешний ключ *co\_task\_id* ссылается на таблицу *task*, тем самым указывая задачу, которой принадлежит комментарий. Внешний ключ *co\_creator* ссылается на таблицу *user*, тем самым указывая автора комментария.

Таблица *attachment* предназначена для хранения прикрепленных к задаче файлов. Поле *at\_id* является первичным ключом. Внешний ключ *at\_task\_id* ссылается на таблицу *task*, тем самым указывая задачу к которой прикреплен файл. Внешний ключ *at\_owner* ссылается на таблицу user и указывает пользователя, прикрепившего файл. При удалении пользователя, все его файлы так же будут удалены.

Таблица *company* используется для хранения компаний внутри системы. В качестве первичного ключа используется поле *co\_id*. Внешний ключ *co\_owner* ссылается на таблицу *user*, тем самым указывая владельца компании.

Таблица *role* используется для хранения ролей пользователей в рамках проекта. Поле *ro\_id* является первичным ключом. Внешний ключ *ro\_project\_id* ссылается на проект, в рамках которого используется та или иная роль. Таблица *role* состоит в отношении многие ко многим с таблицей *permission* и таблицей *user*.

Таблица *permission* используется для хранения прав пользователя в рамках роли. Поле *pe\_id* является первичным ключом. Внешний ключ *pe\_project\_id* ссылается на проект, в рамках которого используется данный вид права. Таблица permission состоит в отношении многие ко многим с таблицей *role*.

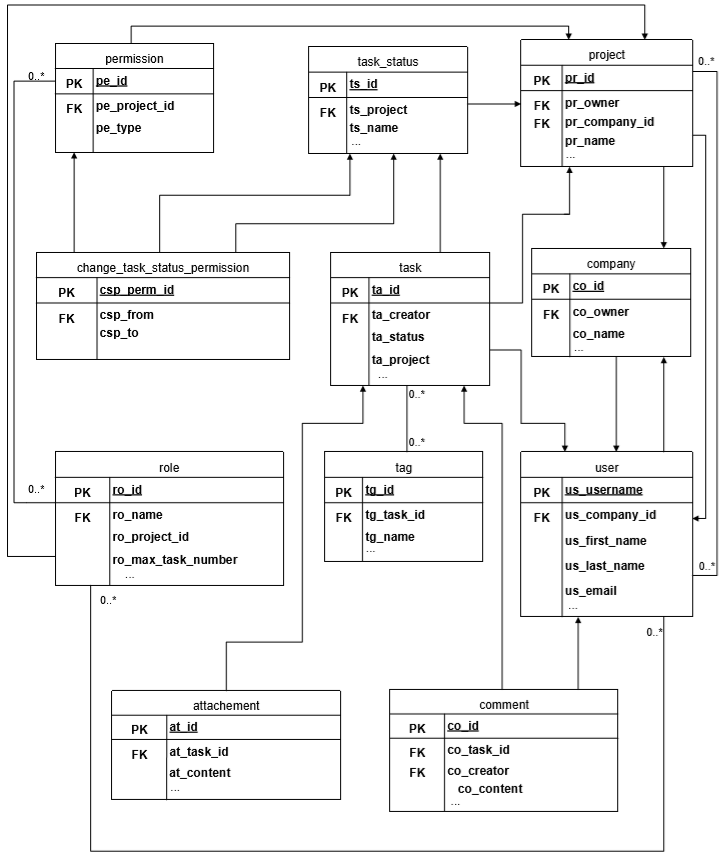
Таблица *change\_status\_permission* предназначена для хранения прав пользователей для смены статуса задачи. Поле *csp\_perm\_id* (идентификатор записи из таблицы *permission*) используется в качестве первичного ключа. Внешний ключ *csp\_status\_from* ссылается на таблицу *task\_status*, тем самым указывая статус задачи в каком статусе может быть изменен. Внешний ключ *csp\_status\_to* ссылается на таблицу *task\_status*, тем самым указывая на какой статус может быть изменен статус задачи. При удалении статуса, на который ссылается один из внешних ключей, запись из таблицы *change\_status\_permission* также будет удалена.

Рисунок 2.4 – Упрощенная модель данных

Таблица *logged\_work* используется для хранения отчетов о затраченном времени по конкретной задаче. Поле *lw\_id* является первичным ключом. Внешний ключ *lw\_username* ссылается на таблицу user, тем самым указывая пользователя составившего отчет.

Таблица *task\_history* используется для хранения истории изменения статуса задачи. Поле *th\_id* является первичным ключом. Внешний ключ *th\_task\_id* ссылается на таблицу task, тем самым указывая задачу. Внешний ключ *th\_username* ссылается на таблицу *user* и указывает пользователя, который внес изменения в статус задачи. Внешний ключ *th\_task\_status* указывает новый статус задачи.

## 2.3 Инфраструктура приложения

В приложении Г представлена диаграмма развертывания разрабатываемого программного средства.

Для работы приложения со стороны веб-сайта (клиент приложения) необходимо устройство с установленным веб-браузером. На серверной части приложения необходимы: контейнер сервлетов Tomcat, платформа Java SE Runtime Environment а так же СУБД MySQL (см. рис. 2.5).

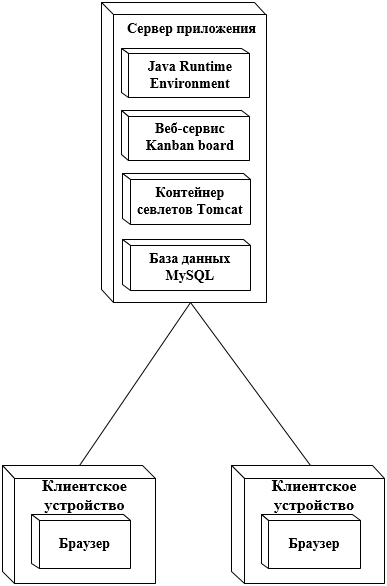


Рисунок 2.5 – Диаграмма развертывания

В данной главе была спроектирована структура программного средства и схема базы данных разрабатываемой системы. Были рассмотрены ключевые таблицы в схеме и установлены связи между ними. Также была спроектирована инфраструктуры разрабатываемого приложения и нарисованы структурная схема, модель данных и диаграмма развертывания.

**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробнее функциональные блоки, из которых состоит разрабатываемое программное средство, а также способы их реализации.

**3.1** Модуль аутентификации

Модуль аутентификации необходим для входа пользователей в систему. Аутентификация производится по логину и паролю, введенными пользователем. Так же данный модуль занимается управлением пользователями системы: регистрацией, обновлением профиля, удалением пользователей.

Данный модуль реализуется с использованием модуля Spring Security фреймворка Spring. Для реализации модуля необходимо всего три класса реализующих логику: *DefaultUserDetailsService*, *SecurityConfiguration, DatabaseUserDao*. И один класс сущности: *User*.

Класс *DefaultUserDetailsService* является реализацией интерфейса *UserDetailsService* модуля Spring Security и предоставляет метод *loadUserByUsername,* который принимает имя пользователя и возвращает объект *User* в случае успешной аутентификации или выбрасывает исключение если, пользователя с таким именем не существует. Данный метод в свою очередь обращается к методу *getUserByUsername* класса *DatabaseUserDao*, который возвращает объект *User*, если пользователь найден, или *null* в обратном случае.

Для настройки процесса аутентификации разработан класс *SecurityConfiguration,* которыйявляется местом для настройки процесса аутентификации. Класс унаследован от класса *WebSecurityConfigurerAdapter* и переопределяет метод *configure,* который принимает объект *HttpSecurity* для настройки процесса аутентификации. В листинге 3.1 приведена настройка аутентификации. Здесь указывается, что при *post*-запросе на url */login* будет осуществляться аутентификация пользователя. Параметр запроса с именем *username* содержит в себе имя пользователя, а параметр *password* содержит пароль пользователя. При успешной аутентификации клиенту придет ответ с кодом *202* (принято), иначе ответ с кодом *401* (не авторизован)*.* Ниже приведен листинг настройки аутентификации.

http.csrf().

Disable().

FormLogin().

loginPage("/login").

usernameParameter("username").

passwordParameter("password").

failureHandler(authenticationFailureHandler());

Второй метод класса *SecurityConfiguration configure* принимает объект типа *AuthenticationManagerBuilder.* В этом методе указывается *UserDetailsService,* который будет использован при аутентификации, так же указывается алгоритм хэширования пароля. Ниже приведен метод configure для настройки аутентификации.

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

Класс *SecurityConfiguration* помечен аннотацией *@Configuration*, тем самым указывается что данный класс является классом для настройки безопасности приложения. На рисунке 3.1 представлена схема реализации механизма аутентификации в Spring Security.

Рисунок 3.1 – Схема работы аутентификации Spring Security

Так же в данном модуле реализуется так называемая функция «remember me», которая позволяет пользователю не авторизовываться каждый раз при входе в приложение. Чтобы этого достичь ключ авторизации сохраняется в cookies пользовательского браузера.

Весь вышеизложенный функционал реализован с использованием Spring Security. Данный фреймворк необходимо включить в зависимости Maven. Ниже приведен листинг для подключения Spring Security в Apache Maven.

<dependency>

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

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

</dependency>

Помимо аутентификации пользователей данный модуль выполняет следующие функции: регистрация пользователей, редактирование профиля пользователей, выход из системы (log out) и удаление пользователя. Для реализации данного функционала используются классы классы: *AuthenticationController*, *DefaultUserDetailsService* и *DatabaseUserDao*.

Класс *DefaultUserDetailsService* был рассмотрен выше, однако с точки зрения реализации интерфейса *UserDetailsService.* Здесь этот класс рассматривается как реализация интерфейса *UserService.*

Класс *AuthenticationController* содержит в себе методы: *logout*, *update*, *delete.* Метод *logout* удаляет текущего пользователя из сессии и закрывает саму сессию.

Метод *update* принимает объект класса *User*, валидирует его и передает в метод *updateUser* класса *DefaultUserDetailsService.* После этот объект будет передан методу *udpateUser* класса *DatabaseUserDao* и будет произведено обновление записи о пользователе в базе данных.

Метод *delete* принимает логин пользователя, далее вызывается метод *deleteUser* класса DefaultUserService и метод *deleteUser* класса *DatabaseUserDao.*

Диаграмма классов настоящего модуля представлена на рисунке 3.2.

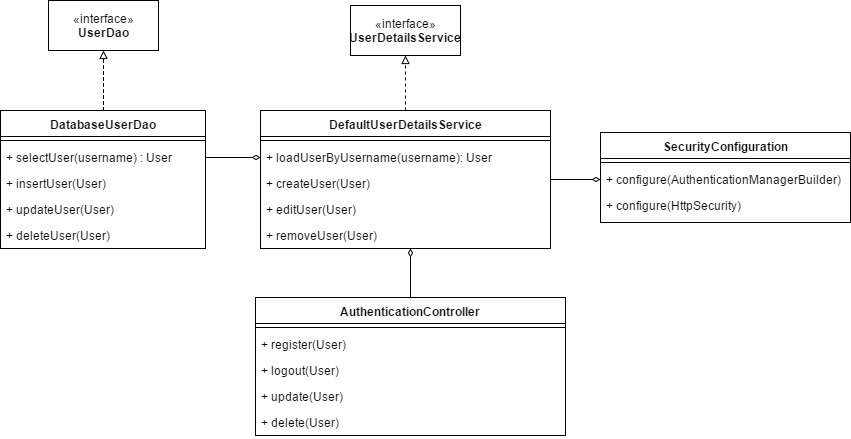


Рисунок 3.2 – Диаграмма классов модуля аутентификации

**3.2** Модуль отправки уведомлений

Данный модуль предназначен для отправки уведомлений пользователям по электронной почте. Класс *EmailNotificationService* реализует интерфейс *NotificationService.* В классе переопределены следующие методы: *send* (метод принимает объект класса *EmailMessage* и производит отправку сообщения по электронной почте), *sendUserRegistration* (метод принимает объект класса *User* и отправляет уведомление пользователю об успешной регистрации), *sendAddedToProjectMessage* (метод принимает объекты классов *Project* и *User* и отправляет пользователю уведомление, о том, что он был добавлен к проекту).

Так же в отдельный файл должны быть вынесены следующие свойства: smtp-хост, номер порта, электронныйй адрес и пароль отправителя сообщений. JavaMail позволяет производить отправку сообщений по следующим протоколам: SMTP, POP, IMAP, MIME.

Данный модуль реализуется с помощью библиотеки JavaMail, которая подключается с помощью добавления зависимости в файл *pom.xml.* Листинг для подключения JavaMail приведен ниже.

<dependency>

     <groupId>javax.mail</groupId>

     <artifactId>mail</artifactId>

     <version>1.4</version>

</dependency>



Рисунок 3.3 – Схема работы JavaMail

На рисунке 3.3 приведена схема работы библиотеки JavaMail. Здесь класс *Session* является базовым представлением сессии и отвечает за соединение с smtp-сервером. Класс *Message* представляет сообщение, которое может быть принято или отправлено, является абстрактным классом. Класс *Address* представляет собой адрес получателя или отправителя сообщения, являетсяя абстрактным классом. Класс *Transport* отвечает за работу по выбранному протоколу. Модуль *Store* и *Folder* создают сессию, соединяются с хранилищем в соответствии с указанными именем пользователя и паролем. На рисунке 3.4 представлена диаграмма классов данного модуля.

