**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

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

## 3.1 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*, *DatabaseIssueDao*, *DatabaseCommentDao*, *DatabaseMessageDao*, *DatabaseIssueLifecycleDao*, DatabaseIssueStatusDao, *DatabaseUserRoleDao.*

Класс *DatabaseUserDao* (см. рис. 3.1)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 *DatabaseUserRoleDao* (см. рис. 3.1)предоставляет интерфейс для доступа к данным, которые касаются ролей групп. Класс реализует следующие методы: *createUserRole* (метод принимает экземпляр класса *Role* и создает новую роль пользователей)*, addUserToRole* (метод принимает идентификатор роли и имя пользователя и производит добавление пользователя к данной группы)*, removeUserFromRole* (метод принимет идентификатор роли и имя пользователя и удаляет пользователя из данной роли)*, deleteUserRole* (метод принимает экземпляр класса *Role* и удаляет группу пользователей), *getUsersRole* (метод принимает идентификатор проекта и возвращает все группы на проекте

Класс *DatabaseIssueDao* (см. рис. 3.2)является точкой доступа к данным касающихся задачи. Данный класс реализует следующие методы: *getIssuesWithPriority* (метод принимает экземпляр класса *Priority* и возвращает список с указанным статусом), *getIssue* (метод принимает идентификатор задачи и возвращает экземпляр класса *Issue*), *updateIssue* (метод принимает экземпляр класса *Issue* и обновляет соответствующее поля в базе данных), *deleteIssue* (метод принимает экземпляр класса *Issue* и удаляет соответствующую задачу из базы данных), *addIssueExecutor* (метод принимает экземпляр класса *User* тем самым прикрепляется исполнитель к задаче), *addComment* (метод принимает экземпляр класса *Comment,* таким образом добавляется комментарий к задаче), *getIssuebyUsers* (метод принимает идентификатор пользователя, и возвращает список его задач), *getUsersbyIssue* (метод принимает идентификатор задачи и возвращает список пользователей, участвующих в данной задаче).

Класс *DatabaseMessageDao* (см. рис. 3.1)представляет собой точку для доступа к данным касающихся cообщений. Данный класс реализует следующие методы методы: *getListMessages* (метод принимает идентификатор пользователяи возвращает список сообщений, с которыми имеет дело данный пользователь), *getMessage* (метод принимает идентификатор сообщения и возвращает экземпляр класса *Message*), *createMessage* (метод принимает экземпляр класса *Message* и создает данное сообщение в базе данных), *deleteMessage* (метод принимает экземпляр класса *Message* и удаляет сообщение из базы данных), *getMessageIssue* (метод принимает идентификатор задачи и возвращает список сообщений, связанных с данной задачей), *getMessageUsers* (метод принимает идентификаторы двух пользователей и возвращает список сообщений между этими пользователями).

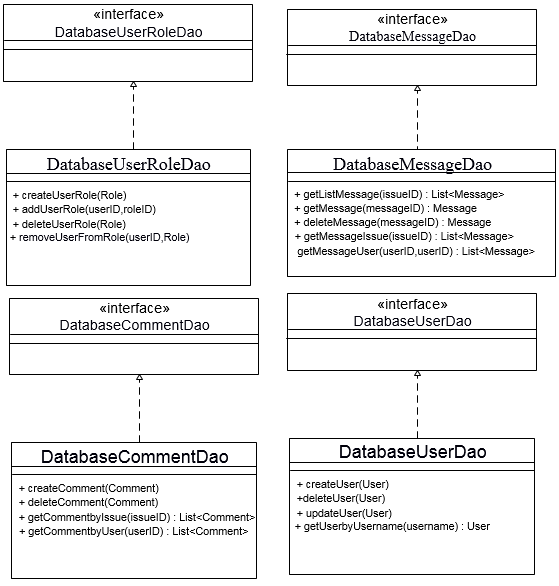


Рисунок 3.1 – Диаграмма классов для DatabaseMessageDao, DatabaseCommentDao, DatabaseUserRoleDao, DatabaseUserDao

Класс *DatabaseIssueStatusDao* (см. рис. 3.2)предоставляет интерфейс, который позволяет манипулировать данными, которые касаются статуса задачи. Данный класс реализует следующие методы: *createIssueStatus* (метод принимает экземпляр класса *Status* и создает новый статус задачи)*, updateIssueStatus* (метод принимает экземпляр класса *User* и обновляет информацию о статусе задачи)*, deleteIssueStatus* (метод принимет экземпляр класса *Status* и удаляет статус задачи).

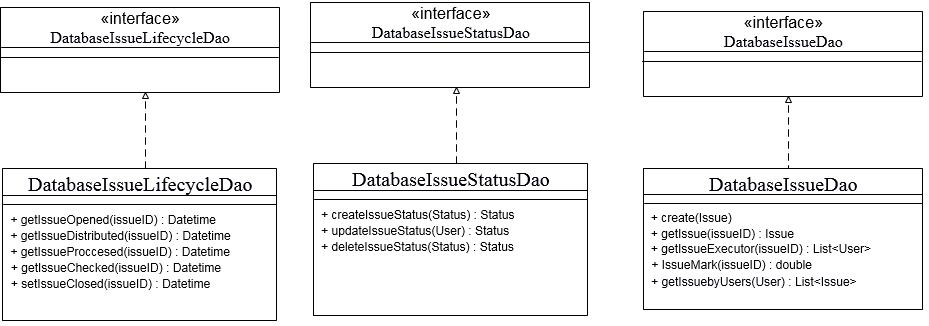


Рисунок 3.2 – Диаграмма классов DatabaseIssueDao, DatabaseIssueStatusDao, DatabaseIssueLifecycleDao

Класс *DatabaseCommentDao* (см. рис. 3.1)предоставляет интерфейс по управлению комментариями над задачами. Класс реализует следующие методы: *createСomment* (метод принимает экземпляр класса *Comment* и создает новый комментарий в базе данных), *deleteComment* (метод принимает экземпляр класса *Comment* и удаляет данный комментарий из базы данных), getCommentsbyIssue (метод принимает экземпляр класса *Issue* и предоставляет список комментариев, участвующих в данной задаче), *getCommentsbyUser* (метод принимает экземпляр класса *User* и предоставляет список комментариев, написанных данным пользователем).

Класс *DatabaseIssueLifecycleDao* (см. рис. 3.2)предоставляет интерфейс для доступа к данным, касающихся жизненного цикла задачи. Данный класс реализует следующие методы: *getIssueOpened* (метод принимает идентификатор задачи и возвращает дату создания текущей задачи), *getIssueProccesed* (метод принимает идентификатор задачи и возвращает дату начала работы над текущей задачей), *getIssueChecked* (метод принимает идентификатор задачи и возвращает дату проверки текущей задачи), *getIssueClosed* (метод принимает идентификатор задачи и возвращает дату закрытия текущей задачи).

**3.2** Модуль прямой и обратной связи

Данный модуль предназначен для отправки уведомлений пользователям по электронной почте. Класс *FeedbackService* 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), *sendAddedIssueMessage* (метод принимает объекты классов *Issue* и *User* и отправляет пользователю уведомление, о том, что он был добавил новую задачу).

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.

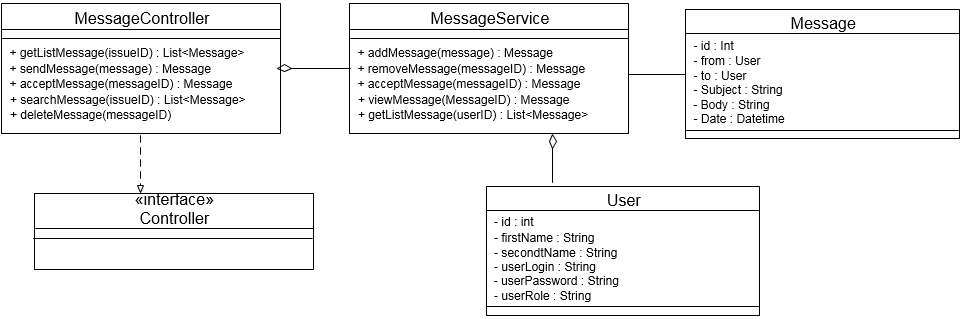


Рисунок 3.4 – Диаграмма классов модуля отправки уведомлений

**3.3** Модуль аутентификации

Модуль аутентификации включает в себя функцию входа пользователей в систему, для работы с продуктом. Для того, чтобы войти в систему пользователю необходимо ввести верный логин и пароль. Модуль аутентификации также необходим для управления пользователями, которые хотят войти в систему, и влючает следующие функции: удаление и обновление профиля, регистрация пользователей. Данный модуль реализуется с использованием модуля Spring Security фреймворка Spring. Для реализации модуля необходимо всего три класса реализующих логику: *UserDatabaseService*, *SecurityConfiguration, DatabaseUserDao*. И один класс сущности: *User*.

Класс *UserDatabaseService*, является реализацией интерфейса *UserDetailsService* в модуле Spring Security и занимается предоставлением метода *loadUserByUsername,* на вход которого принимается имя пользователя и в случае успешной аутентификации возвращается объект *User* или в случае,если пользователя нет не базе данных, то выбрасывает исключение. Данный метод в свою очередь обращается к методу *getUserByUsername* класса *DatabaseUserDao*, который возвращает объект *User*, если пользователь найден, или *null* в обратном случае.

