**4.** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

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

**4.1** Модуль аутентификации

Мудуль аутентификации необходим для входа пользователей в систему. Аутентификация производится по логину и паролю, введенными пользователем. Так же данный модуль занимается управлением пользователями системы: регистрация, обновление профиля, удаление пользователя.

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* является местом для настройки процесса аутентификации. Класс унаследован от класса *WebSecurityConfigurerAdapter* и переопределяет метод *configure,* который принимает объект *HttpSecurity* для настройки процесса аутентификации. В листинге 4.1 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)*.*

*http.csrf().*

*Disable().*

*FormLogin().*

*loginPage("/login").*

*usernameParameter("username").*

*passwordParameter("password").*

*failureHandler(authenticationFailureHandler()).*

*successHandler(authenticationSuccessHandler())*

Листинг 4.1 – Настройка аутентификации

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. В листинге 4.2 приведен данный метод.

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

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

}

Листинг 4.2 – Метод *configure.*

Класс *SecurityConfiguration* помечен аннотацией *@Configuration*, тем самым указывается что данный класс является классом для настройки безопасности приложения. На рисунке 4.1 представлена схема реализации механизма аутентификации в Spring Security.



Рисунок 4.1 – 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, посредством сохранения ключа авторизации в cookies пользовательского браузера.

Весь вышеизложенный функционал реализован с использованием Spring Security. Данный фреймворк необходимо включить в зависимости Maven (см. листинг 4.3).

<dependency>

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

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

</dependency>

Листинг 4.3 – подключение Spring Security в Maven

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 4.2.

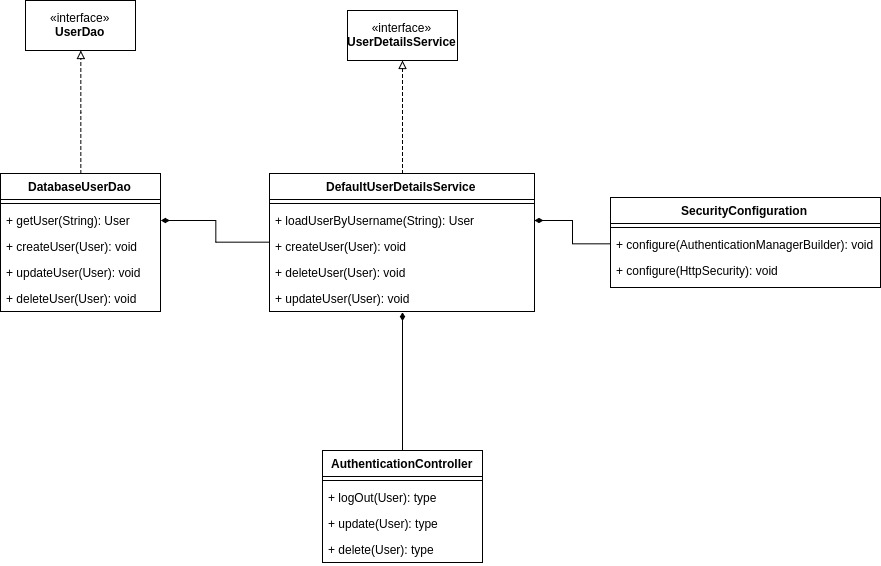


Рисунок 4.2 – Дигарамма классов модуля аутентификации

**4.2** Модуль отправки уведомлений

Данный модуль предназначен для отправки уведомлений ползователям по электронной почте. Непосредственным отправителем письма будет являтся класс *MailSender*, с единственным методом *send,* который принимает три параметра: электронный адрес получателя, тему и тело сообщения. Так же в отдельный файл должны быть вынесены следующие свойства: smtp-хост, номер порта, электронныйй адрес и пароль отправителя сообщений.

Данный модуль реализуется с помощью библиотеки JavaMail, которая подключается с помощью добавления зависимости в файл *pom.xml* (см. листинг 4.4).

<dependency>

     <groupId>javax.mail</groupId>

     <artifactId>mail</artifactId>

     <version>1.4</version>

</dependency>

Листинг 4.4 – Подключение JavaMail

JavaMail позволяет производить отправку сообщений по следующим протоколам: SMTP, POP, IMAP, MIME.



Рисунок 4.3 – Схема работы библиоткеки JavaMail

На рисунке 4.3 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 4.4 представлена диаграмма классов данного модуля.