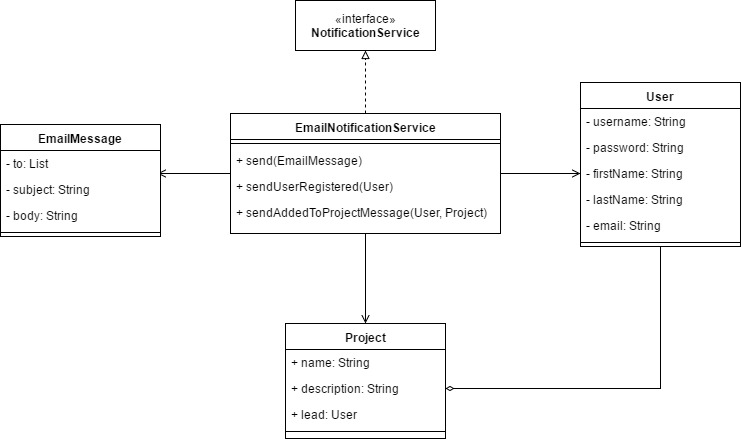
****

Рисунок 3.4 – Диаграмма классов модуля отправки уведомлений

## 3.3 Модуль построения отчетов

Данный модуль предназначен для получения отчета о проделанной работе по проекту. Суть построения отчета заключается в том, что пользователь получает информацию о задачах, находящихся в том или ином статусе.

Для реализации функционала необходимо выделить класс *ReportController*. Все методы класса возвращают список задач, выбранных по тому или иному критерию.

Метод класса *ReportController getTasks* принимает параметр *taskStatus*, и возвращает список объектов класса *Task* с этим статусом. Метод *getHistoryInfo* возвращает список объектов класса *TaskHistory.* В конечном итоге пользователь сможет увидеть изменения, касающиеся задач, которые были произведены на проекте. Так же имеет смысл добавить перегруженную версию метода *getHistoryInfo,* которая принимает параметры *endDate* и *startDate*, что позволяет получить историю изменения задач за определенный период. Метод *getFullStatastic* позволят узнать сколько задач находится в каждом из статусов.

Методы с аналогичными названиями присутствуют в классе *DefaultReportService.* На рисунке 3.5 представлена диаграмма классов модуля построения отчетов.

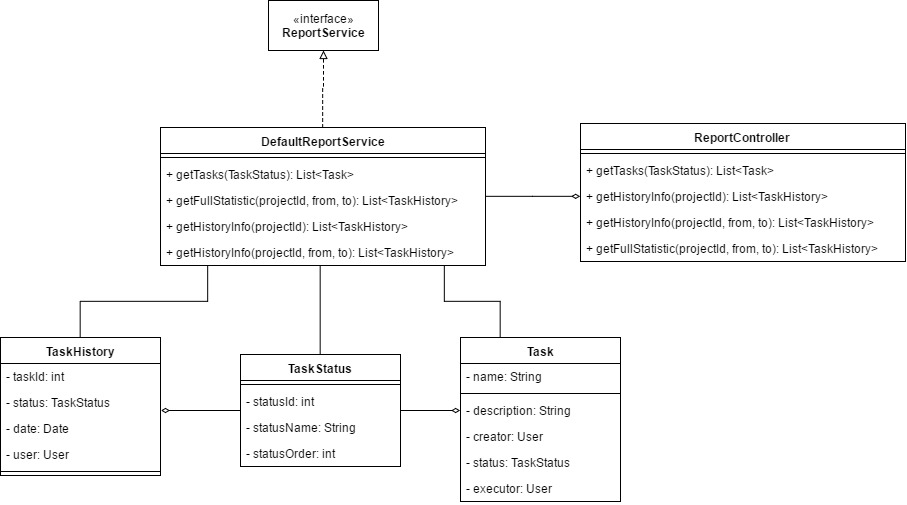


Рисунок 3.5 – Диаграмма классов модуля построения отчетов

Отдельно выделяется класс *TimeTrackController.* Этот класс предназначен для отчета пользователя о проделанной работе. Метод *logWork* принимает экземпляр класса *WorkItem* который представляет собой отчет о проделанной работе. Метод *getLoggedWork* принимает имя пользователя и позволяет получить отчет о проделанной работе для конкретного пользователя. Перегруженная версия этого метода принимает *endDate* и *startDate*, что позволяет получить отчет за определенный временной отрезок.

На рисунке 3.6 представлена диаграмма классов модуля отправки отчетов.

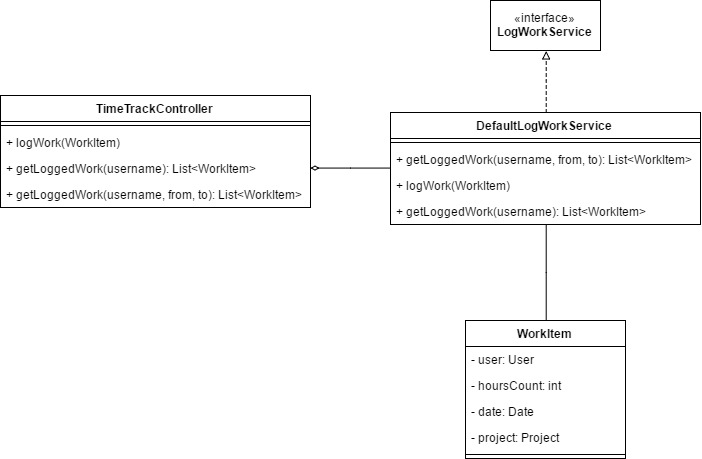


Рисунок 3.6 – Диаграмма классов модуля отправки отчетов

## 3.4 Модуль доступа к данным

Модуль доступа к данным является своего рода адаптером между бизнес логикой приложения и базой данных. В данном модуле можно выделить следующие классы: *DatabaseUserDao*, *DatabaseTaskDao*, *DatabaseProjectDao*, *DatabasePermissionDao*, *DatabaseTaskHistoryDao*, *DatabaseLogWorkDao*, *DatabaseTaskStatusDao*, *DatabaseGroupDao.*

Класс *DatabaseUserDao* является точкой доступа к данным связанных с информацией о пользователе. Данный класс реализует следующие методы: *getUserByUsername* (метод принимает строку с именем пользователя и возвращает объект *User*), *updateUser* (метод принимает экземпляр класса *User* и обновляет информацию о нём в базе данных), метод *createUser* (принимает экземпляр класса *User* и создаёт новую запись о пользователе в базе данных), *deleteUser* (принимает экземпляр класса User и удаляет соответствующую запись из базы данных).

Класс *DatabaseGroupDao* предоставляет интерфейс для доступа к данным, касающихся проектной группы. Класс реализует следующие методы: *createGroup* (метод принимает экземпляр класса *Group* и создает новую группу пользователей)*, addUserToGroup* (метод принимает идентификатор группы и имя пользователя и производит добавление пользователя в группу проекта)*, removeUserFromGroup* (метод принимет идентификатор группы и имя пользователя и удаляет пользователя из проектной группы)*, deleteGroup* (метод принимает экземпляр класса *Group* и удаляет группу пользователей), *getProjectGroups* (метод принимает идентификатор проекта и возвращает все группы на проекте)*.*  На рисунке 3.7 представлена диаграмма классов *DatabaseUserDao* и *DatabaseGroupDao*.

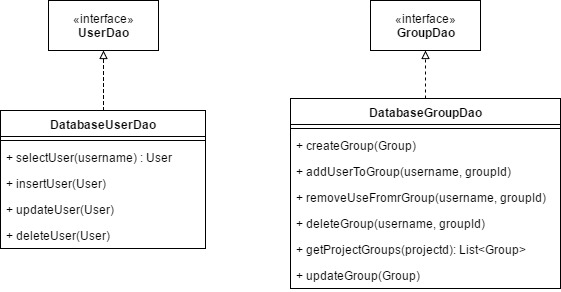


Рисунок 3.7 – Диаграмма классов DatabaseUserDao и DatabaseGroupDao

Класс *DatabaseTaskDao* (см. рис. 3.8) является точкой доступа к данным касающихся задачи. Данный класс реализует следующие методы: *getTasksWithStatus* (метод принимает экземпляр класса *Status* и возвращает список с указанным статусом), *getTask* (метод принимает идентификатор задачи и возвращает экземпляр класса *Task*), *updateTask* (метод принимает экземпляр класса *Task* и обновляет соответствующее поля в базе данных), *deleteTask* (метод принимает экземпляр класса *Task* и удаляет соответствующую задачу из базы данных), *addAttachement* (метод принимает экземпляр класса *Attachement* тем самым создается приложение к задаче), *addComment* (метод принимает экземпляр класса *Comment,* таким образом добавляется комментарий к задаче), *getTaskDetails* (метод принимает идентификатор задачи, и возвращает полную информацию о ней), *getProjectTasks* (метод принимает идентификатор проекта и идентификатор статуса задачи и возвращает задачи соответствующие переданным параметрам).

Класс *DatabaseProjectDao* (см. рис. 3.8)представляет собой точку для доступа к данным касающихся проекта. Данный класс реализует следующие методы методы: *getUsersProjects* (метод принимает экземпляр класса *User* и возвращает список проектов, над которыми работает указанный пользователь), *getProject* (метод принимает идентификатор проекта и возвращает экземпляр класса *Project*), *createProject* (метод принимает экземпляр класса *Project* и создает соответсвующую запись в базе данных), *deleteProject* (метод принимает экземпляр класса *Project* и удаляет проект из базы данных), *getUsersOnProject* (метод принимает идентификатор проекта и возвращает список пользователей участвующих в его разработке), *getLead* (метод принимает идентификатор проекта и возвращает экземпляр класса *User*, который символизирует руководителя проекта).

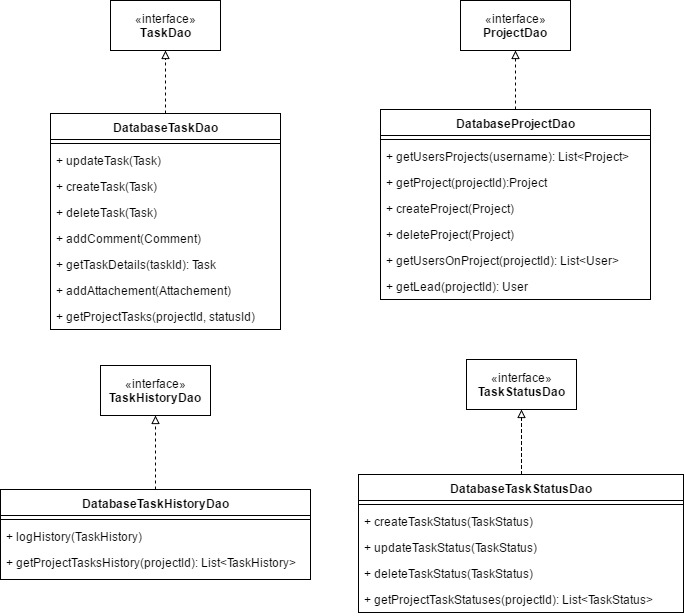
**

Рисунок 3.8 – Диаграмма классов для DatabaseTaskDao, DatabaseProjectDao, DatabaseTaskHistoryDao, DatabaseTaskStatusDao

Класс *DatabaseTaskHistoryDao* (см. рис. 3.8)предоставляет интерфейс для доступа к данным касающихся истории изменения задачи. Данный класс предоставляет следующие методы: *logHistory* (метод принимет экземпляр класса *History* и занимается созданием новой записи в истории задач), *getProjectTasksHistory* (метод принимает идентификатор проекта и возвращает список изменений задач).

Класс *DatabaseTaskStatusDao* (см. рис. 3.8)предоставляет интерфейс, который позволяет манипулировать данными, которые касаются статуса задачи. Данный класс реализует следующие методы: *createTaskStatus* (метод принимает экземпляр класса *Status* и создает новый статус задачи)*, updateTaskStatus* (метод принимает экземпляр класса *Status* и обновляет информацию о статусе задачи)*, deleteTaskStatus* (метод принимет экземпляр класса *Status* и удаляет статус задачи)*, getProjectTaskStatuses* (метод принимает идентификатор пректа и возвращает список статусов, в которых может находится задача).

Класс *DatabasePermissionDao* (см. рис. 3.9)предоставляет интерфейс по управлению правами пользователей на проекте. Класс реализует следующие методы: *createPermission* (метод принимает экземпляр класса *Permission* и создает новое право в базе данных), *deletePermission* (метод принимает экземпляр класса *Permission* и удаляет соответствующее право из базы), *assignPermissionToGroup* (метод принимает экземпляр класса *Permission* и идентификатор группы, тем самым предоставляя группе пользователей определеноое право), *takeOffPermissionFromGroup* (метод принимает экземпляр класса *Permission* и идентификатор проектной группы, тем самым удаляя права пользователя), *getGroupPermissions* (метод принимает идентификатор группы и возвращает список прав для нее).