Класс *SecurityConfiguration* введен для того, чтобы можно было настраивать процесс аутентификации, также данный класс является местом для настройки процесса аутентификации. Данный класс является наследником 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,* в последствии чего данный класс будет использован при аутентификации. 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 мы можем понять,что класс *SecurityConfiguration* используется для обеспечения безопасности приложения.

Для того, что пользователь не авторизовывался каждый раз при входе в приложение, что доставляет некоторое неудобство, в этом модуле есть функция рода «remember me». Чтобы этого достичь ключ авторизации сохраняется в cookies пользовательского браузера.

Вышеуказанный функционал данного модуля реализован с использованием фреймворка Spring Security. Для работы с данным фреймворком, Spring Security необходимо включить в зависимости Apache Maven. Ниже приведен листинг для подключения Spring Security в Maven.

<dependency>

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

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

</dependency>

На рисунке 3.5 представлена схема реализации механизма аутентификации в Spring Security.

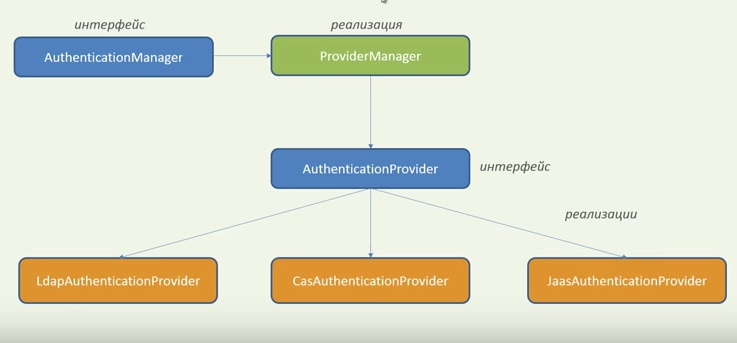


Рисунок 3.5 – Схема работы аутентификации Spring Security

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*, *UserDatabaseService* и *DatabaseUserDao*.

Класс *UserDatabaseService* 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 *update* принимает объект класса *User*, проверяет правильность данных и передает его в метод *updateUser* класса *UserDatabaseService.* 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 3.6.

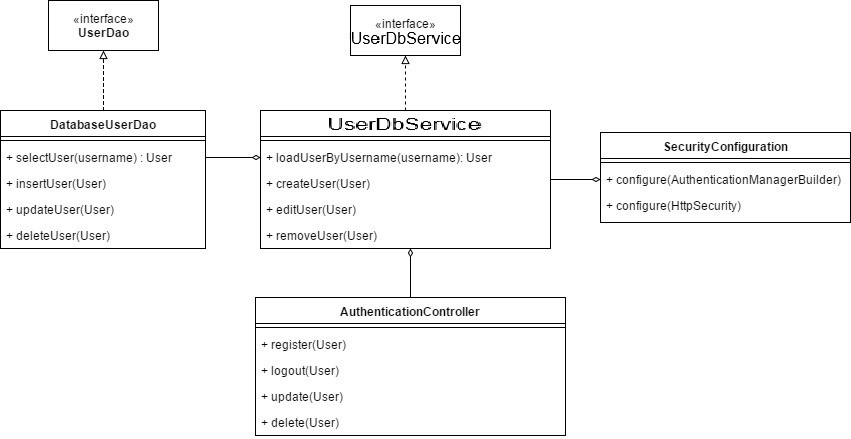


Рисунок 3.6 – Диаграмма классов модуля аутентификации

## 3.4 Модуль авторизированной системы учёта заявок от клиентов

Данный модуль предназначен для размещения проблем и предложений в удобном для пользователей виде. Для этого также используется модуль ранжирования проблем и предложений, который сортирует заявки в необходимом порядке.

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

Метод класса *HelpdeskController getIssue* принимает идентификатор задачи и возвращает объект класса *Issue*. Метод *getComment* принимает идентификатор задачи возвращает список объектов класса *Comment.* В конечном итоге пользователь сможет увидеть все комментарии над интересующей его проблемой. Метод *removeIssue* принимает идентификатор задачи и удаляет данную задачу*.* Метод *removeСomment* принимает идентификатор комментарий и удаляет данный комментарий*.*

Методы с аналогичными названиями присутствуют в классе *HelpdeskService.*

На рисунке 3.7 представлена диаграмма классов модуля авторизированной системы учёта заявок от клиентов.

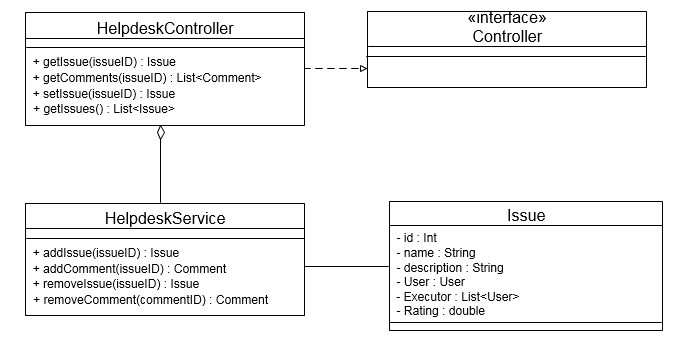


Рисунок 3.7 – Диаграмма классов модуля авторизированной системы учёта заявок от клиентов

Данный модуль связан с модулем для работы над задачами, функционал которого создан для создания задач, смены статуса и приоритета задачи и т. д. Данный модуль представлен в виде следующих классов: *IssueController, IssueService, DatabaseIssueDao, CommentController, CommentService, DatabaseCommentDao.*

Класс *IssueController* является контроллером, и при вызове его методов обработка запроса сразу же пересылается методу с таким же названием из класса *IssueService*.

Класс *IssueService* предоставляет интерфейс по управлению задачами на проекте. Метод *createIssue* принимает объект класса *Issue*, тем самым создавая новую задачу. Метод *getIssue* принимает идентификатор задачи и возвращает объект класса *Issue*. Метод *getIssueExecutor* принимает идентификатор задачи и возвращает список пользователей, которые работают над данной задачей. Метод *getIssueMark* принимает идентификатор задачи и возвращает оценку данной задачи. Метод *getPriority* принимает идентификатор задачи и возращает приоритет задачи. Метод *getIssueComments* принимает идентификатор задачи и возвращает список комментариев над данной задачей.

Класс *DatabaseIssueDao* является точкой доступа к данным касающихся задачи. Данный класс реализует следующие методы: *getIssuesWithPriority* (метод принимает экземпляр класса *Priority* и возвращает список с указанным статусом), *getIssue* (метод принимает идентификатор задачи и возвращает экземпляр класса *Issue*), *updateIssue* (метод принимает экземпляр класса *Issue* и обновляет соответствующее поля в базе данных), *deleteIssue* (метод принимает экземпляр класса *Issue* и удаляет соответствующую задачу из базы данных), *addIssueExecutor* (метод принимает экземпляр класса *User* тем самым прикрепляется исполнитель к задаче), *addComment* (метод принимает экземпляр класса *Comment,* таким образом добавляется комментарий к задаче), *getIssuebyUsers* (метод принимает идентификатор пользователя, и возвращает список его задач), *getUsersbyIssue* (метод принимает идентификатор задачи и возвращает список пользователей, участвующих в данной задаче).Класс *CommentController* является контроллером, и при вызове его методов обработка запроса сразу же пересылается методу с таким же названием из класса *IssueService*.

Класс *CommentService* предоставляет интерфейс по управлению задачами на проекте. Метод *createComment* принимает объект класса *Comment*, тем самым создавая новый комментарий. Метод *getComment* принимает идентификатор комментария и возвращает объект класса *Comment*. Метод *getCommentUser* принимает идентификатор комментария и возвращает объект класса *User*.

Класс *DatabaseCommentDao* предоставляет интерфейс по управлению комментариями над задачами. Класс реализует следующие методы: *createСomment* (метод принимает экземпляр класса *Comment* и создает новый комментарий в базе данных), *deleteComment* (метод принимает экземпляр класса *Comment* и удаляет данный комментарий из базы данных), getCommentsbyIssue (метод принимает экземпляр класса *Issue* и предоставляет список комментариев, участвующих в данной задаче), *getCommentsbyUser* (метод принимает экземпляр класса *User* и предоставляет список комментариев, написанных данным пользователем).

## 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, HelpdeskController, AdminController, IssueController,*

*PrivateMessageController* (см. рис. 3.8)*.*

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).

Для реализации функционала необходимо выделить класс *HelpdeskController*. Все методы класса возвращают задачи, комментарии или список задач, выбранных по тому или иному критерию. Метод класса *HelpdeskController getIssue* принимает идентификатор задачи и возвращает объект класса *Issue*. Метод *getComment* принимает идентификатор задачи возвращает список объектов класса *Comment.* В конечном итоге пользователь сможет увидеть все комментарии над интересующей его проблемой. Метод *removeIssue* принимает идентификатор задачи и удаляет данную задачу*.* Метод *removeСomment* принимает идентификатор комментарий и удаляет данный комментарий*.*

Класс *AdminController* предоставляет интерфейс для администрирования. Метод *addBlockedUser* принимает идентификатор пользователя и блокирует пользователя. Метод *addPerson* принимает идентификатор пользователя и роли пользователя и добавляет пользователя в систему. Метод *getListBlockedUser* возвращает список заблокированных пользователей. Метод *getUsers* возвращает список пользователей.