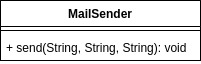
****

Рисунок 4.4 – Диаграмма классов модуля отправки уведомлений

**4.3** 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*, таким образом метод возвращает список объектов класса *Task* с определенным 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 классах *DefaultReportService* (класс уровня бизнес логики) и *ReportDao* (класс уровня доступа к данным). На рисунке 4.5 представлена диаграмма классов модуля построения отчетов.

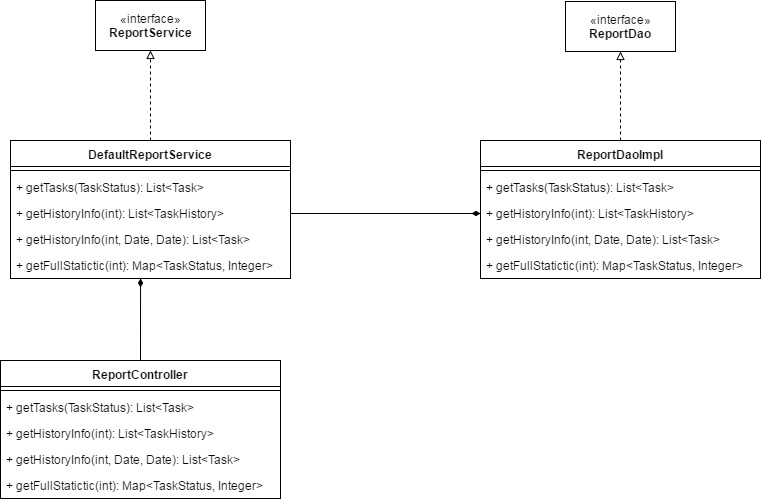


Рисунок 4.5 – Модель классов модуля построения отчетов

Отдельно выделяется класс *TimeTrackController.* Этот класс предназначен для отчета пользователя о проделанной работе. Метод *logWork* принимает экземпляр класса *Work* 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.

Методы с аналогичными названиями присутствуют в классах *DefaultLogWorkService* и *LogWorkDao.*

**4.4** 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: *DefaultUserDao*, *DefaultTaskDao*, *DefaultProjectDao*, *DefaultPermissionDao*, *DefaultTaskHistoryDao*, *DefaultLogWorkDao*, *DefaultTaskStatusDao*, *DefaultGroupDao.*

Класс *DefaultUserDao* 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). На рисунке 4.6 представлена диаграмма *DefaultUserDao*.

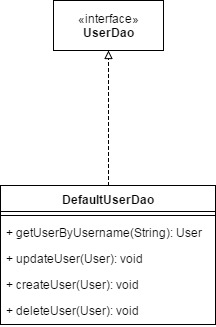


Рисунок 4.6 – Диаграмма классов DefaultUserDao

Класс *DefaultTaskDao* 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). На рисунке 4.7 представлена диаграмма для *DefaultTaskDao*.

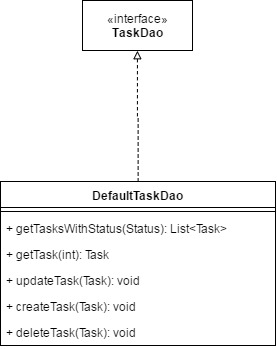


Рисунок 4.7 – Диаграмма классов DefaultTaskDao

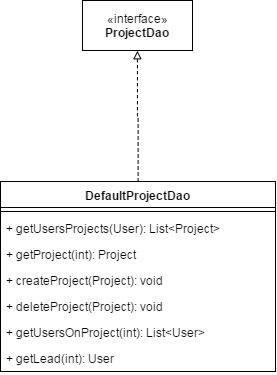


Рисунок 4.9 – Диаграмма классов DefaultProjectDao

Класс *DefaultProjectDao* 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). На рисунке 4.9 представлена диаграмма для *DefaultProjectDao.*

Класс *DefaultPermissionDao* 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). На рисунке 4.10 представлена диаграмма для *DefaultPermissionDao*.