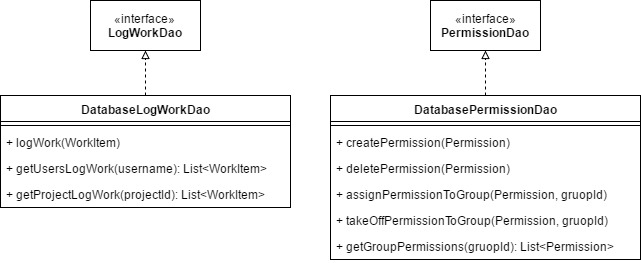


Рисунок 3.9 – Диаграмма классов для DatabaseLogWorkDao и DatabasePermissionDao

Класс *DatabaseLogWorkDao* (см. рис. 3.9)предоставляет интерфейс для доступа к данным, касающихся отчета о проделанной работе. Данный класс реализует следующие методы: *logWork* (метод принимает экземпляр класса *WorkItem* и создает запись о проделанной работе в базе данных), *getUsersLogWork* (метод принимает имя пользователя и возвращает список отчетов по временным затратам для конкретного пользователя), *getProjectLogWork* (метод принимает идентификатор проекта и возвращает список отчетов по проекту).

## 3.5 Модуль API веб-сервиса

Модуль API веб-сервиса является своего рода ядром разрабатываемой системы. Данный модуль принимает запросы от модуля маршрутизации фреймворка Spring, затем обращается, по необходимости, к другим модулям веб-сервиса, производит предварительную обработку ответа и посылает ответ модулю маршрутизации.

Разрабатываемый модуль представлен следующим множеством классов: *AuthenticationController, TemplateController, ProjectController, TaskController,*

*LogWorkController* (см. рис. 3.10)*.*

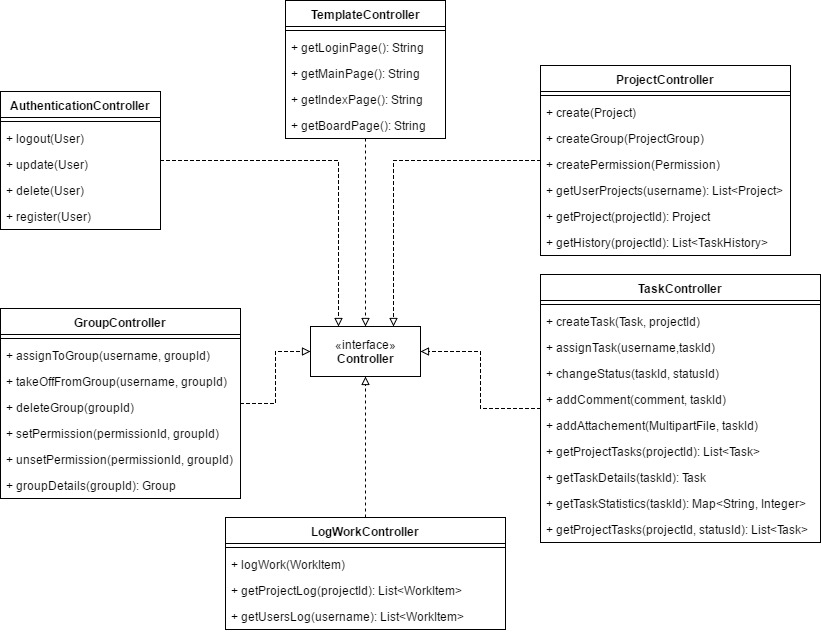


Рисунок 3.10 – Диаграмма классов модуля API веб-сервиса

Класс *AuthenticationController* предоставляет интерфейс по управлению пользователями. Класс реализует следующие методы: *logout* (метод принимает текущего пользователя и закрывает его сессию, таким образом пользователь выходит из системы), *updateProfile* (принимает экземпляр класса *User* и обновляет профиль пользователя), *deleteUser* (принимает экземпляр класса *User* и удаляет пользователя из системы).

Класс *TemplateController* предоставляет интерфейс по получению шаблонов веб-страниц. Метод *getLoginPage* возвращает название шаблона со страницей входа в систему. Метод *getMainPage* возвращает название шаблона с главной страницей приложения. Метод *getIndexPage* возвращает возвращает название шаблона главной страницы приложения. Метод *getBoardPage* возвращает название шаблона с доской проекта.

Класс *ProjectController* предоставляет интерфейс по управлению проектом. Метод *create* принимает экземпляр класса Project и создает новый проект в системе. Метод *createGroup* принимает объект класса ProjectGroup и добавляет новую группу в проект. Метод *assignToGroup* принимает имя пользователя и идентификатор группы, таким образом пользователь добавляется в проектную группу. Метод *takeOffFromGroup* принимает имя пользователя и идентификатор группы, таким образом пользователь удаляется из проектной группы. Метод *createPermission* принимает идентификатор проекта и экземпляр класса Permission, таким образом на проекте создается новый тип права. Метод *getUserProject* принимает имя пользователя и возвращает список проектов, над которыми работает пользователь. Метод *getProject* принимает идентификатор проекта ивозвращает полную информацию по проекту. Метод *getHistory* возвращает историю изменений в проекте.

Класс *GroupController* предоставляет интерфейс по управлению группами пользователей. Метод *assignToGroup* принимает имя пользователя и идентификатор группы, тем самым пользователь присоединятся к группе проекта. Метод *takeOffFromGroup* принимает имя пользователя и идентификатор группы и удаляет пользователя из группы. Метод *deleteGroup* принимает идентификатор группы, таким образом проектна группа удаляется. Методы *setPermission* и *unsetPermission* принимают идентификаторы группы и права и задают либо лишают права проектную группу. Метод *groupDetails* принимает принимает идентификатор группы и возвращает полную информацию о группе.

Класс *TaskController* предоставляет интерфейс по управлению задачами на проекте. Метод *createTask* принимает объект класса Task и идентификатор проекта, таким образом создается новая задача. Метод *assignTask* принимает имя пользователя и идентификатор задачи, тем самым пользователю назначается задача к исполнению. Метод *addComment* принимает содержимое комментария и идентификатор задачи, тем самым к задаче добавляется новый комментарий. Метод *addAttachement* принимает файл и идентификатор задачи, таким образом к задаче добавляется приложение. Метод *getProjectTasks* принимает идентификатор проекта и возвращает список задач на проекте, а его перегруженная версия вдобавок принимает идентификатор статуса задачи. Метод *getTaskStatistics* принимает идентификатор проекта и возвращает отображение где ключ – это название статуса задачи, а значение – количество задач в этом статусе.

Класс *LogWorkController* предоставляет интерфейс по управлению отчетами о проделанной работе. Метод *logWork* принимает экземпляр класса *WorkItem,* таким образом создается отчет о проделанной пользователем работе. Метод *getProjectLog* принимает идентификатор проекта и возвращает все отчеты по проекту. Метод *getUsersLog* принимает имя пользователя и возвращает список отчетов для конкретного пользователя.

## 3.6 Модуль маршрутизации фреймворка Spring

Данный модуль является своего рода точкой входа в веб-сервис со строны веб-сайта. Модуль представляет собой сервлет, который принимает http-запрос и делигирует его одному из контроллеров модуля API веб-сервиса.

Ключевую роль в данном модуле играет класс *DispatcherServlet.* Данный класс является сервлетом и способен принимать запросы по протоколу http. После того, как на этот сервлет поступил запрос с помощью класса *HandlerMapping* производится выбор: какому контроллеру делегировать полученный запрос. Контроллер представляет собой класс, помеченный аннотацией *@Controller.* Выбор контроллера основан на url по которому произошло обращение, параметрах запроса и http-методе. Чтобы указать соответствие между запросом и вызываемым методом используется аннотация *@RequestMapping.* Данной анотацией можно пометить как методы контроллера, так и сам класс. Здесь с помощью атрибута value можно указать url, с помощью атрибута method указывается метод http-запроса и с мощью атрибута requestParams можно указать параметры запроса при которых метод будет вызван.

## 3.7 Модуль управления задачами

Данный модуль предназначен для создания задач, смены статуса задачи поиска задач и т. д. Данный модуль представлен в виде следующих классов: *TaskController, DefaultTaskService, DatabaseTaskDao.* Диаграмма классов разрабатываемого модуля представлена на рисунке 3.11.

Класс *TaskController* является является контроллером, и при вызове его методов обработка запроса сразу же делегируется методу с такой же сигнатурой из класса *DefaultTaskService*.

Класс *DefaultTaskService* предоставляет интерфейс по управлению задачами на проекте. Метод *createTask* принимает объект класса Task и идентификатор проекта, таким образом создается новая задача. Метод *assignTask* принимает имя пользователя и идентификатор задачи, тем самым пользователю назначается задача к исполнению. Метод *addComment* принимает содержимое комментария и идентификатор задачи, тем самым к задаче добавляется новый комментарий. Метод *addAttachement* принимает файл и идентификатор задачи, таким образом к задаче добавляется приложение. Метод *getProjectTasks* принимает идентификатор проекта и возвращает список задач на проекте, а его перегруженная версия вдобавок принимает идентификатор статуса задачи. Метод *getTaskStatistics* принимает идентификатор проекта и возвращает отображение где ключ – это название статуса задачи, а значение – количество задач в этом статусе.

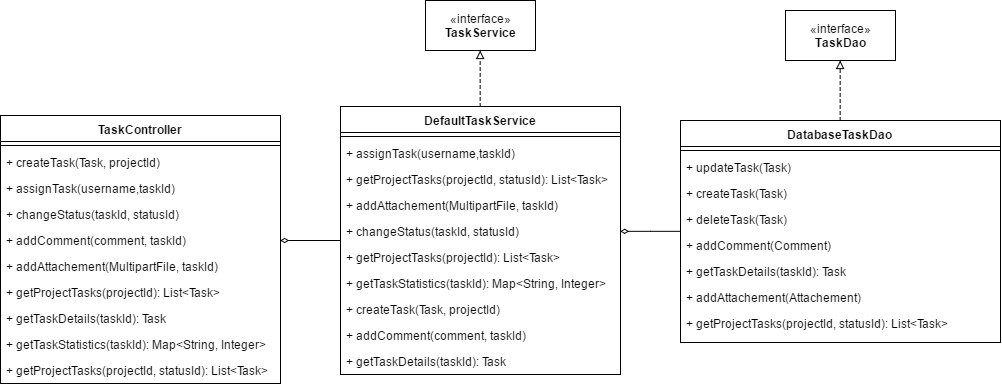


Рисунок 3.11 – Диаграмма классов модуля управления задачами

Класс *DatabaseTaskDao* является точкой доступа к данным касающихся задачи. Данный класс реализует следующие методы: *getTasksWithStatus* (метод принимает экземпляр класса *Status* и возвращает список с указанным статусом), *getTask* (метод принимает идентификатор задачи и возвращает экземпляр класса *Task*), *updateTask* (метод принимает экземпляр класса *Task* и обновляет соответствующее поля в базе данных), *deleteTask* (метод принимает экземпляр класса *Task* и удаляет соответствующую задачу из базы данных), *addAttachement* (метод принимает экземпляр класса Attachement тем самым создается приложение к задаче), *addComment* (метод принимает экземпляр класса *Comment,* таким образом добавляется комментарий к задаче), *getTaskDetails* (метод принимает идентификатор задачи, и возвращает полную информацию о ней), *getProjectTasks* (метод принимает идентификатор проекта и идентификатор статуса задачи и возвращает задачи соответствующие переданным параметрам).

## 3.8 Модуль администрирования и авторизации

В данном модуле сосредоточены функции получения прав пользователя, назначения прав пользователям, а также для администрирования системы в целом (создание проекта, назначение руководителя проекта). Авторизация производится с использованием модуля Spring Security фреймворка Spring. После создания проекта его руководитель создает проектную группу. Для каждой группы указываются её права: возможность создавать задачу, право на смену статуса задач, может ли пользователь выбирать задачу самостоятельно, может ли пользователь назначать задачи другим пользователям.

Авторизация осуществляется фреймворком Spring Security. Для его настройки необходимо унаследоваться от класса *WebSecurityConfigurerAdapter* и переопределить метод *configure.* Этот процесс описан в пункте 3.1 данного раздела. Здесь необходимо на объекте класса *HttpSecurity* вызвать метод *antMatchers* и передать в него url, и роль пользователя, который сможет пройти по этому url.

Однако необходимо защитить некоторые методы сервисов от доступа пользователей, которые не имеют права на это. Для этого над методом необходимо поставить анотацию *@PreAuthorize* и в ней в атрибуте *value* написать выражение на языке *SpEL,* которое проверяет, может данный пользователь вызывать этот метод или нет. В случает если выражение вернуло *false* клиенту придет ответ с кодом 403 (Доступ запрещен), иначе метод выполнится и, в случае необходимости, будет возвращен результат его выполнения.

Администрирование системы: создание проектов, создание групп проектов, создание прав пользователей др. Эти процессы сосредоточены в классах *ProjectController* и *TaskController.* Болееподробно эти классы и их методы рассмотрены в пункте 3.5 данной главы.

## 3.9 Модуль взаимодействия с веб-сервисом

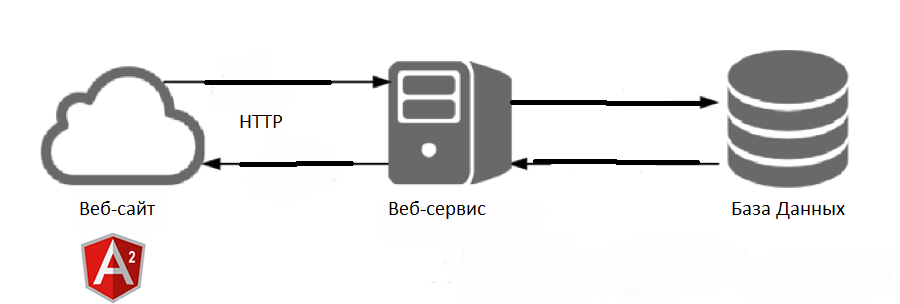
Модуль взаимодействия с веб-сервисом расположен на стороне веб-сайта и посылает запросы веб-сервису по протоколу http или https. Таким образом веб-сайт и веб-сервис обмениваются данными в формате JSON или XML. На рисунке 3.12 представлена схема взаимодействия веб-сайта и веб-сервиса.