Класс *IssueController* предоставляет интерфейс по управлению задачами. Метод *createIssue* принимает объект класса Issue, тем самым создавая новую задачу. Метод *getIssue* принимает идентификатор задачи и возвращает объект класса Issue. Метод *getIssueExecutor* принимает идентификатор задачи и возвращает список пользователей, которые работают над данной задачей. Метод *getIssueMark* принимает идентификатор задачи и возвращает оценку данной задачи. Метод *getPriority* принимает идентификатор задачи и возращает приоритет задачи. Метод *getIssueComments* принимает идентификатор задачи и возвращает список комментариев над данной задачей.

Класс *PrivateMessageController* предоставляет интерфейс по управлению личных сообщений пользователей. Метод *getMessage* принимает идентификатор задачи и возвращает список сообщений по данной задаче. Метод *sendMessage* принимает экземпляр класса *Message* и отправляет сообщение. Метод *acceptMessage* принимает экземпляр класса *Message* и принимает сообщение. Метод *deleteMessage* принимает идентификатор сообщения и удаляет данное сообщение. Диаграмму классов, отражающую данный модуль можно увидеть на рисунке 3.8.

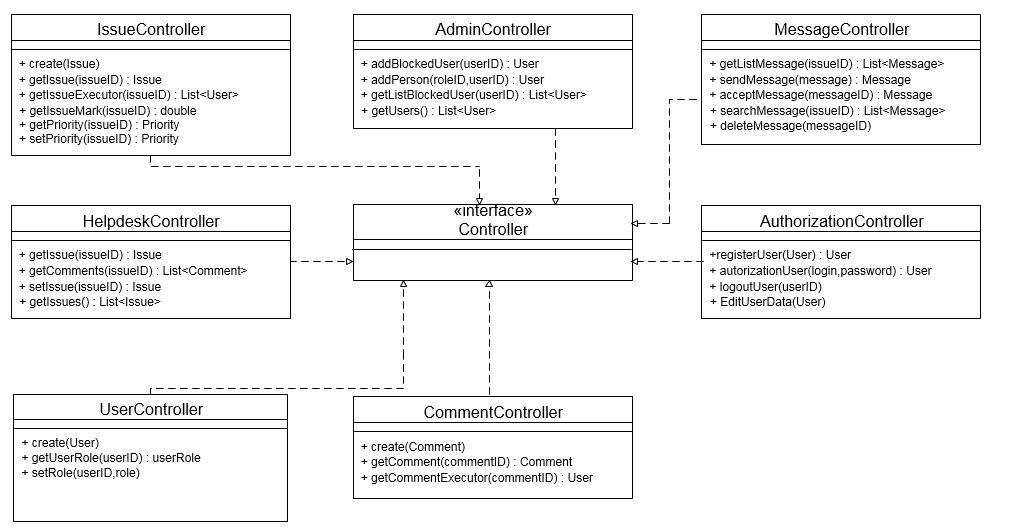


Рисунок 3.8 – Диаграмма классов модуля API веб-сервиса

## 3.6 Модуль маршрутизации

Данный модуль реализован с помощью фреймворка Spring и является своего рода точкой входа в веб-сервис. Модуль существует в виде сервлета, который принимает http-запрос и направляет данный запрос 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 ранжирования проблем и предложений

Данный модуль предназначен для ранжирования проблем и предложений, для более эффективной работы сотрудников над приложением и наименьших затрат времени на устранение наиболее актуальных проблем.

Данный модуль связан с модулем для работы над задачами, функционал которого создан для создания задач, смены статуса и приоритета задачи и т. д. Данный модуль представлен в виде следующих классов: *IssueController, IssueService, DatabaseIssueDao.*.

Класс *IssueController* является является контроллером, и при вызове его методов обработка запроса сразу же пересылается методу с таким же названием из класса *IssueService*.

Класс *IssueService* предоставляет интерфейс по управлению задачами. Метод *createIssue* принимает объект класса Issue, тем самым создавая новую задачу. Метод *getIssue* принимает идентификатор задачи и возвращает объект класса Issue. Метод *getIssueExecutor* принимает идентификатор задачи и возвращает список пользователей, которые работают над данной задачей. Метод *getIssueMark* принимает идентификатор задачи и возвращает оценку данной задачи. Метод *getPriority* принимает идентификатор задачи и возращает приоритет задачи. Метод *getIssueComments* принимает идентификатор задачи и возвращает список комментариев над данной задачей.

Класс *DatabaseIssueDao* является точкой доступа к данным касающихся задачи. Данный класс реализует следующие методы: *getIssuesWithPriority* (метод принимает экземпляр класса *Priority* и возвращает список с указанным статусом), *getIssue* (метод принимает идентификатор задачи и возвращает экземпляр класса *Issue*), *updateIssue* (метод принимает экземпляр класса *Issue* и обновляет соответствующее поля в базе данных), *deleteIssue* (метод принимает экземпляр класса *Issue* и удаляет соответствующую задачу из базы данных), *addIssueExecutor* (метод принимает экземпляр класса *User* тем самым прикрепляется исполнитель к задаче), *addComment* (метод принимает экземпляр класса *Comment,* таким образом добавляется комментарий к задаче), *getIssuebyUsers* (метод принимает идентификатор пользователя, и возвращает список его задач), *getUsersbyIssue* (метод принимает идентификатор задачи и возвращает список пользователей, участвующих в данной задаче).

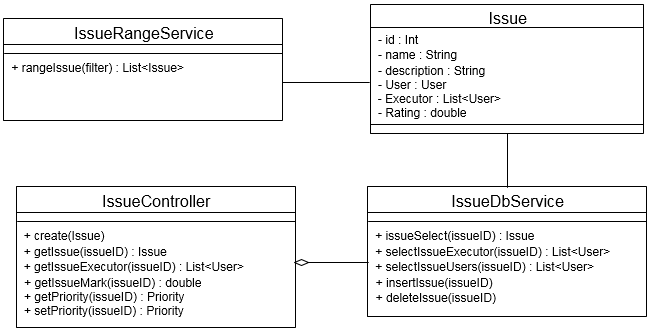


Рисунок 3.9 – Диаграмма классов модуля ранжирования проблем и предложений

## 3.8 Модуль администрирования и авторизации

В данном модуле сосредоточены функции получения прав пользователя, назначения прав пользователям, а также для администрирования системы в целом. Авторизация производится с использованием модуля Spring Security фреймворка Spring.

Администрирование системы представлено в классе *AdminController.* Класс *AdminController* предоставляет интерфейс для администрирования.Метод *addBlockedUser* принимает идентификатор пользователя и блокирует пользователя. Метод *addPerson* принимает идентификатор пользователя и роли пользователя и добавляет пользователя в систему. Метод *getListBlockedUser* возвращает список заблокированных пользователей. Метод *getUsers* возвращает список пользователей.

Авторизация осуществляется фреймворком Spring Security. Для его настройки необходимо унаследоваться от класса *WebSecurityConfigurerAdapter* и переопределить метод *configure.* 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 аннотацию *@PreAuthorize* и в ней в атрибуте *value* написать выражение на языке *SpEL,* которое проверяет, может данный пользователь вызывать этот метод или нет. В случае, если выражение вернуло *false* клиенту придет ответ с кодом 403 (Доступ запрещен), иначе метод выполнится и, в случае необходимости, будет возвращен результат его выполнения.

## 3.9 Модуль взаимодействия с веб-сервисом

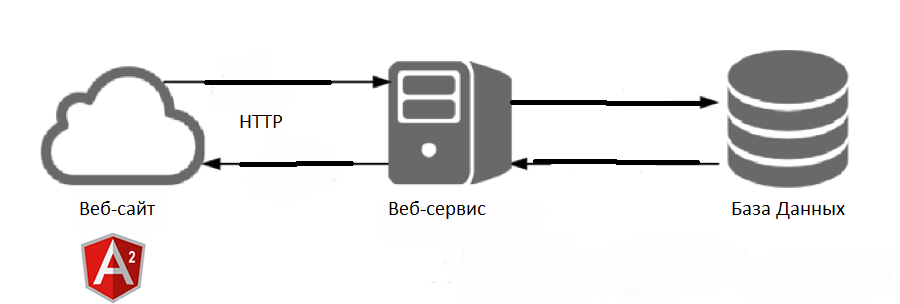
Модуль взаимодействия с веб-сервисом расположен на стороне веб-сайта и посылает запросы веб-сервису по протоколу http или https. Таким образом веб-сайт и веб-сервис обмениваются данными в формате JSON или XML. На рисунке 3.10 представлена схема взаимодействия веб-сайта и веб-сервиса.

Рисунок 3.10 – Схема взаимодействия в веб-сервисом

В Angular взаимодействие с веб-сервисом происходит с помощью 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 визуализации

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. 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. На рисунке 3.11 представлена схема рендеринга html-страниц в фреймворке Angular.