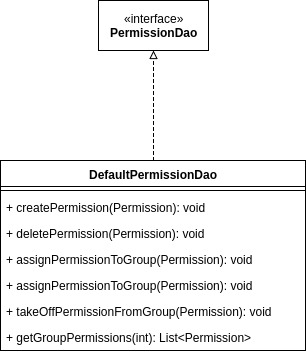


Рисунок 4.10 – Диаграмма классов DefaultPermissionDao

Класс *DefaultTaskHistoryDao* 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 измененений задач). На рисунке 4.8 представлена диаграмма *DefaultTaskHistoryDao.*

Класс *DefaultLogWorkDao* 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). На рисунке 4.11 представлена диаграмма для класса *DefaultLogWorkDao.*

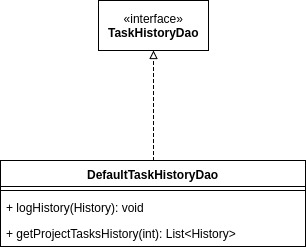
**

Рисунок 4.11 – Диаграмма классов DefaultTaskHistoryDao

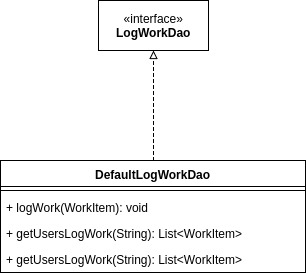


Рисунок 4.12 – Диаграмма классов DefaultLogWorkDao

Класс *DefaultTaskStatusDao* 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). На рисунке 4.13 представлена диаграмма *DefaultTaskStatusDao*.

Класс *DefaultGroupDao* 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 4.14 представлена диаграмма *DefaultGroupDao.*

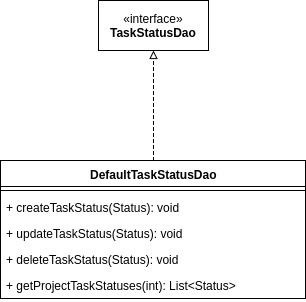


Рисунок 4.13 – Диаграмма классов DefaultTaskStatusDao

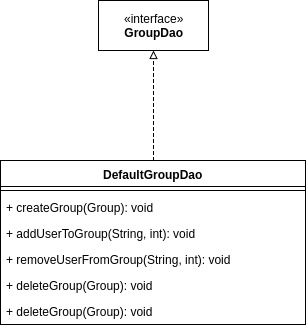


Рисунок 4.14 – Диаграмма классов DefaultGroupDao

**4.5** 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.* Диаграмма классов разрабатываемого модуля представлена на рисунке 4.15.

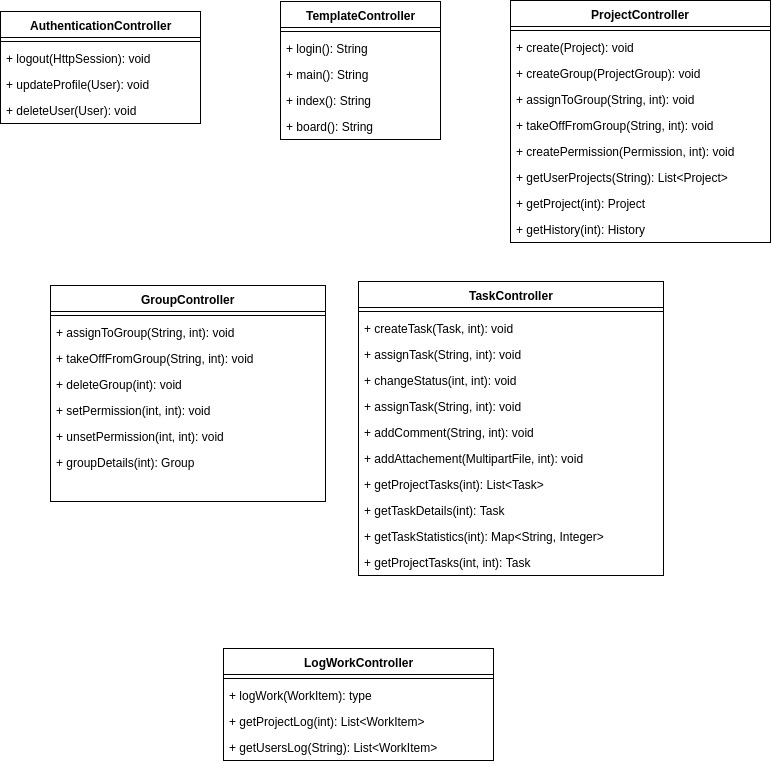


Рисунок 4.15 – Диаграмма классов модуля API веб-сервиса

Класс *AuthenticationController* предоставляет интерфейс по управлению пользователями. Класс реализует следующие методы: *logout* (метод принимает текущуб пользовательскую сессию и закрывает ее, 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 *login* возвращает название шаблона со страницей входа в систему. Метод *main* возвращает название шаблона с главной страницей приложения. Метод *index* возвращает возвращает название шаблона главной страницы приложения. Метод *board* 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.

**4.6** 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.