Рисунок 3.12 – Схема взаимодействия в веб-сервисом

В Angular взаимодействие с веб-сервисом происходит с помощю зависимости *$http.* Данная зависимость позволяет отправлять сервису http-запросы по выбранному URL и с учетом http-метода. Например, при получении данных с веб сервиса вызывается метод $http.get(), в него передается URL, по этому URL к веб-сервису совершается запрос с http-методом GET. Для обработки ответа сервиса необходимо в метод then передать 2 функции. Первая функция вызовется при ответе сервиса с кодом 2xx, вторая вызовется при ответе сервиса с кодом 4xx или же при внутренней ошибке сервера 5xx.

## 3.10 Модуль рендеринга html-страниц

Модуль рендеринга html-страниц предназначен для отрисовки html-страниц по заданному шаблону. Шаблон страницы загружается с веб-сервиса один раз при первом обращении к шаблону. После этого шаблон сохраняется на стороне клиент и дальнейшая его выгрузка с веб-сервиса уже не требуется.

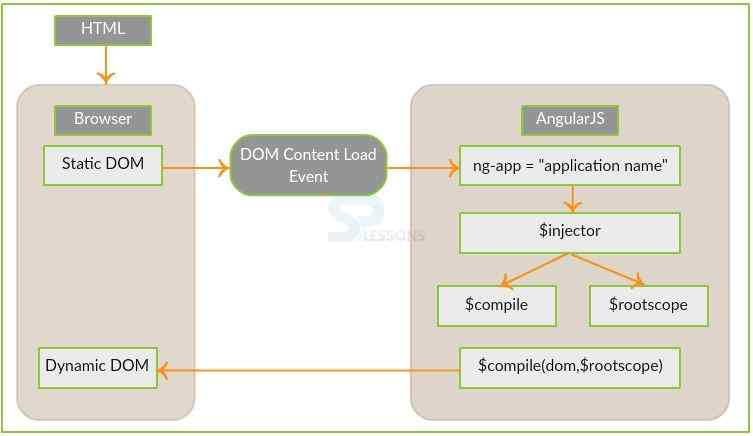


Рисунок 3.13 – Рендеринг страниц в Angular

На рисунке 3.13 представлена схема рендеринга html-страниц в фреймворке Angular.

Рендеринг html-страницы происходит следующим образом: веб-клиент выгружает с веб-сервиса html-страницу, которая является шаблоном. В местах, где нужно вставлять контент ставятся своего рода метки в виде двойных фигурных скобках и идентификатора объекта из модели ({{varName}}). После получения ответа веб-сервиса данные в формате JSON сохраняются в модели и автоматически переносятся на html-страницу.

## 3.11 Фреймворк Angular

Фреймворк Angular является основным модулем на стороне веб-сайта. Данный модуль управляет как процессом взаимодействия веб-сайта с веб-сервисом, так и процессом рендеринга html-страниц. В данном модуле подключаются следующие зависимости: *ngRoute* (необходима для динамической маршрутизации между страницами) и *dragular* для реализации функционала по перетаскиванию задач. Ниже приведен листинг использования *ngRoute* и *dragular*.

app.config(['$routeProvider', function ($routeProvider) {  
$routeProvider.when("/",  
{template : '<h1>Hello</h1>',  
controller : 'StartController'  
}).when("/main", {templateUrl : '/main',  
controller : 'MainPageController'  
}).when('/index', {templateUrl : '/index'});  
}]);

.controller('RemoveOnSpillWithModel', ['$scope', '$element', 'dragularService', function TodoCtrl($scope, $element, dragularService)

{

$scope.items1 = [{content: Str1'}, {content: ‘str2'}, {content: 'Item 3'}, {content: 'Item 4'}];

$scope.items2 = [{content: 'You can drop me in the left container.'}, {content: 'Item 6'}, {content: 'Item 7'}, {content: 'Item 8'}];

var containers = $element.children().eq(0).children(); dragularService.cleanEnviroment(); dragularService([containers[0],containers[1]],{ containersModel: [$scope.items1, $scope.items2], removeOnSpill: true });

}]);

В данной главе были рассмотрены более детально модули, составляющие разрабатываемое программное средство, так же была спроектирована диаграмма классов всего программного средства.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данной главе более детально будет рассмотрена реализация некоторых функций и процессов разрабатываемого программного средства.

## 4.1 Процесс создания задачи

Процесс создания задачи является одним из ключевых процессов, разрабатываемого программного средства. На рисунке 4.1 представлен алгоритм работы данного процесса в виде диаграммы активности.

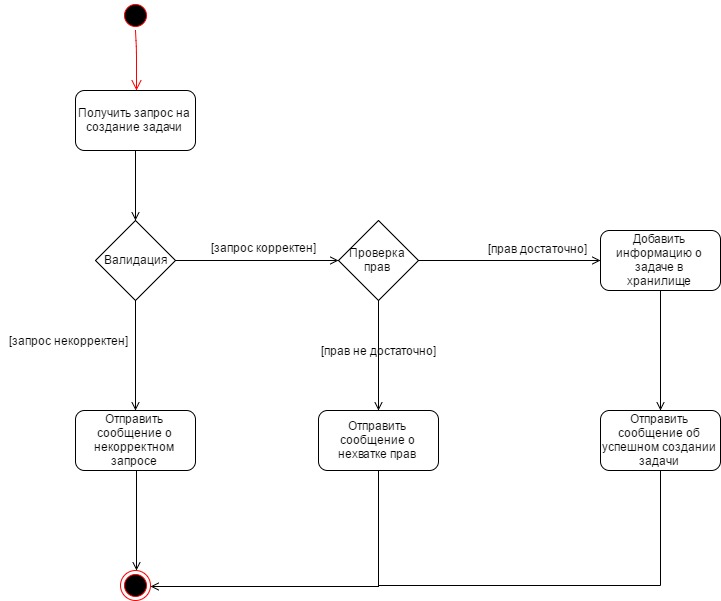


Рисунок 4.1 – Диаграмма активности процесса создания задачи

После получения запроса на создание задачи производится его валидация: у задачи должно присутствовать название, и его длина должна быть не больше восьмидесяти символов, у задачи должно быть описание не более 150 символов, у задачи должен быть указан проект, которому она принадлежит. Если валидация не была пройдена, на сторону клиента отправляется сообщение о некорректном запросе.

В случае успешного прохождения валидации выполняется проверка прав пользователя, чтобы выяснить может ли данный пользователь создавать задачи на данном проекте. В случае, если у пользователя недостаточно прав, ему отправляется сообщение об этом.

Если у пользователя оказалось достаточно прав для создания задачи, производится добавление задачи в базу данных, и отправка пользователю уведомления об успешном создании задачи.

## 4.2 Поиск задачи

Процесс поиска задач необходим для отображения задач, удовлетворяющих некоторому критерию. Поиск может осуществляться по тегам или по названию задачи. Поиск задач производится в рамках одного проекта. На рисунке 4.2 представлена диаграмма активности для процесса поиска задач.

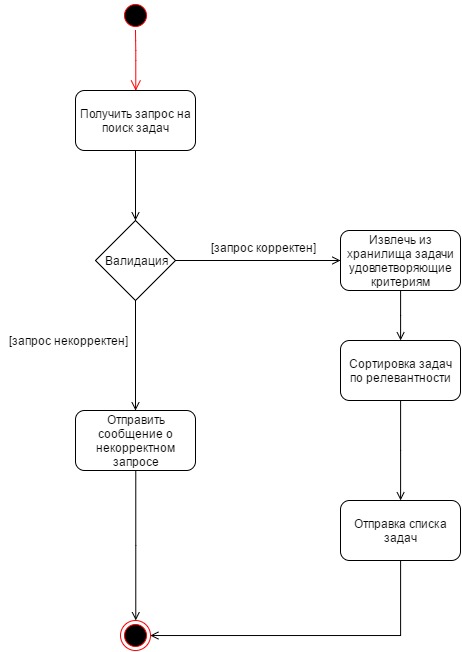


Рисунок 4.2 – Диаграмма активности процесса поиска задач

После получения запроса на поиск задач производится валидация запроса: строка запроса не должна быть пустой. Если вылидация не была пройдена, то пользователю отравляется уведомление о некорректно введенном запросе.

В случае успешного прохождения валидации из хранилища извлекаются все задачи, удовлетворяющие критериям поиска. Затем задачи сортируются по релевантности (по максимальному удовлетворению критериям запроса) и производится отправка списка задач клиенту. Ниже представлен SQL-запрос для поиска задач и метод уровня бизнес-логики searchTaks.

SELECT ta\_id, ta\_name, ta\_summary, ta\_description FROM task where ta\_name like :task\_name or   
ta\_id in (select tt\_task\_id from tag\_task inner join tag on lb\_id=tt\_tag\_id where lb\_name in (:tags)) and ta\_project\_id=:project\_id;

@Override  
public List<Task> searchTasks(Set<Tag> tags, String taskName, int projectId) {  
 List<Task> tasks = taskDao.searchTasks(tags, taskName, projectId);  
  
 for(Task task: tasks){  
 task.setTags(taskDao.getTaskTags(task.getId()));  
 }  
  
 Collections.sort(tasks, new RelevantTaskComparator(tags));  
  
 return tasks;  
}

## 4.3 Организация доступа к базе данных

В разрабатываемом продукте доступ к базе данных осуществляется с помощью модуля spring-jdbc фреймворка Spring. Для непосредственного исполнения SQL-запросов используется класс NamedParameterJdbcTemplate.

При извлечении данных таким способом необходимо реализовать интерфейс RowMapper и параметризировать его типом возвращаемого объекта. Ниже приведен листинг, где реализован интерфейс RowMapper для извлечения проектов из базы данных.

private final RowMapper<Project> projectRowMapper = (resultSet, i)->{

Project project = new Project(); project.setId(resultSet.getInt(TableColumn.PROJECT\_ID)); project.setName(resultSet.getString(TableColumn.PROJECT\_NAME));

project.setUsersNumber(resultSet.getInt(TableColumn.USER\_COUNT));

project.setDescription(resultSet.getString(TableColumn.PROJECT\_DESCRIPTION));

project.setTotalTaskCount(resultSet.getInt(TableColumn.TASK\_COUNT));

User lead = new User();

lead.setUsername(resultSet.getString(TableColumn.USERNAME));

lead.setFirstName(resultSet.getString(TableColumn.FIRST\_NAME));

lead.setLastName(resultSet.getString(TableColumn.LAST\_NAME));

project.setLead(lead);

return project;

};

В листинге выше осуществляется извлечение информации о проекте вместе с его лидером посредством извлечения данных из экземпляра класса ResultSet.

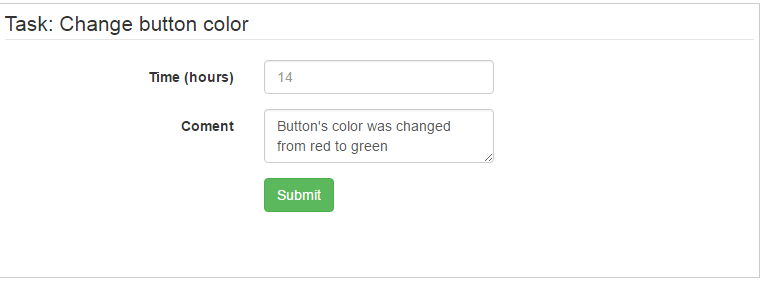
Для того чтобы выполнить запрос необходимо вызвать метод queryForList объекта класса NamedParameterJdbcTemplate и передать в него три параметра: SQL-запрос (параметры запроса должны быть отмечена как <:имя\_параметра>), одну из реализаций интерфейса SqlParameterSource (здесь отражается соответствие между именем параметра и его значением) и реализацию интерфейса RowMapper.

## 4.4 Построение отчетов о затраченном времени

Данный модуль предоставляет возможность сотрудникам составлять отчет о времени, которое они затратили на решение той или иной задачи. При построении отчета пользователь должен ввести время в часах, которое он затратил и описать в комментарии какая работа была произведена. На рисунке 4.3 приведена форма, где пользователь может отправить отчет.

На рисунке 4.4 представлена диаграмма активности для процесса построения отчета о затраченном времени.

После получения запроса на построение отчета производится его валидация. Проверяется введенное количество часов (число должно быть положительным), а также проверяется наличие комментария к отчету. Если валидация не была пройдена, то клиенту отправляется сообщение об ошибке. В противном случае производится проверка прав пользователя.