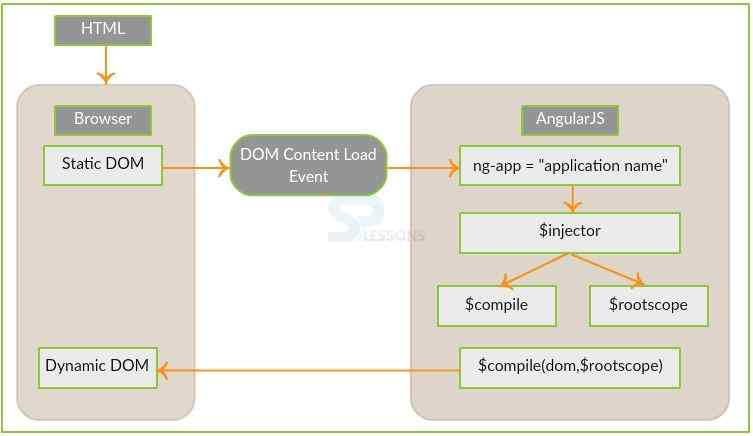


Рисунок 3.11 – Рендеринг страниц в Angular

## 3.11 Модуль пользовательского интерфейса

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). Ниже приведен листинг использования *ngRoute*.

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

Для данного программного продукта, который разрабатывается в ходе дипломного проекта, одним из самых важных процессов является процесс создания задачи. На рисунке 4.1 представлен алгоритм работы данного процесса.

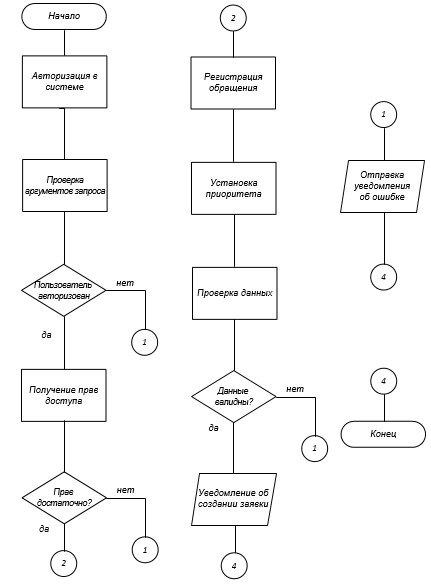


Рисунок 4.1 – Алгорим создания заявки

После обращения в службу поддержки приложения проверяется в текущей сессии авторизован ли в системе пользователь. Если пользователь не авторизован в системе, то клиенту отправляется сообщение об ошибке, в котором указана причина ошибки авторизации. При успешной проверке авторизации пользователя системы ему приходит успешный результат проверки авторизации.

После того, как пользователь отправил запрос, для того, чтобы создать задачу, происходит процесс проверки прав пользователя. Он обращается к сервису *PermissionService*, для того чтобы получить права пользователя. В случае успешного получения прав пользователя происходит регистрация обращения в *RequestService.*

После получения запроса на создание задачи производится установка приоритета задачи. После установки приоритета задачи производится валидация задачи: у задачи должно присутствовать название, и его длина должна быть не больше восьмидесяти символов, у задачи, должен присутствовать приоритет. Если валидация не была пройдена, на сторону клиента отправляется сообщение о некорректном запросе.

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 Обработка запроса

Для того, чтобы пользователь мог добавить задачу, а затем сотрудники компании смогли ее обработать, пользователю в начале необходимо авторизоваться.

Сначала пользователь обращается в службу поддержки приложения с помощью *RequestController*. После обращения в службу поддержки приложения проверяется в текущей сессии авторизован ли в системе пользователь. Если пользователь не авторизован в системе, то клиенту отправляется сообщение об ошибке, в котором указана причина ошибки авторизации. При успешной проверке авторизации пользователя системы ему приходит успешный результат проверки авторизации.

После того, как пользователь отправил запрос, для того, чтобы создать задачу, происходит процесс проверки прав пользователя. Он обращается к сервису *PermissionService*, для того чтобы получить права пользователя. В случае успешного получения прав пользователя происходит регистрация обращения в *RequestService.*

После получения запроса на создание задачи производится установка приоритета задачи. После установки приоритета задачи производится валидация задачи: у задачи должно присутствовать название, и его длина должна быть не больше восьмидесяти символов, у задачи, должен присутствовать приоритет. Если валидация не была пройдена, на сторону клиента отправляется сообщение о некорректном запросе.

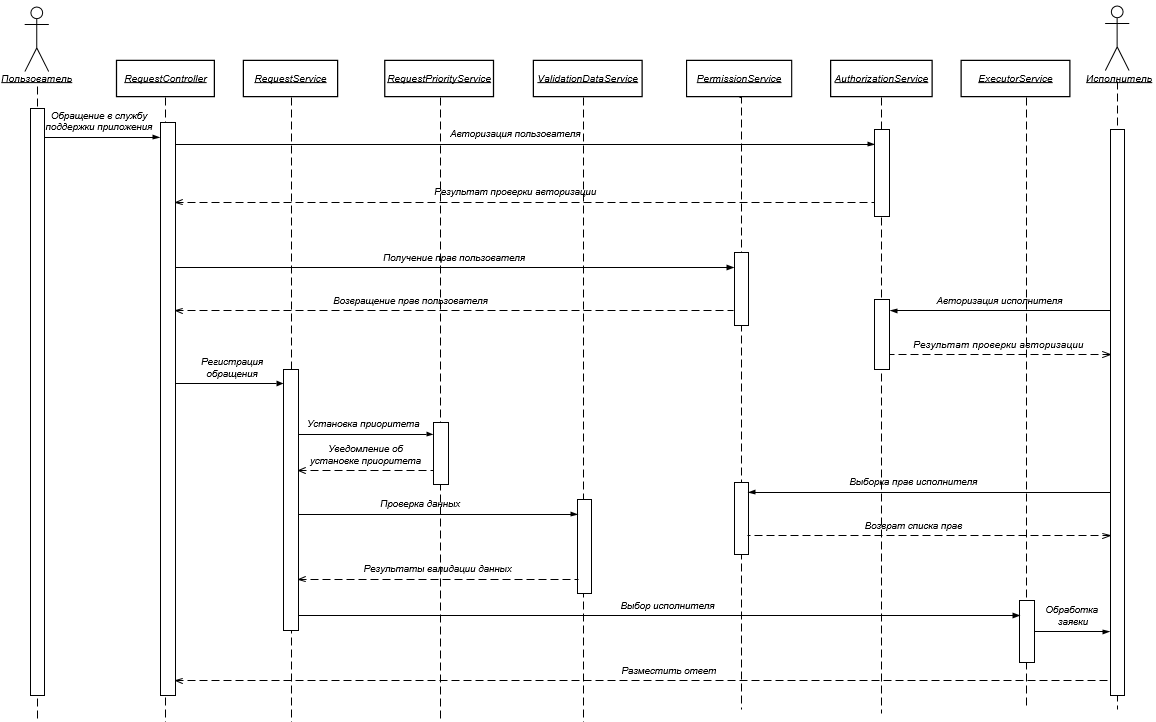


Рисунок 4.2 — Диаграмма последовательности обработки запроса

## Затем происходит обращение в сервис исполнителя *ExecutorService*, где происходит выборка исполнителя для обработки запроса. После обработки запроса, исполнитель обращается к *RequestController*, чтобы разместить ответ.

## 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<Issue> issueRowMapper = (resultSet, i)->{

Issue issue = new Issue (); issue.setId(resultSet.getInt(TableColumn.ISSUE\_ID)); issue.setName(resultSet.getString(TableColumn. ISSUE \_NAME));

issue.setDescription(resultSet.getString(TableColumn. ISSUE \_DESCRIPTION));

issue.setTotalCommentCount(resultSet.getInt(TableColumn.COMMENT\_COUNT));

User user = new User();

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

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

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

issue.setUser(user);

return issue;

};

В листинге выше осуществляется извлечение информации о задаче с создателем данной задачи 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 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 4.3 представлена диаграмма активности для процесса поиска задач.

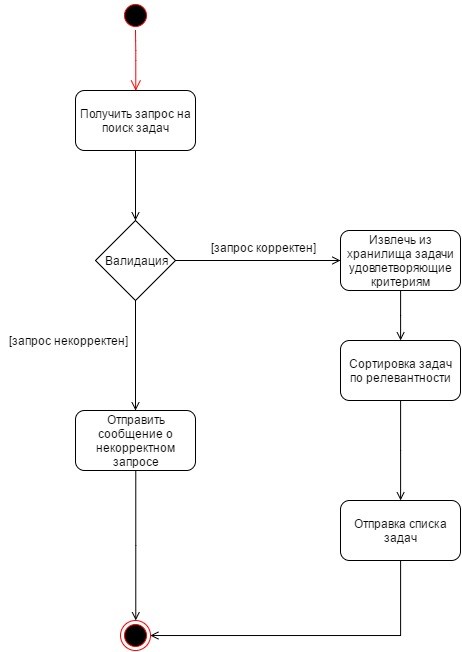


Рисунок 4.3 – Диаграмма активности процесса поиска задач

После получения запроса на поиск задач необходимо совершить валидацию данного запроса: строка запроса не должна быть пустой. Если результат валидации был отрицательным, то пользователю отравляется уведомление о некорректно введенном запросе.

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

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

@Override  
public List<Issue> search Issue (Set<Tag> tags, String issueName) {  
 List<Issue> issues = issue Dao.searchIssue (tags, issueName);  
  
 for(Issue issue: issues){  
 issue.setTags(issueDao.getIssueTags(issue.getId()));  
 }  
  
 Collections.sort(issues, new RelevantIssueComparator(tags));  
  
 return issues;  
}

## 4.5 Изменение прав доступа

Изменение прав доступа пользователя является одной из ключевых операций в разрабатываемом программном продукте. Изменение прав доступа может выполнять только администратор. Для этого администратор должен произвести авторизацию в системе. Если пользователь не авторизован в системе, то клиенту отправляется сообщение об ошибке, в котором указана причина ошибки авторизации. При успешной проверке авторизации пользователя системы ему приходит успешный результат проверки авторизации. Когда администратор успешно авторизовался, ему необходимо перейти на часть сайта, которая предназначена для администрирования над данным программным продуктом. После этого администратор производит выбор пользователя для того, чтобы произвести модификацию прав доступа. На рисунке 4.4 изображен алгоритм для данного процесса.

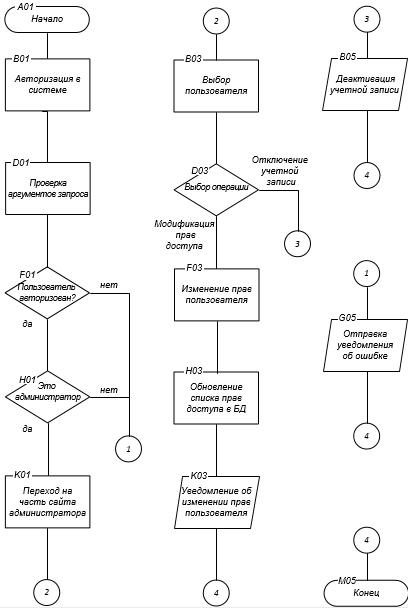


Рисунок 4.4 – Диаграмма активности процесса смены статуса задачи

Далее, после того, как администратор произвел изменение прав доступа, происходит обновление списка прав доступа в базе данных. Затем администратору приходит уведомление об успешном изменении прав пользователя. Ниже приведен фрагмент кода, обновляющий права доступа.

@Override

@Transactional

@PreAuthorize("@permissionDao.canChangeStatus(principal.username, User.userRole, newRole)")

public void changeUserRole(User user, UserRole new UserRole)

{

UserDao.updateRole(user, new UserRole);

}

Аннотация @Transactional говорит о том, что метод changeUserRole выполняется в рамках одной транзакции. Аннотация @PreAuthorize проверяет обладает ли пользователь достаточными правами для изменения статуса задачи. Метод updateRole обновляет права доступа в базе данных.

## 4.6 Закрытие заявки

Закрытие заявки является показателем того, что исполнитель выполнил свою задачу и пользователь удовлетворён ответом

Для того, чтобы пользователь мог добавить задачу, а затем сотрудники компании смогли ее обработать, пользователю в начале необходимо авторизоваться.

Сначала пользователь обращается в службу поддержки приложения с помощью *RequestController*. После обращения в службу поддержки приложения проверяется в текущей сессии авторизован ли в системе пользователь. Если пользователь не авторизован в системе, то клиенту отправляется сообщение об ошибке, в котором указана причина ошибки авторизации. При успешной проверке авторизации пользователя системы ему приходит успешный результат проверки авторизации.

После того, как пользователь отправил запрос, для того, чтобы создать задачу, происходит процесс проверки прав пользователя. Он обращается к сервису *PermissionService*, для того чтобы получить права пользователя. В случае успешного получения прав пользователя происходит регистрация обращения в *RequestService.*

После получения запроса на создание задачи производится установка приоритета задачи. После установки приоритета задачи производится валидация задачи: у задачи должно присутствовать название, и его длина должна быть не больше восьмидесяти символов, у задачи, должен присутствовать приоритет. Если валидация не была пройдена, на сторону клиента отправляется сообщение о некорректном запросе.

## Затем происходит обращение в сервис исполнителя *ExecutorService*, где происходит выборка исполнителя для обработки запроса. После обработки запроса, исполнитель обращается к *RequestController*, чтобы разместить ответ.

После этого, пользователю приходит уведомление, о том, что его заявка обработана. Далее возможны два сценария. Первый сценарий заключается в том, что пользователя не удовлетворил ответ на его вопрос. Тогда он может оставить комментарий, и заяка идёт на доработку. Если пользователь удовлетворён ответом, он сообщает об этом, и заяку можно считать закрытой.

На рисунке 4.5 можно увидеть алгоритм закрытия задачи.

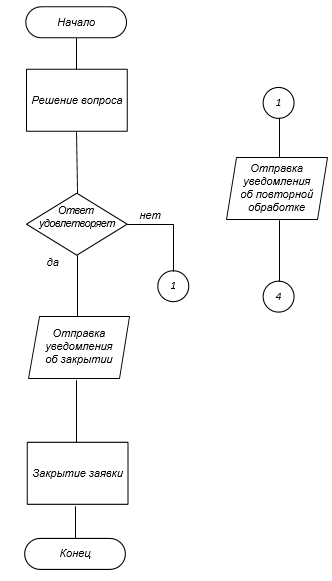


Рисунок 4.5 – Алгоритм закрытия задачи

Ниже приведен фрагмент кода, показывающий, что заяка была закрыта.

Аннотация @Transactional говорит о том, что метод CloseIssue выполняется в рамках одной транзакции. Аннотация @PreAuthorize проверяет обладает ли пользователь достаточными правами для изменения статуса задачи. Метод issueClosed изменяет статус задачи и делает ее закрытой.

@Override

@Transactional

@PreAuthorize("@issueDao.getIssue.issueID == principal.issueid")

public void CloseIssue (String issueID, int statusID){

IssueDao. issueClosed (issueID, CLOSED);

}

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 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 4.6 изображена диаграмма активности для данного процесса.

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(issue.issueStatus, newStatus)")

public void changeIssueStatus(Issue issue, issueStatus newIssueStatus)

{

IssueDao.updateStatus(issue, newIssueStatus);

}

Аннотация @Transactional говорит о том, что метод changeIssueStatus 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.

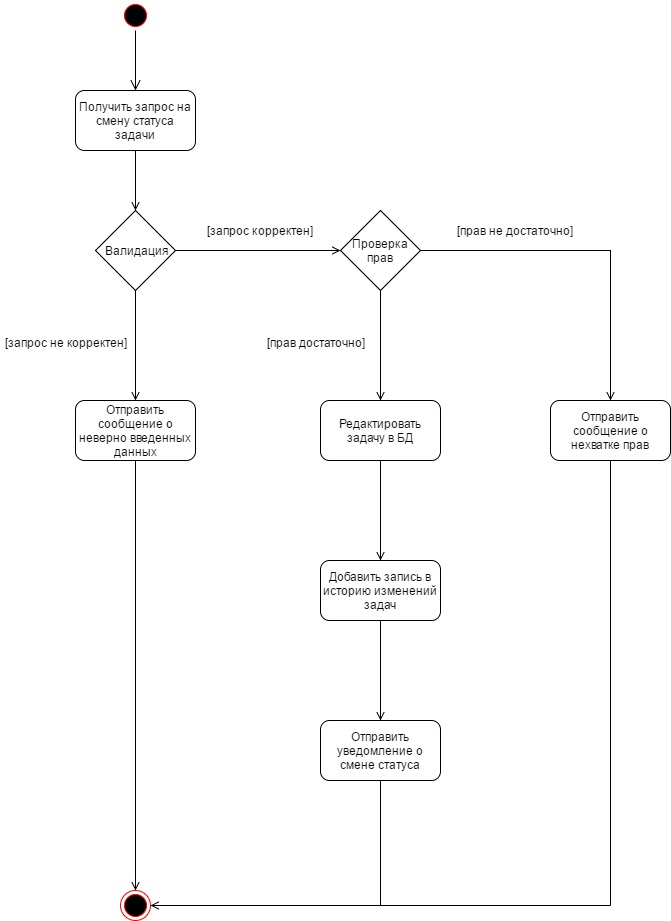


Рисунок 4.6 – Диаграмма активности процесса смены статуса задачи

## 4.8 Блокирование пользователей

Блокировка пользователей необходима для того, чтобы соблюдать порядок в разрабатываемом программном продукте. Блокировать пользователей может выполнять только администратор. Для этого администратор должен произвести авторизацию в системе. Если пользователь не авторизован в системе, то клиенту отправляется сообщение об ошибке, в котором указана причина ошибки авторизации. При успешной проверке авторизации ему приходит успешный результат проверки авторизации. Когда администратор успешно авторизовался, ему необходимо перейти на административную часть сайта. После этого администратор производит выбор пользователя для того, чтобы его заблокировать. На рисунке 4.7 изображен алгоритм для данного процесса.