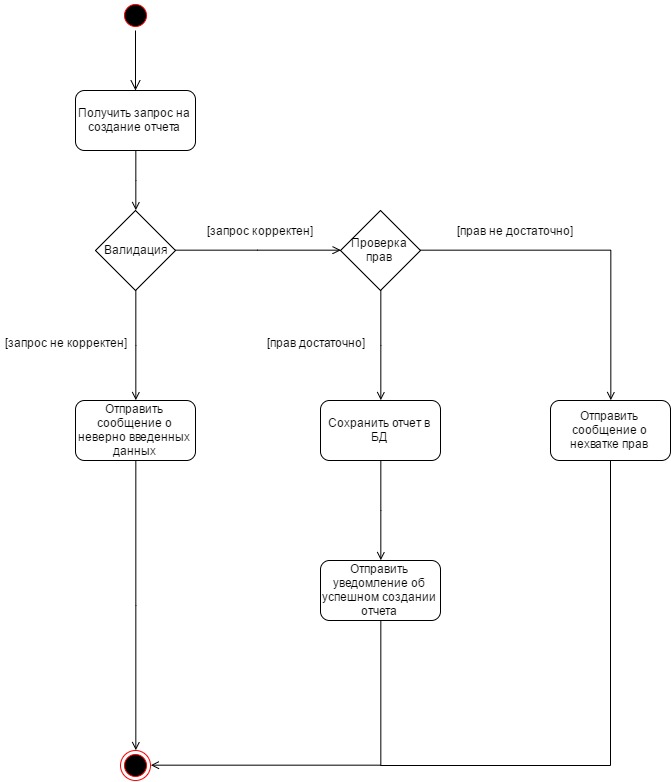
Рисунок 4.3 – Форма заполнения отчета о времени

Рисунок 4.4 – Диаграмма активности процесса построения отчетов

При проверке прав пользователя выяснятся, может ли данный пользователь списать свое рабочее время на данную задачу. Это необходимо для того, чтобы предотвратить списание рабочего времени на задачу, над которой пользователь не работал. Если у пользователя оказалось не достаточно прав, то ему отправляется уведомление об этом.

Если у пользователя достаточно прав, то производится сохранение отчета в базу данных, и отправка ему уведомления об успешном создании отчета.

## 4.5 Изменение статуса задачи

Изменение статуса задачи является одной из ключевых операций в разрабатываемом программном продукте. Изменение статуса производится с помощью перетаскивания задач из одной колонки в другую. Каждый пользователь, работающий над проектом, имеет определенный набор прав, определяющий из какого статуса в какой задача может быть перемещена. На рисунке 4.5 изображена диаграмма активности для данного процесса.

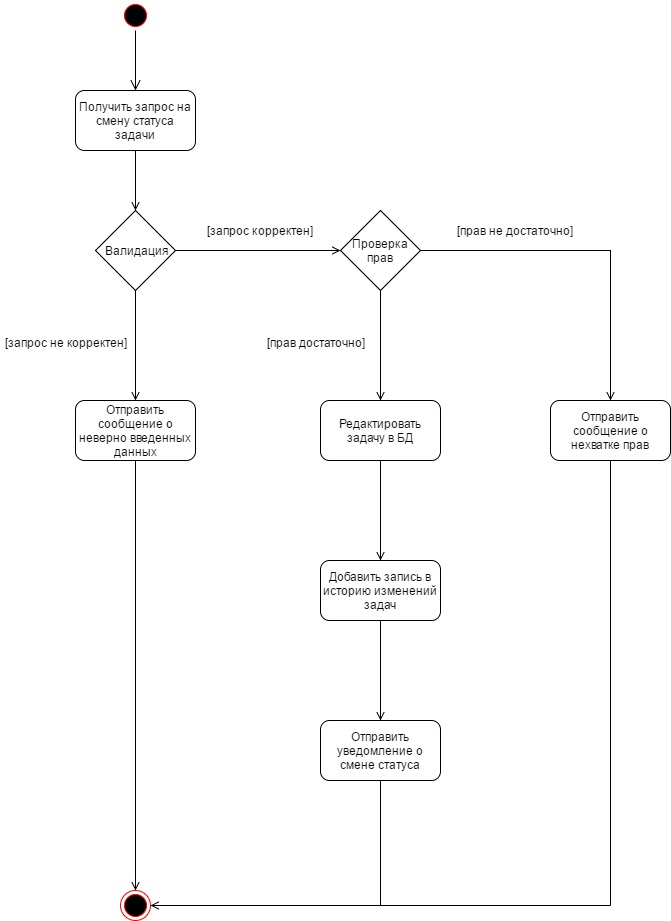


Рисунок 4.5 – Диаграмма активности процесса смены статуса задачи

После получения запроса на смену статуса задачи производится его валидация: у задачи должен быть указан ее идентификатор, новый и прежний статусы. В случае успешного прохождения валидации выполняется проверка прав пользователя, тем самым выясняется может ли он изменить статус задачи с текущего на новый. Если прав у пользователя достаточно, то задача обновляется в базе данных, добавляется запись в историю изменения задач, а пользователю приходит уведомление о смене статуса задачи. Ниже приведен фрагмент кода, обновляющий статус задачи.

@Override

@Transactional

@PreAuthorize("@permissionDao.canChangeStatus(principal.username, task.taskStatus, newStatus)")

public void changeTaskStatus(Task task, TaskStatus newTaskStatus){

taskDao.updateStatus(task, newTaskStatus);

HistoryItem historyItem = new HistoryItem(task, newTaskStatus);

taskHistoryDao.createHistory(historyItem);

}

Аннотация @Transactional говорит о том, что метод changeTaskStatus выполняется в рамках одной транзакции. Аннотация @PreAuthorize проверяет обладает ли пользователь достаточными правами для изменения статуса задачи. Метод updateStatus обновляет статус задачи в базе данных, а метод createHistory добавляет запись в историю изменения задач.

## 4.6 Добавление пользователя в проектную группу

Каждый созданный проект обладает набором проектных групп, в которые могут быть добавлены пользователи с одинаковой ролью (например, тестировщики, разработчики и др.). Каждая группа обладает своим списком прав, которые позволяют пользователям совершать те или иные действия. В данном подразделе описывается процесс добавления пользователя в проектную группу (см. рис. 4.6).

После получения запроса на создание проектной группы выполняется проверка прав пользователя, тем самым выясняется может ли данный пользователь добавлять участников в группу (добавление в группу может осуществить только руководитель проекта). Если прав достаточно, то производится редактирование списка пользователей, входящих в группу и отправляется уведомление об успешном добавлении пользователя. В обратном случае отправляется уведомление о нехватке прав, и пользователь увидит всплывающее окно уведомления.

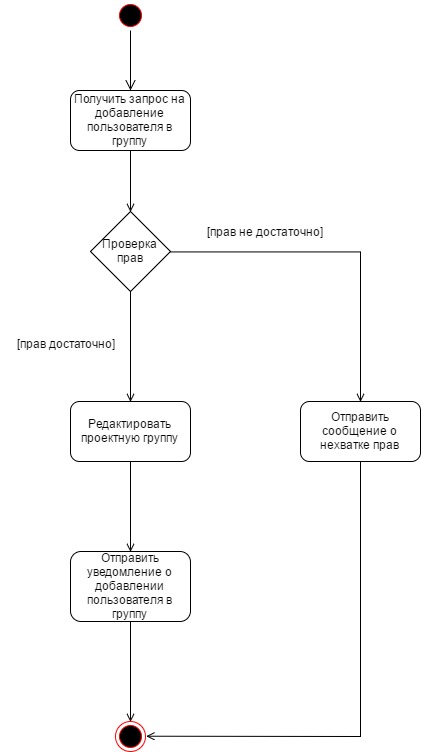


Рисунок 4.6 – Диаграмма активности процесса добавления пользователя в группу

Ниже приведен фрагмент кода, добавляющий пользователя к проектной группе.

Аннотация @Transactional говорит о том, что метод addUserToGroup выполняется в рамках одной транзакции. Аннотация @PreAuthorize проверяет обладает ли пользователь достаточными правами для изменения статуса задачи. Метод addToGroup добавляет пользователя в базу данных.

@Override

@Transactional

@PreAuthorize("@projectDao.getProjectLead(projectId).username == principal.username")

public void addUserToGroup(String username, int projectId, int groupId){

groupDao.addToGroup(username, groupId);

}

Аннотация @Transactional говорит о том, что метод addUserToGroup выполняется в рамках одной транзакции. Аннотация @PreAuthorize проверяет обладает ли пользователь достаточными правами для изменения статуса задачи. Метод addToGroup добавляет пользователя в базу данных.

## 4.7 Создание проектной группы

Проектная группа включает в себя набор пользователей, объединенных одной ролью и одинаковыми правами. Примером группы может служить группа тестировщиков или разработчиков. На рисунке 4.7 представлена диаграмма активности для процесса создания проектной группы.

Изначально со стороны клиента принимается запрос на создание новой проектной группы, далее производится проверка прав пользователя (только руководитель проекта может создавать группу). Если прав достаточно, то запись о созданной группе добавляется в базу данных, а пользователь получает уведомление, от том что группа была успешно создана. В противном случае пользователь получит уведомление о нехватке прав.

@Override

@Transactional

@PreAuthorize("@projectDao.getProjectLead(projectId).username == principal.username")

public void createGroup(Group group){

groupDao.createGroup(username, groupId);

}

В приведенном выше листинге @Transactional указывает, что данный метод выполняется в рамках одной транзакции. Аннотации @PreAuthorize передается выражение, которое проверяет является ли данный пользователь руководителем проекта. Если он не является таковым, то клиенту будет отправлено соответствующее уведомление с кодом ошибки 403, иначе в базу будет добавлена новая группа.

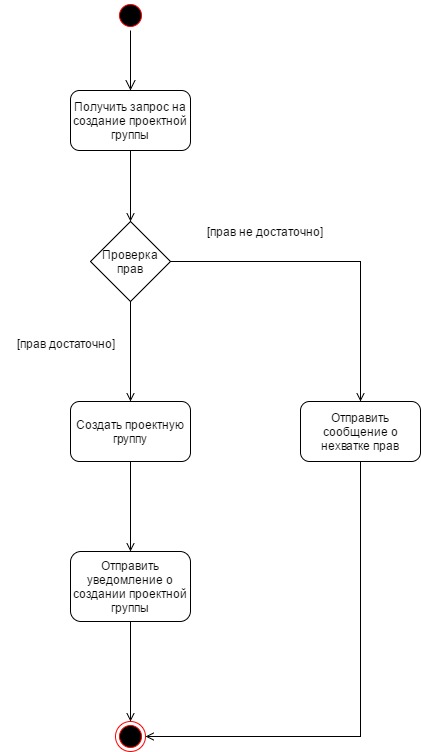


Рисунок 4.7 – Диаграмма активности процесса добавления пользователя в группу

## 4.8 Добавление прав к проектной группе

Как упоминалось выше у каждой группы имеется свой набор прав. Права бывают следующих типов: право на создание задачи, право на выбор задачи (пользователь сам выбирает задачу или ему кто-то должен выдать задачу) и право на изменение статуса. Однако, пользователю выдается право на «перетаскивание» задачи из одного конкретного статуса в другой конкретный статус. На рисунке 4.8 представлена диаграмма для данного случая.

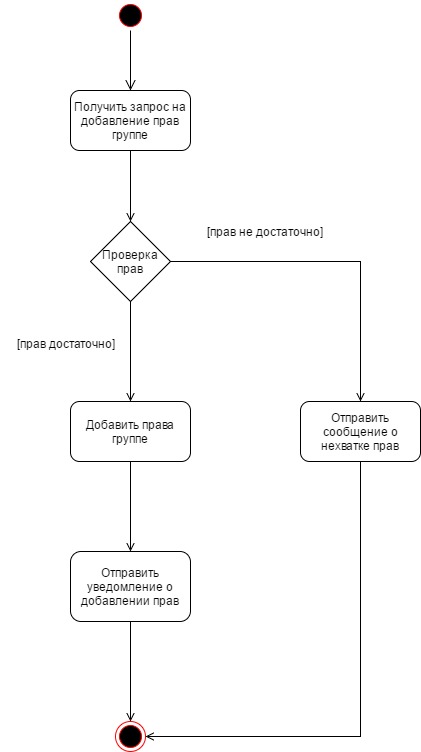


Рисунок 4.8 – Диаграмма активности добавления прав проектной группе

В данной главе были разработаны все модули, которые были спроектированы при системном и функциональном проектировании. Так же были построены диаграммы активности и была построена диаграмма последовательности для создания задачи и изменения статуса задачи.

**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Одним из основных этапов разработки программного средства является его тестирование. Тестирование ПО – это процесс выявление ошибок при разностороннем исследовании приложения. Выявление ошибок осуществляется путём сопоставления, реального и ожидаемого результатов тестов. В более широком смысле тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий две различные цели: продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям и выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Каждый этап разработки программного обеспечения сопровождается написанием большого количества разнообразных тестов. Ниже приведены основные виды тестирования:

— модульное тестирование;

— функциональное тестирование;

— тестирование производительности;

— тестирование совместимости;

— тестирование интерфейса пользователя.

Написание тестов необходимо для снижения рисков нарушения работоспособности приложения при внесении дополнительных изменений. Сам процесс написания тестов является довольно трудозатратным и требующим большого количества времени, но эти затраты окупаются надежностью и стабильности программного средства. По мере усложнения кода проекта стоимость устранения дефектов ПО может экспоненциально возрастать. Инструменты статического и динамического анализа помогают сократить эти затраты благодаря обнаружению программных ошибок на ранних этапах жизненного цикла ПО.

В ходе разработки настоящего программного средства были применены следующие виды тестирования: модульное и функциональное. Модульное тестирование производилось с использованием библиотеки Junit версии 4.12, функциональное проектирование производилось на персональном компьютере Lenovo G700 с установленной ОС Manjaro 17.0.1 Gellivara, процессором Intel Core i7 и оперативной памятью объемом 8 гигабайт.

**5.1** Функциональное тестирование

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях. При тестировании предполагается обработка данных и предсказуемая реакция приложения, когда данные, поданные на вход не являются корректными.

Результаты функционального тестирования приложения приведены в таблице 5.1.

Таблица 5.1 – Тест-кейсы и результаты их выполнения

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент | Название | Шаги и ожидаемый результат | Результат выполнения |
| 1 | 2 | 3 | 4 |
| Окно входа в систему | Аутентификация с корректным логином и паролем | 1 Ввести логин и пароль  2 Нажать кнопку «Login»  Ожидаемый результат: переход на главную страницу приложения | Тест пройден успешно |
| Окно регистрации пользователя | Регистрация пользователя | 1Ввести логин и пароль  2 Ввести адрес электронной почты  3 Ввести имя и фамилию  4 Нажать кнопку «Register»  Ожидаемый результат: получение уведомления об успешной регистрации | Тест пройден успешно |
| Окно доски проекта | Смена статуса задачи | 1 Перетащить задачу из статуса «New» в статус «In progress»  Ожидаемый результат: Отображение задачи в колонке «In progress» | Тест пройден успешно |
| Добавления комментария к задаче | 1 Нажать кнопку «Comment» на задаче  2 Ввести текст комментария  3 Нажать кнопку «Send comment»  Ожидаемы результат: Комментарий добавился к задаче | Тест пройден успешно |

Таблица 5.1 – Продолжение

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Окно доски проекта | Добавление приложения к задаче | 1 Нажать кнопку «Attach»  2 Выбрать файл  3 Нажать кнопку «Add»  Ожидаемый результат: добавление файла к приложениям задачи | Тест пройден успешно |
| Удаление задачи | 1 Начать кнопку «Х» на задаче  2 Задача удалена  Ожидаемый результат:  задача удалена из списка | Тест пройден успешно |
| Создание новой задачи при наличии необходимых прав | 1 Нажать кнопку «New task»  2 Заполнит поля с названием задачи и описанием  3 Нажать кнопку «Create»  4 Получить уведомление об успешном создании задачи  Ожидаемы результат:  создана новая задача | Тест пройден успешно |
| Окно построения отчетов | Получение отчета по количеству задач в каждом из статусов | 1 Нажать кнопку «Task progress»  2 Получить информацию о задачах на проекте  Ожидаемы результат:  отображен отчет по количеству задач в каждом из статусов | Тест пройден успешно |

Как видно из таблицы 5.1 все функциональные тесты были пройдены успешно. Ошибки, повлекшие за собой некорректное выполнение тестов, были оперативно устранены. В следующем пункте более детально будет рассмотрено модульное тестирование разработанного продукта.

**5.2** Модульное тестирование

Модульное тестирование – процесс в [программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), позволяющий проверить на корректность отдельные модули [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к [регрессии](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Основной целью модульного тестирования является разбиение разрабатываемой системы на модули и проверка работоспособности каждого модуля по отдельности. Если при вызове какого-либо метода определенного модуля есть необходимость обратиться к другому модулю, то вместо него подставляется так называемы mock-объект, который по своей сути является своего рода заглушкой.

При разработке настоящего программного средства была использована методология TDD (Test Driven Development). Суть этой методологии в том, что для разрабатываемого модуля изначально пишутся тесты, а позже пишется реализация конкретных методов модуля таким образом, чтобы тесты были пройдены. Разработка через тестирование требует от разработчика создания автоматизированных модульных тестов, определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, либо нет. Когда они выполняются, говорят, что тест пройден.

При разработке тестов была использована библиотека Junit версии 4.12. Для подключения библиотеки был использован фреймворк для автоматизации сборки проекта Apache Maven. Ниже приведен пример подключения данной библиотеки.

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

Полный перечень разработанных тестов приведен в приложении Д. Ниже рассматриваются тесты, покрывающие основной функционал данного программного продукта.

Для класса ProjectController ключевым методом является createProject. Данный метод принимает экземпляр класса Project и создает новый проект в системе. В первом случае рассмотрим поведение метода при передаче в него null. При передаче null метод должен выбросить исключение IllegalArgumentException. Ниже приведен листинг для данного теста.

@Test(expected = IllegalArgumentException.class)

public void createProjectTestWithEmptyProject(){

Project project = null;

ProjectController projectController = new ProjectController();

projectController.createProject(null);

}

Ниже приведен тест для случая, где не указан руководитель проекта. Ожидается, что при таком вызове метод выбросит исключение InvalidUserException.

@Test(expected = IllegalArgumentException.class)

public void createProjectTestWithoutLead(){

Project project = new Project();

project.setName(“New Project”);

ProjectController projectController = new ProjectController();

project.setLead(null);

projectController.createProject(project);

}

Для класса TaskController ключевыми методами являются методы createTask и changeTaskStatus.

В следующем тесте производится попытка создания новой задачи без указания проекта, которому она принадлежит. В этом случае ожидается, что метод выбросит исключение InvalidProjectException. Ниже приведен листинг для данного теста.

@Test(expected = InvalidProjectException.class)

public void createTaskTestWithoutProject(){

Task task = new Task();

task.setName(“Create new button”);

task.setDescription(“Description”);

task.setCreator(new User(“vladislav.zavadski”));

task.setProject(null);

TaskController taskController = new TaskController();

taskController.createTask(task);

}

В следующем тесте производится попытка создания задачи для проекта, которого не существует. В таком случае ожидается исключение ProjectNotFoundException. Ниже приведен листинг для данного теста.

@Test(expected = InvalidProjectException.class)

public void createTaskTestWithInvalidProject(){

Task task = new Task();

task.setName(“Create new button”);

task.setDescription(“Description”);

task.setCreator(new User(“vladislav.zavadski”));

Project project = new Project();

project.setId(666);

TaskController taskController = new TaskController();

taskController.createTask(task);

}

Далее будет рассмотрен случай изменения статуса задачи, на статус которого не существует. В данном случае следует ожидать, что метод changeTaskStatus выбросит исключение InvalidTaskStatusException. Листинг данного теста приведен ниже.

@Test(expected = InvalidTaskStatusException.class)

public void changeTaskStatusTestToInvalidStatus(){

Task task = new Task();

task.setId(10);

task.setTaskStatus(new TaskStatus(2));

TaskController taskController = new TaskController();

taskController.changeTaskStatus(task, new TaskStatus(3));

}

В следующем тесте производится попытка изменить статус задачи, которой не существует. В таком случае ожидается, что методом будет выброшено исключение TaskNotFoundException. Листинг для данного теста приведен ниже.

@Test(expected = InvalidTaskStatusException.class)

public void changeTaskStatusTestToInvalidStatus(){

Task task = new Task();

task.setId(666);

task.setTaskStatus(new TaskStatus(2));

TaskController taskController = new TaskController();

taskController.changeTaskStatus(task, new TaskStatus(3));

}

Далее смоделируем ситуацию поиска задач на проекте. В таком случае ожидается, что метод getProjectTasks вернет список всех задач на проекте. В конкретном случает задач всего 4. Листинг кода для данного теста приведен ниже.

@Test

public void getProjectTasksTest(){

TaskController taskController = new TaskController();

List<Task> tasks = taskController.getProjectTasks(2);

assertEquals(2, tasks.size());

}

В таблице 5.2 представлено покрытие тестами функционала наиболее важных модулей программного средства.

Таблица 5.2 – Покрытие модулей тестами

|  |  |
| --- | --- |
| Название модуля | Покрытие тестами |
| Модуль управления задачами | 6 случаев, ошибок не выявлено |
| Модуль отправки уведомлений | 3 случая, 1 ошибка исправлена |
| Модуль доступа к данным | 32 случая, 6 ошибок исправлено |
| Модуль аутентификации | 5 случаев, ошибок не выявлено |

Покрытие unit-тестами вышеперечисленных модулей позволило выявить и исправить 7 ошибок.

**5.3** Вывод

В данной главе были рассмотрены виды тестирования, которые использовались во время разработки программного средства. Использование функционального и модульного тестирования позволило выявить и устранить ошибки. Разработанное программное средство проходит тестовые испытания, что свидетельствует о его работоспособности.

**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ

ПРОГРАММНОГО СРЕДСТВА ОТСЛЕЖИВАНИЯ ЗАДАЧ

**7.1** Характеристика программного средства

Целью дипломного проекта является разработка программного средства управления задачами методом управления разработкой kanban. Целевыми пользователями данного программного продукта являются средние и малые IT-компании. С помощью этого программного средства пользователь получит следующие возможности:

– автоматизация управления проектом,

– отслеживание степени готовности проекта,

– отслеживание статистики по затраченному времени.

Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течение 4 лет; планируется продать 50 лицензий в 2017 году, 150 лицензий в 2018 году, 200 лицензий в 2019 году, 250 лицензий в 2020 году.

Экономическая целесообразность инвестиций в разработку и реализацию программного продукта определяется на основе расчета и оценке следующих показателей:

– чистый дисконтированный доход;

– срок окупаемости инвестиций;

– рентабельность инвестиций в разработку программного продукта.

**7.2** Расчет сметы затрат и отпускной цены на разработку программного средства

Расчет величины основной заработной платы участников команды определяется по формуле:

|  |  |
| --- | --- |
|  | (7.1) |

где *n* — количество исполнителей на конкретное программное средство;

— часовая тарифная ставка i-го исполнителя, руб.;

— количество рабочих часов в день, ч.;

— эффективный фонд рабочего времени i-го исполнителя, дн.;

— коэффициент премирования (можно принять ).

Примем тарифную ставку 1-го разряда равной 160,00 рублей. Среднемесячная норма рабочего времени составляет 168 часов.

Часовой тарифный оклад руководителя проекта с 15 разрядом составляет 160 ∙ 3,48 / 168 = 3,31 рубля.

Часовой тарифный оклад инженера-программиста 9 разряда составляет 160 ∙ 2,32 / 168 = 2,21 рубля.

Результаты расчета основной заработной платы исполнителей представлены в таблице 7.1.

Таблица 7.1 — Результаты расчета основной заработной платы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Исполнитель | Раз-ряд | Тарифный коэффици-ент | Месячная тарифная ставка, руб | Часовая тариф-ная ставка, руб | Плано-вый фонд рабочего времени, дн. | Заработная плата, руб |
| Руководитель проекта | 15 | 3,48 | 556,08 | 3,31 | 30 | 794,4 |
| Инженер-программист | 9 | 2,32 | 371,28 | 2,21 | 90 | 1591,30 |
| Итого |  |  |  |  |  | 2385,7 |
| Премия, руб. (50%) |  |  |  |  |  | 1192,85 |
| Основная заработная плата |  |  |  |  |  | 3578,55 |

Затраты на дополнительную заработную плату команды разработчиков () включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

|  |  |
| --- | --- |
|  | (7.2) |

где — норматив дополнительной заработной платы ().

В нашем случае, дополнительная зарплата будет равна:

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной заработной платы и вычисляются по формуле:

|  |  |
| --- | --- |
|  | (7.3) |

где — норматив отчисления на социальные нужды (34+0,6%).

Отчисления на социальные нужды составляют:

Расходы по статье «Машинное время» (), включающие оплаты машинного времени, необходимого для разработки и отладки программного продукта, осуществляется по формуле:

|  |  |
| --- | --- |
|  | (7.4) |

где — цена одного машино-часа, руб;

— количество часов работы в день, ч.;

— длительность проекта, дн.

Стоимость машино-часа на предприятии составляет 1,60 рублей. Разработка проекта займет 90 дней. Количество рабочих часов в день — 8. Таким образом затраты по статье «Машинное время» составят:

Расходы по статье «Прочие затраты» включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются в процентах к основной заработной плате.

|  |  |
| --- | --- |
|  | (7.5) |

где — норматив прочих затрат, %.

Возьмем норматив прочих расходов от основной заработной платы, тогда расходы по статье «Прочие затраты составят»:

Затраты по статье «Накладные расходы» (), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств, а также с расходами на общехозяйственные нужды, рассчитываются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где — норматив накладных расходов, 70%.

Тогда:

Общая сумма расходов по всем статьям сметы на программный продукт рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

Рассчитаем сумму расходов по всем статьям сметы:

Кроме того, потребуются дальнейшие затраты на сопровождение и адаптацию , которые определяются по нормативу :

|  |  |
| --- | --- |
|  | (7.8) |

где — норматив расходов на сопровождение и адаптацию, %;

— смета расходов без расходов на сопровождение и адаптацию, руб.

Примем равным 5%, тогда:

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программного продукта по формуле:

|  |  |
| --- | --- |
|  | (7.9) |

Общая сумма расходов на разработку:

Прибыль ПП рассчитывается по формуле:

где Пс – прибыль от реализации ПП заказчику;

Урп – уровень рентабельности ПП (15%);

Cп – себестоимость ПП.

Прогнозируемая отпускная цена ПП будет определена по следующей формуле:

Подставив значения в формулу получим прогнозируемую отпускную цену программного продукта:

**7.3** Расчет экономического эффекта от продажи программного продукта

Экономический эффект для разработчика программного обеспечения заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного программного средства.

Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течение 6 лет; планируется продать 50 лицензий в 2017 году, 150 лицензий в 2018 году, 200 лицензий в 2019 году, 250 лицензий в 2020 году. На основании маркетингового исследования отпускная цена одной копии лицензии составила 60 рублей.