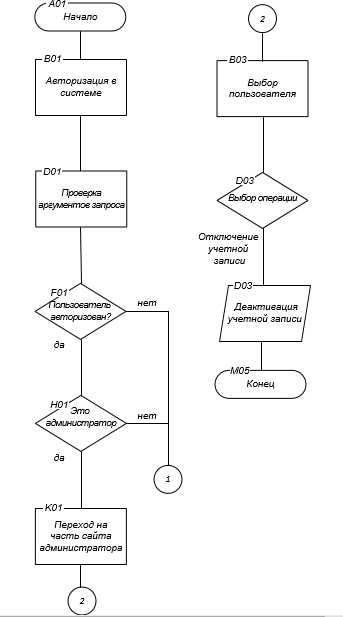


Рисунок 4.7 – Алгоритм блокировки пользователя

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 G780 с установленной ОС Windows, 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 Изменить статус задачи 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 таблицы*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Окно доски заявки | 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 issue»  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 |
| Закрытие задачи | 1 Нажать кнопку «Mark issue as closed»  2 Выбрать уровень удовлетворения оказания услуг  Ожидаемы результат:  Отображение задачи как закрытой, и уровня удовлетворенности | 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>

Ниже рассматриваются тесты, покрывающие основной функционал данного программного продукта.

Для класса IssueController ключевым методом является createIssue. Данный метод принимает экземпляр класса Issue 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 createIssueTestWithEmptyIssue(){

Issue issue = null;

IssueController issueController = new IssueController();

issueController.createIssue(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 createIssueTestWithoutLead(){

Issue issue = new Issue();

issue.setName(“New Issue”);

IssueController issueController = new IssueController();

issue.setLead(null);

issueController.createIssue(issue);

}

Для класса IssueController ключевыми методами являются методы createIssue и changeIssueStatus.

В следующем тесте производится попытка создания новой задачи без указания проекта, которому она принадлежит. Ниже приведен листинг для данного теста.

@Test(expected = InvalidIssueException.class)

public void createIssueTestWithoutIssue(){

Issue issue = new Issue();

issue.setName(“Create new button”);

issue.setDescription(“Description”);

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

issue.setIssue(null);

IssueController issueController = new IssueController();

issueController.createIssue(issue);

}

В следующем тесте производится попытка создания задачи для проекта, которого не существует. Ниже приведен листинг для данного теста.

@Test(expected = InvalidIssueException.class)

public void createIssueTestWithInvalidIssue()

{

Issue issue = new Issue();

issue.setName(“Create new button”);

issue.setDescription(“Description”);

issue.setCreator(new User(“pyotr\_romanov”));

Issue issue = new Issue();

issue.setId(666);

IssueController issueController = new IssueController();

issueController.createIssue(issue);

}

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

@Test(expected = InvalidIssueStatusException.class)

public void changeIssueStatusTestToInvalidStatus()

{

Issue issue = new Issue();

issue.setId(10);

issue.setIssueStatus(new IssueStatus(2));

IssueController issueController = new IssueController();

issueController.changeIssueStatus(issue, new IssueStatus(3));

}

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

@Test(expected = InvalidIssueStatusException.class)

public void changeIssueStatusTestToInvalidStatus()

{

Issue issue = new Issue ();

issue.setId(666);

issue.setIssueStatus(new IssueStatus(2));

IssueController issueController = new IssueController();

issueController.changeIssueStatus(issue, new IssueStatus(3));

}

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

@Test

public void getIssueTest(){

IssueController issueController = new IssueController();

List<Issue> issues = issueController.getIssueIssues(2);

assertEquals(4, issues.size());

}

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

@Test

public void getIssuesTest()

{

IssueController issueController = new IssueController();

List<Issue> issues = issueController.getProjectIssues(2);

assertEquals(2, issues.size());

}

Далее смоделируем ситуацию поиска задач в нашем программном продукте. В таком случае ожидается, что метод getIssues вернет список всех задач. В конкретном случае задач всего 4. Листинг кода для данного теста приведен ниже.

@Test

public void getIssueTest(){

IssueController issueController = new IssueController();

List<Issue> issues = issueController.getProjectIssues(2);

assertEquals(2, issues.size());

}

**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.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 6.1 Системные требования

При разработке программного средства был использован язык программирования Java. Использование этого языка не накладывает ограничений на используемую операционную систему. Однако для работоспособности серверной части данного приложения необходимо установить некоторое программное обеспечение.

Для исполнения программных модулей, написанных на языке Java необходимо установить JVM. При разработке настоящего программного продукта использовалась восьмая версия Java.

Для нормальной работы программного средство необходимо учитывать минимальные системные требования. Минимальные системные требования приведены в таблице 6.1.

Таблица 6.1 – Минимальные системные требования

|  |  |
| --- | --- |
| Параметр | Минимальное значение |
| Операционная система | Windows 7 SP2 или Ubuntu Linux 12.04 LTS |
| Процессор | Intel Celeron |
| Оперативная память | 2048 MB DDR3 |
| Жесткий диск | HDD 120 GB |
| Сетевое подключение | 10 MB/s |

Для использования программного продукта на стороне клиента необходимо любое устройство с доступом в интернет и предустановленным браузером Google Chrome версии 45.0 и выше либо Mozilla Firefox 50.0 и выше. Установка какого-либо другого ПО не нужна.

Для запуска приложения на стороне сервера необходимо установить Apache Maven версии 3.5.0. После установки необходимо открыть терминал в папке проекта и написать команду mvn clean package. После этого необходимо ввести команду java –jar Feedback.jar. Выполнение команды может занять несколько минут, поскольку все используемые библиотеки будут загружены из сети интернет. После этого в терминале появится номер порта, на котором запущено приложение. Для того, чтобы проверить, что все правильно работает необходимо ввести в браузере адрес <http://localhost:номер_порта>.

## 6.2 Использование программного средства

**6.2.1** Стартовая страница и форма аутентификации

На рисунке 6.1 представлена стартовая страница разработанного программного продукта.

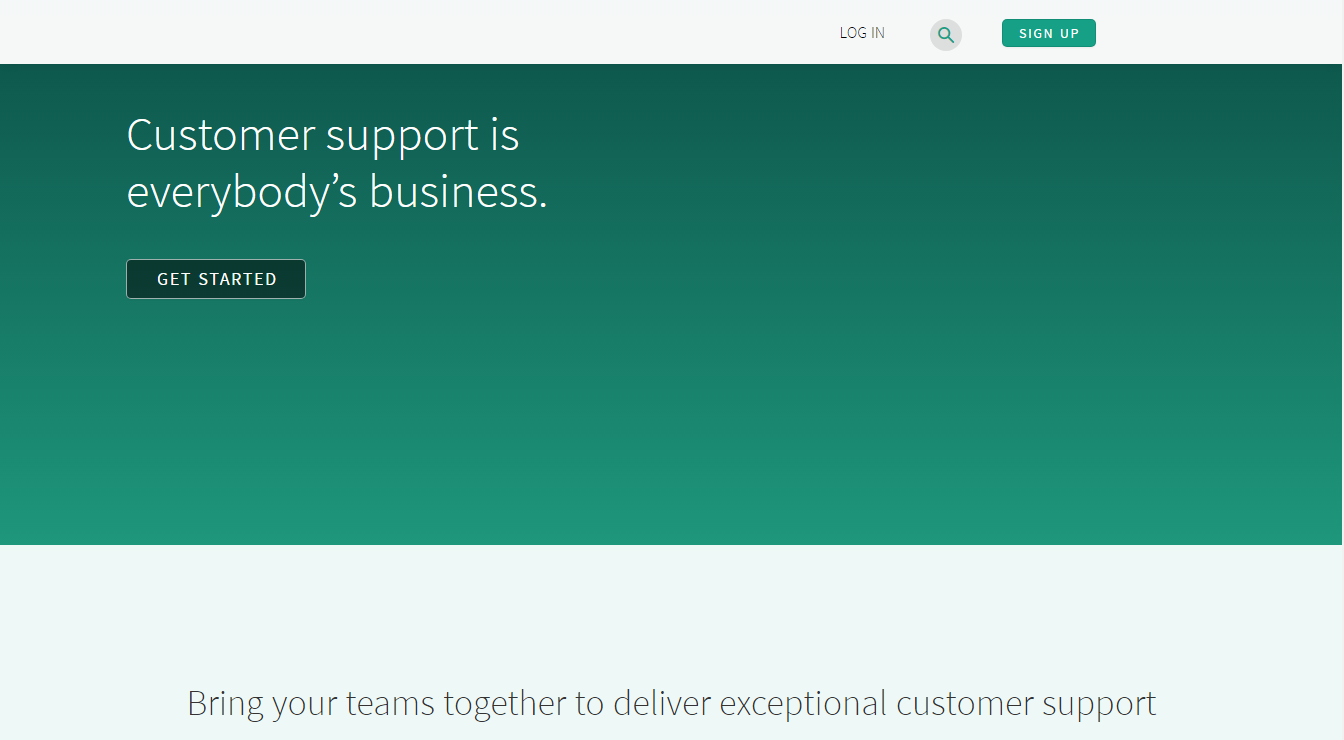
****

Рисунок 6.1 – Стартовая страница

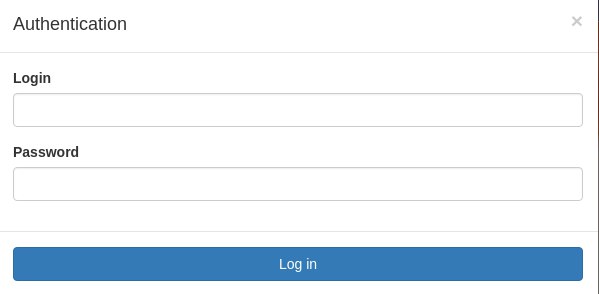
Чтобы пройти аутентификацию пользователю необходимо нажать кнопку «Login», после этого будет отображено модальное окно, где пользователю необходимо ввести логин и пароль (см. рис. 6.2).