Прибыль от продажи одной лицензии программного продукта определяется по формуле:

|  |  |
| --- | --- |
|  | (7.12) |

где Ц — отпускная цена одной копии лицензии программного продукта;

НДС — сумма налога на добавленную стоимость;

N — количество лицензий, которые купят клиенты;

— сумма расходов на разработку и реализацию.

Сумма налога на добавленную стоимость рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.13) |

где — ставка налога на добавленную стоимость, равняется 20 %.

Рассчитаем сумму налога на добавленную стоимость:

Затраты на реализацию примем как 15% от затрат на разработку. Тогда сумма расходов на разработку и реализацию будет равна:

Рассчитаем прибыль от продажи одной лицензии программного продукта по формуле (7.10):

Чистая прибыль от продажи одной лицензии программного продукта рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.14) |

где — ставка налога на прибыль, 18%.

Подставив данные в формулу (7.12) получаем чистую прибыль от продажи одной лицензии программного продукта:

Суммарная чистая годовая прибыль по проекту в целом рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.15) |

Прибыль по проекту за каждый год продаж составляет:

**7.4** Расчет показателей эффективности разработки программного продукта

Для проведения сравнительного анализа размера суммы затрат на разработку программного средства и получаемого экономического эффекта необходимо привести их к одному единому моменту времени — началу расчетного периода, что обеспечит их сопоставимость. Для этого необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования (α) соответствующего года t, который определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.16) |

где — норматив приведения разновременных затрат и результатов (нормативная ставка дисконта), в долях единицы в год;

— расчетный год, ;

— порядковый номер года.

На 01.05.2017 г. ставка рефинансирования составляет 15%. Используя формулу (7.13) рассчитаем коэффициенты дисконтирования:

2017 г.; ;

2018 г.; ;

2019 г.; ;

2020 г.; ;

Расчет показателей эффективности инвестиций по разработке продукта представлен в таблице 7.2.

Таблица 7.2 — Результаты расчета эффективности инвестиционного проекта по разработке программного продукта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатель | Единицы измерения | Расчётный период | | | |
| 2017 г. | 2018 г. | 2019 г. | 2020 г. |
| РЕЗУЛЬТАТ | | | | | |
| 1 Экономический эффект | руб. | 1640 | 4920 | 6560 | 8200 |
| Коэффициент дисконтирования | доли ед. | 1 | 0.87 | 0.76 | 0.65 |
| 2 Дисконтированный результат | руб. | 1640 | 4280.4 | 4985.6 | 5330 |
| 3 Затраты на разработку программного средства | руб. |  |  |  |  |
| 4 Дисконтированные инвестиции | руб. | 10031,78 |  |  |  |
| ЭКОНОМИЧЕСКИЙ ЭФФЕКТ | | | | | |
| 5 Чистый дисконтированный доход по годам | руб. | -8391.78 | 4280.4 | 4985.6 | 5330 |
| 6 Чистый дисконтированный доход нарастающий по годам | руб. | -8391.78 | -4111.38 | 874.22 | 6204.22 |

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта () по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

где — среднегодовая величина чистой прибыли за расчетный период.

Среднегодовая величина чистой прибыли за расчетный период определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.18) |

где — величина чистой прибыли за i-ый расчетный год;

n — расчетное количество лет.

Среднегодовая величина чистой составит:

Таки образом рентабельность инвестиций составит:

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей эффективности:  
 — чистый дисконтированный доход за четыре года составит 6204,22 руб.;

— затраты на разработку программного продукта окупятся на третий год его использования;

— рентабельность инвестиций составит 53 %.

Таким образом, разработка и реализация программного продукта является эффективной, а также является целесообразным осуществлять инвестиции в его разработку.

# ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом было реализовано веб-приложение, состоящее из веб-сайта и веб-сервиса. Был реализован следующий функционал:

– организация ведения проектов с использованием доски Kanban;

– построение отчетов о затраченном времени;

– поиск задач;

– создание проектов, ролей, управление списком пользователей.

Серверная часть разработанной системы реализована на языке Java, с использованием фреймворка Spring. Клиентская часть написана с использованием фреймворка Angular на языке JavaScript. В качестве СУБД использовалась MySQL.

При разработке дипломного проекта была проведена структурная декомпозиция, в ходе которой были выделены компоненты: модуль API веб-сервиса, модуль аутентификации, модуль отправки уведомлений, модуль построения отчетов, модуль маршрутизации фреймворка Spring, модуль управления задачами, модуль администрирования и авторизации, модуль доступа к данным, база данных веб-сервиса, модуль рендеринга html-страниц, модуль взаимодействия с веб-сервисом, фреймворк Angular. Так же была спроектирована база данных.

Вдобавок были спроектированы и разработаны вышеперечисленные модули. Были спроектированы классы, из которых состоит каждый из блоков. Была спроектирована диаграмма классов.

Также были рассмотрены виды тестирования, которые использовались во время разработки программного средства. Использование функционального и модульного тестирования позволило выявить и устранить ошибки. Разработанное программное средство проходит тестовые испытания, что свидетельствует о его работоспособности.

Были разработаны все модули, которые были спроектированы при системном и функциональном проектировании. Так же были построены диаграммы активности и была построена диаграмма последовательности для создания задачи и изменения статуса задачи.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое SPA [Электронный ресурс]. – 2017. – Режим доступа: <http://www.codenet.ru/webmast/js/spa/>.
2. AngularJS [Электронный ресурс]. – 2017. – Режим доступа: <https://ru.wikipedia.org/wiki/AngularJS>.
3. Краткое руководство по ReactJS [Электронный ресурс]. – 2017. – Режим доступа: https://habrahabr.ru/post/248799/.
4. Технологии Java для веб-разработки [Электронный ресурс]. – 2017. – Режим доступа: <https://www.techinfo.net.ru/docs/web/javawebdev.html>
5. Spring [Электронный ресурс]. – 2017. – Режим доступа: https://spring.io.
6. Java Server Faces [Электронный ресурс]. – 2017. – Режим доступа: <https://ru.wikipedia.org/wiki/JavaServer_Faces>.
7. Grails [Электронный ресурс]. – 2017. – Режим доступа: <https://ru.wikipedia.org/wiki/Grails>.
8. Документация СУБД SQLite [Электронный ресурс]. – 2017. – Режим доступа: <https://www.sqlite.org/docs.html>.
9. MySQL [Электронный ресурс]. – 2017. – Режим доступа: https://www.mysql.com.
10. Документация СУБД Postgres [Электронный ресурс]. – 2017. – Режим доступа: <https://postgresql.org/docs>.
11. Технико-экономическое обоснование дипломных проектов. – 2005. – В.А. Палицын.

# ПРИЛОЖЕНИЕ А

(обязательное)

Программное средство Kanban board. Исходный код.

Содержимое файла DatabaseProjectDao.java

package by.bsuir.kanban.dao.impl;

import by.bsuir.kanban.dao.ProjectDao;

import by.bsuir.kanban.domain.Project;

import by.bsuir.kanban.domain.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.stereotype.Repository;

import java.util.List;

/\*\*

\* Created by vladislav on 08.04.17.

\*/

@Repository

public class DatabaseProjectDao implements ProjectDao {

private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

private static final String GET\_USER\_PROJECTS\_QUERY = "SELECT pr\_id, pr\_name, pr\_description, us\_username, us\_first\_name, us\_last\_name, us\_count, ta\_count\n" +

"FROM project INNER JOIN project\_user ON pr\_id=pu\_project\_id INNER JOIN user ON pr\_lead=us\_username \n" +

"INNER JOIN (SELECT pr\_id AS pp\_iidd, count(\*) us\_count FROM project INNER JOIN project\_user ON pr\_id=pu\_project\_id WHERE pu\_username=:username GROUP BY pr\_id) AS uc \n" +

"ON pp\_iidd=pr\_id\n" +

"INNER JOIN (SELECT count(ta\_project) ta\_count, pu\_project\_id ppi FROM task\n" +

"RIGHT JOIN project\_user ON ta\_project=pu\_project\_id WHERE pu\_username=:username \n" +

"GROUP BY pu\_project\_id) AS tsk\n" +

"ON ppi=pr\_id\n" +

"WHERE pu\_username=:username LIMIT :startFrom, :limit;";

private static final String GET\_PROJECT\_QUERY = "select pr\_id, pr\_name, pr\_description, count(\*) as user\_number, us\_username, us\_first\_name, us\_last\_name\n" +

"from project inner join project\_user on pu\_project\_id=pr\_id inner join user on pr\_lead=us\_username where pu\_username = ?";

private static final String INSERT\_PROJECT\_QUERY = "insert into project (pr\_name, pr\_lead, pr\_company\_id) values (:name, :lead.username, :company.id)";

private final RowMapper<Project> projectRowMapper = (resultSet, i)->{

Project project = new Project();

project.setId(resultSet.getInt(TableColumn.PROJECT\_ID));

project.setName(resultSet.getString(TableColumn.PROJECT\_NAME));

project.setUsersNumber(resultSet.getInt(TableColumn.USER\_COUNT));

project.setDescription(resultSet.getString(TableColumn.PROJECT\_DESCRIPTION));

project.setTotalTaskCount(resultSet.getInt(TableColumn.TASK\_COUNT));

User lead = new User();

lead.setUsername(resultSet.getString(TableColumn.USERNAME));

lead.setFirstName(resultSet.getString(TableColumn.FIRST\_NAME));

lead.setLastName(resultSet.getString(TableColumn.LAST\_NAME));

project.setLead(lead);

return project;

};

@Autowired

public DatabaseProjectDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public List<Project> getUsersProjects(String username, int limit, int startFrom){

MapSqlParameterSource sqlParameterSource = new MapSqlParameterSource();

sqlParameterSource.addValue("username", username);

sqlParameterSource.addValue("startFrom", startFrom);

sqlParameterSource.addValue("limit", limit);

return namedParameterJdbcTemplate.query(GET\_USER\_PROJECTS\_QUERY, sqlParameterSource, projectRowMapper);

}

@Override

public Project getProject(int projectId){

return namedParameterJdbcTemplate.getJdbcOperations().queryForObject(GET\_PROJECT\_QUERY, new Object[]{projectId}, projectRowMapper);

}

@Override

public void createProject(Project project){

BeanPropertySqlParameterSource beanPropertySqlParameterSource = new BeanPropertySqlParameterSource(project);

namedParameterJdbcTemplate.update(INSERT\_PROJECT\_QUERY, beanPropertySqlParameterSource);

}

}

Содержимое файла DatabaseTaskDao.java

package by.bsuir.kanban.dao.impl;

import by.bsuir.kanban.dao.TaskDao;

import by.bsuir.kanban.domain.Priority;

import by.bsuir.kanban.domain.Status;

import by.bsuir.kanban.domain.Task;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.stereotype.Repository;

import java.util.List;

/\*\*

\* Created by vladislav on 09.04.17.

\*/

@Repository

public class DatabaseTaskDao implements TaskDao {

private static final String GET\_PROJECT\_TASKS\_QUERY = "select ta\_id, ta\_name, ta\_summary, ta\_priority, ta\_status, ta\_order, ts\_order, ts\_name from task inner join task\_status on ts\_id=ta\_status where ta\_project = :projectId and ta\_status = :taskStatusId order by ts\_order, ta\_priority, ta\_order limit :startFrom, :rowNumber;";

private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

private static final RowMapper<Task> taskRowMapper = ((resultSet, i) -> {

Task task = new Task();

task.setId(resultSet.getInt(TableColumn.TASK\_ID));

task.setName(TableColumn.TASK\_NAME);

task.setSummary(TableColumn.TASK\_SUMMARY);

task.setPriority(Priority.valueOf(resultSet.getString(TableColumn.TASK\_PRIORITY)));

task.setOrder(resultSet.getInt(TableColumn.TASK\_ORDER));

Status status = new Status();

status.setName(resultSet.getString(TableColumn.TASK\_STATUS\_NAME));

status.setOrder(resultSet.getInt(TableColumn.TASK\_STATUS\_ORDER));

status.setId(resultSet.getInt(TableColumn.TASK\_STATUS\_ID));

task.setTaskStatus(status);

return task;

});

@Autowired

public DatabaseTaskDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

@Override

public List<Task> getProjectTasks(int projectId, int taskStatusId, int startFrom, int limit) {

MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();

mapSqlParameterSource.addValue(ParameterName.PROJECT\_ID, projectId);

mapSqlParameterSource.addValue(ParameterName.TASK\_STATUS\_ID, taskStatusId);

mapSqlParameterSource.addValue(ParameterName.START\_FROM, startFrom);

mapSqlParameterSource.addValue(ParameterName.ROW\_NUMBER, limit);

return namedParameterJdbcTemplate.query(GET\_PROJECT\_TASKS\_QUERY, mapSqlParameterSource, taskRowMapper);

}

}

Содержимое файла DatabaseTaskStatusDao.java

package by.bsuir.kanban.dao.impl;

import by.bsuir.kanban.dao.TaskStatusDao;

import by.bsuir.kanban.domain.Status;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.stereotype.Repository;

import java.util.List;

/\*\*

\* Created by vladislav on 09.04.17.

\*/

@Repository

public class DatabaseTaskStatusDao implements TaskStatusDao {

private static final String GET\_AVAILABLE\_STATUS\_QUERY = "SELECT ts\_id as ta\_status from task\_status WHERE ts\_project = ? ORDER BY ts\_order;";

private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

private static final RowMapper<Status> taskStatusRowMapper = ((resultSet, i) -> {

Status status = new Status();

status.setId(resultSet.getInt(TableColumn.TASK\_STATUS\_ID));

return status;

});

@Autowired

public DatabaseTaskStatusDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate){

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

@Override

public List<Status> getAvailableProjectStatus(int projectId) {

return namedParameterJdbcTemplate.getJdbcOperations().query(GET\_AVAILABLE\_STATUS\_QUERY, new Object[]{projectId}, taskStatusRowMapper);

}

}

Содержимое файла DatabaseUserDao.java

package by.bsuir.kanban.dao.impl;

import by.bsuir.kanban.dao.UserDao;

import by.bsuir.kanban.domain.Company;

import by.bsuir.kanban.domain.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Lookup;

import org.springframework.dao.EmptyResultDataAccessException;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.stereotype.Repository;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

/\*\*

\* Created by ulza1116 on 4/7/2017.

\*/

@Repository

public class DatabaseUserDao implements UserDao {

private static final String GET\_USER\_QUERY = "select us\_username, us\_password, us\_last\_name, us\_first\_name, us\_email, us\_can\_create\_project, co\_id, co\_name from user inner JOIN company on co\_id=us\_company\_id where us\_username = ?";

private static final String GET\_IMAGE\_QUERY = "SELECT us\_picture from user where us\_username = ?";

private static final String GET\_USERS\_ON\_PROJECT\_QUERY = "select us\_first\_name, us\_last\_name, us\_username, us\_picture from users inner join project\_user on pu\_username=us\_username where pu\_project\_id = :projectId order by us\_last\_name limit :startFrom, :rowNumber";

private static final String INSERT\_USER\_QUERY = "insert into user (us\_username, us\_email, us\_first\_name, us\_last\_name, us\_password, us\_picture, us\_can\_create\_project, us\_company\_id) VALUES (:username, :email, :firstName, :lastName, :password, :picture, :canCreateProject, :company.id);";

private static final String UPDATE\_USER\_QUERY = "UPDATE user SET us\_first\_name = :firstName, us\_last\_name = :lastName, us\_password = :password, us\_email = :email WHERE us\_username = :username";

private static final String SELECT\_USERS\_FROM\_COMPANY = "select us\_username, us\_first\_name, us\_last\_name, us\_picture from user where us\_company\_id=?;";

private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

private static final RowMapper<User> userRowMapper = ((resultSet, i) -> {

User user = new User();

user.setUsername(resultSet.getString(TableColumn.USERNAME));

user.setPassword(resultSet.getString(TableColumn.PASSWORD));

user.setFirstName(resultSet.getString(TableColumn.FIRST\_NAME));

user.setLastName(resultSet.getString(TableColumn.LAST\_NAME));

user.setEmail(resultSet.getString(TableColumn.EMAIL));

user.setCanCreateProject(resultSet.getInt(TableColumn.CREATE\_PROJECT) != 0);

user.setAuthorities(Collections.emptyList());

Company company = new Company();

company.setId(resultSet.getInt(TableColumn.COMPANY\_ID));

company.setName(resultSet.getString(TableColumn.COMPANY\_NAME));

user.setCompany(company);

return user;

});

private final RowMapper<User> userOnProjectRowMapper = ((resultSet, i) -> {

User user = new User();

user.setUsername(resultSet.getString(TableColumn.USERNAME));

user.setFirstName(resultSet.getString(TableColumn.FIRST\_NAME));

user.setLastName(resultSet.getString(TableColumn.LAST\_NAME));

user.setPicture(resultSet.getString(TableColumn.PICTURE));

return user;

});

@Autowired

public DatabaseUserDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public User getUserByUsername(String username) {

try {

return namedParameterJdbcTemplate.getJdbcOperations().queryForObject(GET\_USER\_QUERY, new Object[]{username}, userRowMapper);

}

catch (EmptyResultDataAccessException ex){

return null;

}

}

public String getImagePath(String username){

try {

return namedParameterJdbcTemplate.getJdbcOperations().queryForObject(GET\_IMAGE\_QUERY, new Object[]{username}, String.class);

}

catch (EmptyResultDataAccessException ex){

return null;

}

}

@Override

public List<User> getUsersOnProject(int projectId, int startFrom, int limit){

MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();

mapSqlParameterSource.addValue(ParameterName.PROJECT\_ID, projectId);

mapSqlParameterSource.addValue(ParameterName.START\_FROM, startFrom);

mapSqlParameterSource.addValue(ParameterName.ROW\_NUMBER, limit);

return namedParameterJdbcTemplate.query(GET\_USERS\_ON\_PROJECT\_QUERY, mapSqlParameterSource, userOnProjectRowMapper);

}

@Override

public void createUser(User user){

SqlParameterSource sqlParameterSource = new BeanPropertySqlParameterSource(user);

namedParameterJdbcTemplate.update(INSERT\_USER\_QUERY, sqlParameterSource);

}

@Override

public void updateUser(User user){

SqlParameterSource sqlParameterSource = new BeanPropertySqlParameterSource(user);

namedParameterJdbcTemplate.update(UPDATE\_USER\_QUERY, sqlParameterSource);

}

@Override

public List<User> selectUsersByCompany(int companyId){

return namedParameterJdbcTemplate.getJdbcOperations().query(SELECT\_USERS\_FROM\_COMPANY, new Object[]{companyId}, userOnProjectRowMapper);

}

}

Содержимое файла DatabaseUserProjectDao.java

package by.bsuir.kanban.dao.impl;

import by.bsuir.kanban.dao.UserProjectDao;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.stereotype.Repository;

/\*\*

\* Created by vladislav on 09.04.17.

\*/

@Repository

public class DatabaseUserProjectDao implements UserProjectDao {

private static final String IS\_ASSIGN\_QUERY = "SELECT count(\*) from project\_user WHERE pu\_project\_id = :projectId and pu\_username = :username;";

private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Autowired

public DatabaseUserProjectDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

@Override

public boolean isUserAssignOnProject(String username, int projectId) {

MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();

mapSqlParameterSource.addValue(ParameterName.PROJECT\_ID, projectId);

mapSqlParameterSource.addValue(ParameterName.USERNAME, username);

return namedParameterJdbcTemplate.queryForObject(IS\_ASSIGN\_QUERY, mapSqlParameterSource, Integer.class) != 0;

}

}

Содержимое файла RelevantTaskComparator.java

package by.bsuir.kanban.service.util;

import by.bsuir.kanban.domain.Tag;

import by.bsuir.kanban.domain.Task;

import java.util.Comparator;

import java.util.Set;

/\*\*

\* Created by vladislav on 22.05.17.

\*/

public class RelevantTaskComparator implements Comparator<Task> {

private static final int GREATER = 1;

private static final int EQUALS = 0;

private static final int LESS = -1;

private Set<Tag> searchedTasks;

public RelevantTaskComparator(Set<Tag> searchedTasks) {

this.searchedTasks = searchedTasks;

}

@Override

public int compare(Task task1, Task task2) {

int task1ContainsCount = 0;

int task2ContainsCount = 0;

for(Tag tag: searchedTasks){

if(task1.getTags().contains(tag)){

task1ContainsCount++;

}

if(task2.getTags().contains(tag)){

task2ContainsCount++;

}

}

if(task1ContainsCount < task2ContainsCount){

return LESS;

}

else if(task1ContainsCount > task2ContainsCount){

return GREATER;

}

else {

return EQUALS;

}

}

}

Содержимое файла MailNotificationSender.java

package by.bsuir.kanban.service.impl;

import by.bsuir.kanban.domain.Notification;

import by.bsuir.kanban.service.NotificationSender;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Service;

/\*\*

\* Created by ulza1116 on 5/2/2017.

\*/

@Service

public class MailNotificationSender implements NotificationSender {

private JavaMailSender javaMailSender;

@Autowired

public MailNotificationSender(JavaMailSender javaMailSender){

this.javaMailSender = javaMailSender;

}

@Override

@Async

public void sendNotification(Notification notification) {

SimpleMailMessage simpleMailMessage = new SimpleMailMessage();

simpleMailMessage.setFrom(notification.getFrom());

simpleMailMessage.setTo(notification.getTo());

simpleMailMessage.setSubject(notification.getSubject());

simpleMailMessage.setText(notification.getBody());

javaMailSender.send(simpleMailMessage);

}

}

Содержимое файла DefaultProjectService.java

package by.bsuir.kanban.service.impl;

import by.bsuir.kanban.dao.ProjectDao;

import by.bsuir.kanban.dao.TaskDao;

import by.bsuir.kanban.dao.TaskStatusDao;

import by.bsuir.kanban.dao.UserDao;

import by.bsuir.kanban.domain.Project;

import by.bsuir.kanban.domain.Status;

import by.bsuir.kanban.domain.Task;

import by.bsuir.kanban.domain.User;

import by.bsuir.kanban.service.ProjectService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* Created by vladislav on 08.04.17.

\*/

@Service

public class DefaultProjectService implements ProjectService {

private final ProjectDao projectDao;

private final TaskDao taskDao;

private final TaskStatusDao taskStatusDao;

private final UserDao userDao;

@Autowired

public DefaultProjectService(ProjectDao projectDao, TaskDao taskDao, TaskStatusDao taskStatusDao, UserDao userDao) {

this.projectDao = projectDao;

this.taskDao = taskDao;

this.taskStatusDao = taskStatusDao;

this.userDao = userDao;

}

@Override

public List<Project> getUserProjects(User user, int limit, int startFrom) {

return projectDao.getUsersProjects(user.getUsername(), limit, startFrom);

}

@Override

@PreAuthorize("isAuthenticated() and principal.canCreateProject")

public void createProject(Project project){

User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

project.setCompany(user.getCompany());

projectDao.createProject(project);

}

@Override

@PreAuthorize("@databaseUserProjectDao.isUserAssignOnProject(principal.username, projectId)")

public Project getProject(int projectId){

Project project = projectDao.getProject(projectId);

List<Status> statuses = taskStatusDao.getAvailableProjectStatus(projectId);

List<Task> tasks = new ArrayList<>();

for(Status status : statuses){

List<Task> tasksInStatus = taskDao.getProjectTasks(projectId, status.getId(), TASK\_START\_FROM, TASK\_LIMIT);

tasks.addAll(tasksInStatus);

}

project.setTasks(tasks);

project.setUsers(userDao.getUsersOnProject(projectId, TASK\_START\_FROM, TASK\_LIMIT));

return project;

}

}

Содержимое файла DefaultUserService.java

package by.bsuir.kanban.service.impl;

import by.bsuir.kanban.ImageConstant;

import by.bsuir.kanban.dao.UserDao;

import by.bsuir.kanban.domain.Notification;

import by.bsuir.kanban.domain.User;

import by.bsuir.kanban.service.NotificationSender;

import by.bsuir.kanban.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.io.File;

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.List;

/\*\*

\* Created by ulza1116 on 4/7/2017.

\*/

@Service

public class DefaultUserService implements UserDetailsService, UserService {

private final UserDao userDao;

private final PasswordEncoder passwordEncoder;

private final NotificationSender notificationSender;

@Autowired

public DefaultUserService(UserDao userDao, PasswordEncoder passwordEncoder, NotificationSender notificationSender) {

this.notificationSender = notificationSender;

this.userDao = userDao;

this.passwordEncoder = passwordEncoder;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userDao.getUserByUsername(username);

if(user == null){

throw new UsernameNotFoundException("User with username " + username + " was not found");

}

return user;

}

@Override

@Transactional

public void createUser(User user){

String password = getRandomPassword();

user.setPassword(passwordEncoder.encode(password));

userDao.createUser(user);

Notification notification = new Notification();

notification.setBody(user.getUsername() + " " + password);

notification.setFrom("vladislav.zavadski@gmail.com");

notification.setSubject("Registration");

notification.setTo(user.getEmail());

notificationSender.sendNotification(notification);

}

@Override

@Transactional

@PreAuthorize("isAuthenticated()")

public void updateUser(User user){

User currentUser = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

User tempUser = new User(user);

tempUser.setUsername(currentUser.getUsername());

if(tempUser.getPassword() == null){

tempUser.setPassword(currentUser.getPassword());

}

else {

tempUser.setPassword(passwordEncoder.encode(tempUser.getPassword()));

}

userDao.updateUser(tempUser);

}

@Override

@PreAuthorize("isAuthenticated()")

public File getUserPicture(String username, String imageDirectoryFolder){

String pathToImage = userDao.getImagePath(username);

if(pathToImage == null){

return new File(imageDirectoryFolder + ImageConstant.DEFAULT\_USER\_IMAGE);

}

File file = new File(pathToImage);

if(file.exists()){

return file;

}

else {

return new File(imageDirectoryFolder + ImageConstant.DEFAULT\_USER\_IMAGE);

}

}

@Override

@PreAuthorize("isAuthenticated() and (principal.company.id eq #companyId)")

public List<User> getUsersInCompany(int companyId){

return userDao.selectUsersByCompany(companyId);

}

private String getRandomPassword(){

SecureRandom secureRandom = new SecureRandom();

return new BigInteger(64, secureRandom).toString(32);

}

}

# ПРИЛОЖЕНИЕ Б

(справочное)

Программное средство Kanban board. Схема структурная.

# ПРИЛОЖЕНИЕ В

(справочное)

Программное средство Kanban board. Диаграмма использования.