Рисунок 6.2 – Окно аутентификации

В случае, если пользователем были введены не верные логин и пароль, будет выведено сообщение об ошибке. Если же аутентификация была пройдена успешно, то пользователь перенаправляется на главную страницу приложения.

**6.2.2** Главная страница приложения

На рисунке 6.3 представлена главная страница разработанного программного продукта.

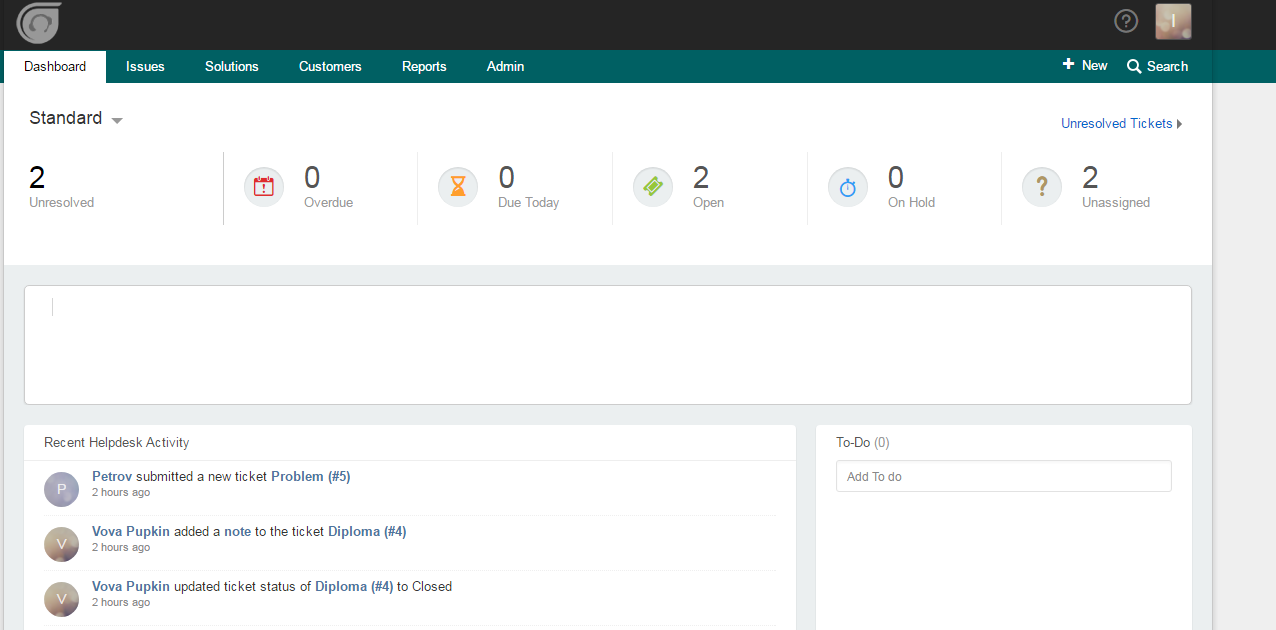


Рисунок 6.3 – Главная страница

На главной странице представлены колонки, позволяющие выбрать раздел, где мы хотим работать. По умолчанию открыта вкладка Dashboard, где мы можем увидеть список задач, которые необходимо закрыть, задачи которые не открыты, закрытые задачи. Также в левом нижнем углу можно увидеть недавнюю активность пользователей.

**6.2.3** Создание новой задачи

Создать новую задачу может обычный пользователь или администратор. На главной странице, где отражены несколько вкладок для работы с задачами необходимо нажать на кнопку New(см. рис. 6.4). Далее будет список, где необходимо выбрать пункт Issue.

В окне создания нового проекта необходимо ввести название задачи, (см. рис. 6.5) и краткое описание самого проекта. Поля со звездочками явявляются обязательными.

После того, как все поля были заполнены необходимо нажать кнопку «Submit». После создания проекта он будет отображен в списке задач пользователя. Теперь пользователь может перейти к задаче и начать работу с ней.

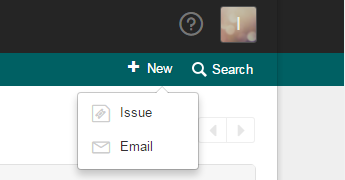


Рисунок 6.4 – Окно выбора новой задачи

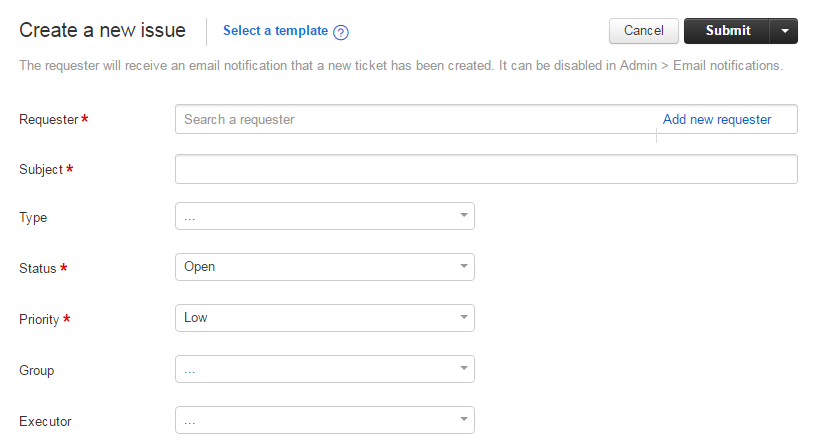


Рисунок 6.5 – Окно создания новой задачи

**6.2.4** Окно задач

На рисунке 6.6 представлен вид главного окна проекта.

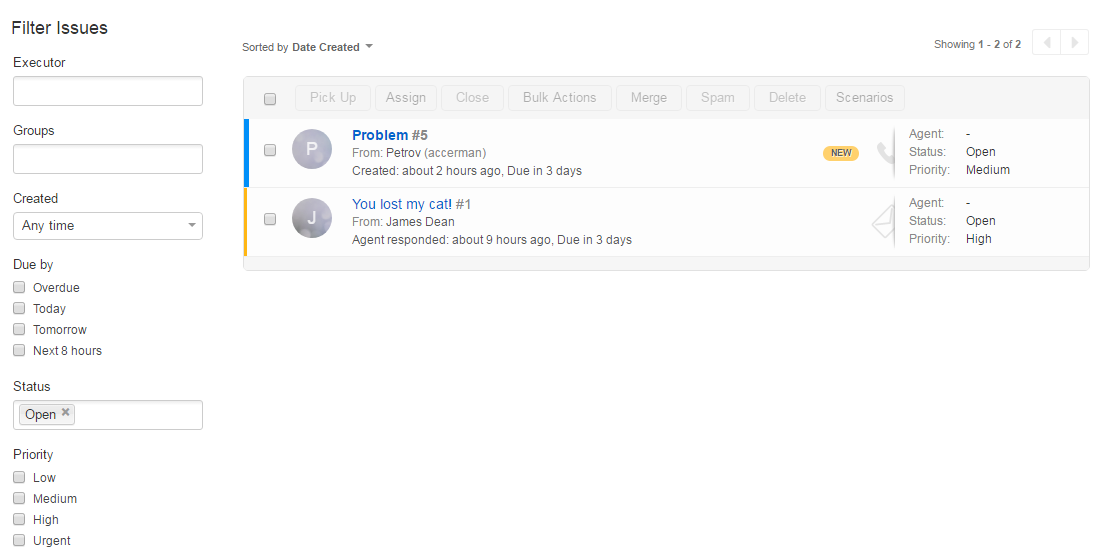


Рисунок 6.6 – Главное окно задач

Для того, чтобы перейти на вкладку главного окна задач, необходимо нажать на кнопку Issues. Здесь можно увидеть, что задачи отображаются по определенным компонентам, которые используются для удобной сортировки задач. Здесь можно увидеть, что присутствуют фильтры времени, приоритета и исполнителя.

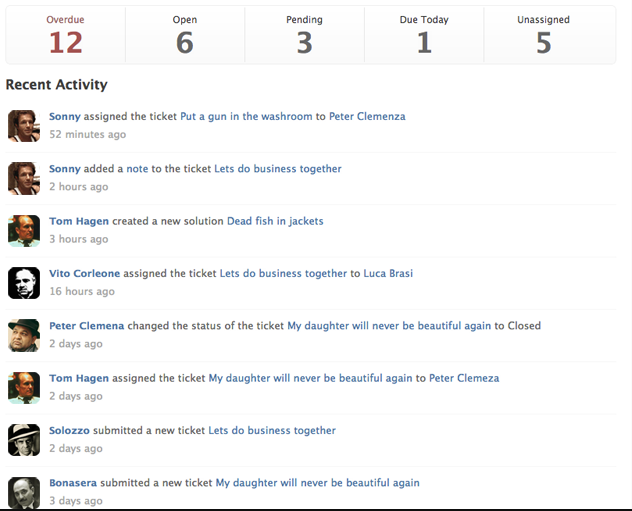


Рисунок 6.7 – Окно активности пользователей

**6.2.5** Удаление задачи

Для удаления задачи необходимо выбрать данную задачу и выбрать вкладку more. Затем можно удалить задачу (см. рис. 6.8).



Рисунок 6.8 –Уведомление удаления задачи

Для удаления задачи необходимо нажать кнопку «Delete», иначе кнопку «Cancel». При удалении проекта так же будут удалены все сущности с ним связанные: статус, приоритет, исполнитель и др.

**6.2.6** Просмотр списка пользователей и добавление в группу

Для просмотра полного списка пользователей необходимо нажать кнопку «Customers» в верхней части окна проекта. После этого будет отображено окно со списком всех пользователей, которые зарегистрированы в системе. Так же имеется возможность поиска пользователей (см. рис. 6.9).

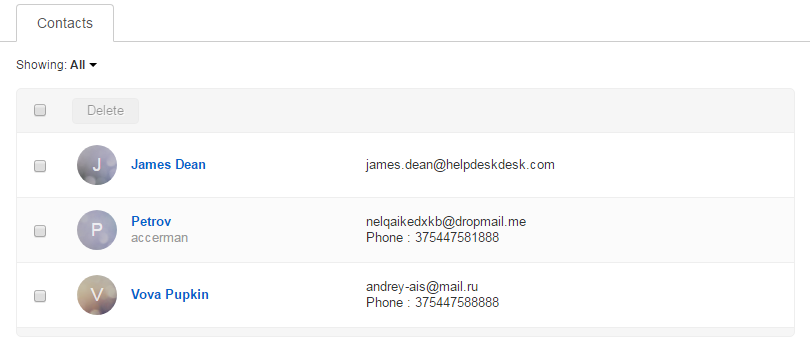


Рисунок 6.9 – Окно пользователей

Для добавления пользователя в группу необходимо выбрать одного из пользователей (доступно только руководителю проекта). После этого будет отображено окно редактирования группы (см. рис. 6.10).

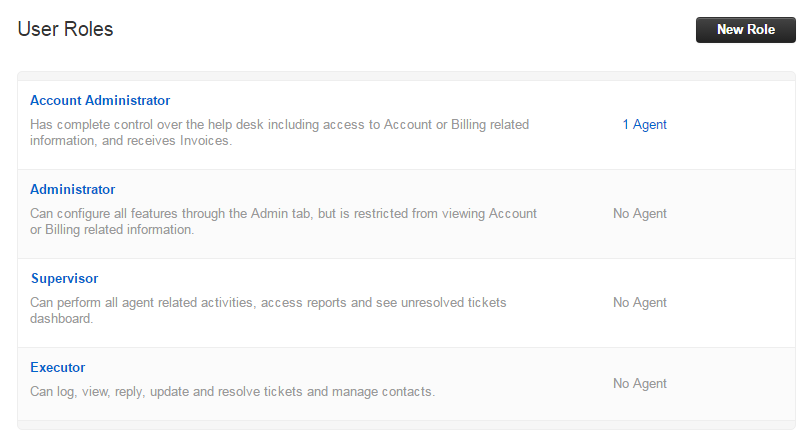


Рисунок 6.10 – Окно изменения группы пользователя

В представленном окне синим цветом подсвечена текущие роли, которые присутствуют в системе. Для смены роли необходимо щелкнуть по названию роли, которую необходимо установить пользователю, после этого окно редактирования роли будет автоматически закрыто.

**6.2.7** Добавление комментария к задаче

Для добавления комментария к задаче необходимо выбрать данную задачу (доступно только руководителю проекта, либо пользователю с соответствующими правами). После этого будет отображено окно с задачей.

В данном окне необходимо выбрать вкладку Reply, чтобы можно было добавить свой комментарий (см. рис. 6.11).

После этого пользователь может в личном кабинете увидеть пришедшее уведомление от исполнителя. Пользователь с помощью этой же вкладки может добавить свой комментарий к данной задаче.

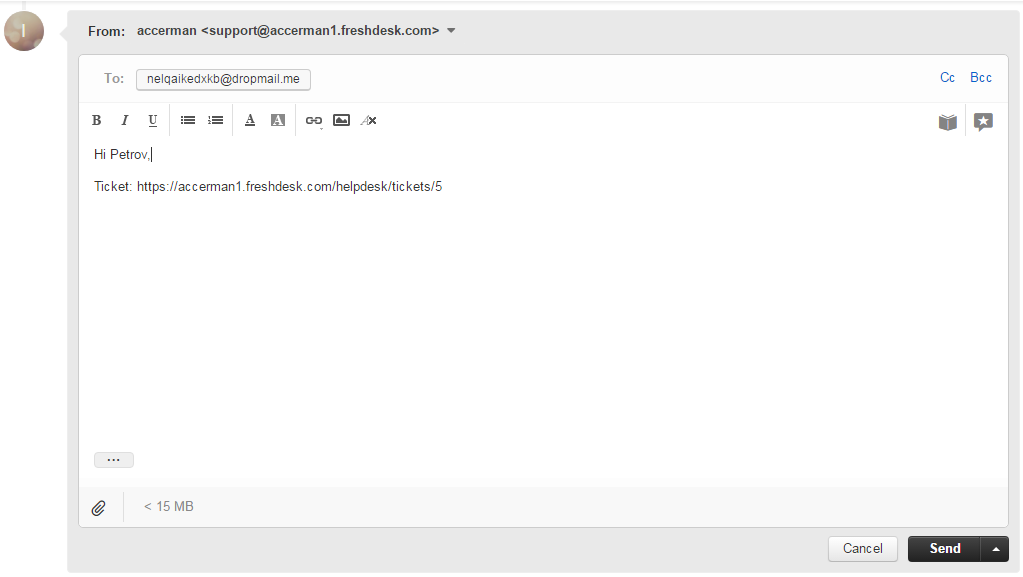


Рисунок 6.11 – Окно добавления комментария

**6.2.8** Закрытие задачи

После создания заявки, а затем ее обработки, пользователь решает закрывать заявку или нет. В случае, если пользователь остаётся не удовлетворенным ответом, он может продолжить переписку по данной задаче и пытаться найти ответ на свой вопрос. В случае, если пользователь удовлетворен ответом, он может перейти во вкладку с задачами, в которых он участвует. Затем он должен выбрать ту задачу, в которой он является создателем, выбрать ее и нажать на кнопку Mark issue as closed (см. рис. 6.12).

После этого данная задача помечается как решенная и пользователь может оценить уровень удовлетворенности работы сотрудников по данному продукту. Это помогает администратору более качественно распределять заявки, для эффективной помощи клиентам (см. рис. 6.13).

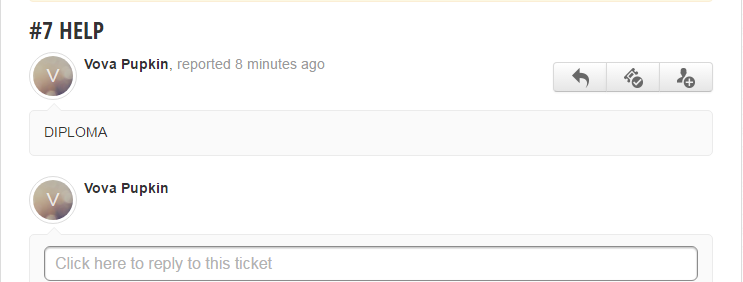


Рисунок 6.12 – Окно закрытия задачи

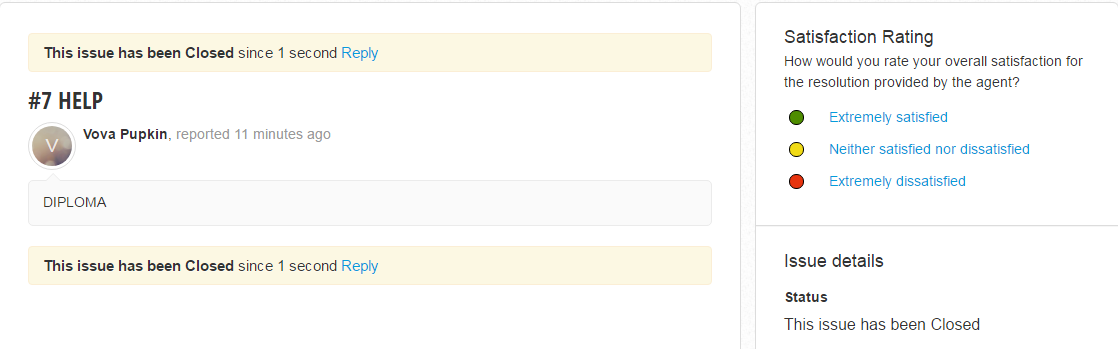


Рисунок 6.13 – Окно уведомления об закрытии задачи

Также, после того, как задача была помечена как решенная, пользователь может нажать на Reply, помеченную синим цветом и прокомментировать данную задачу.

**6.2.9** Отправление сообщения

Создать новую задачу может обычный пользователь или администратор. На главной странице, где отражены несколько вкладок для работы с задачами необходимо нажать на кнопку New. Далее будет список, где необходимо выбрать пункт E-mail. Далее необходимо ввести пользователя, которому будет адресовано сообщение, тему письма, а также само сообщение. Для отправки сообщения необходимо нажать кнопку Send и сообщение будет отправлено.

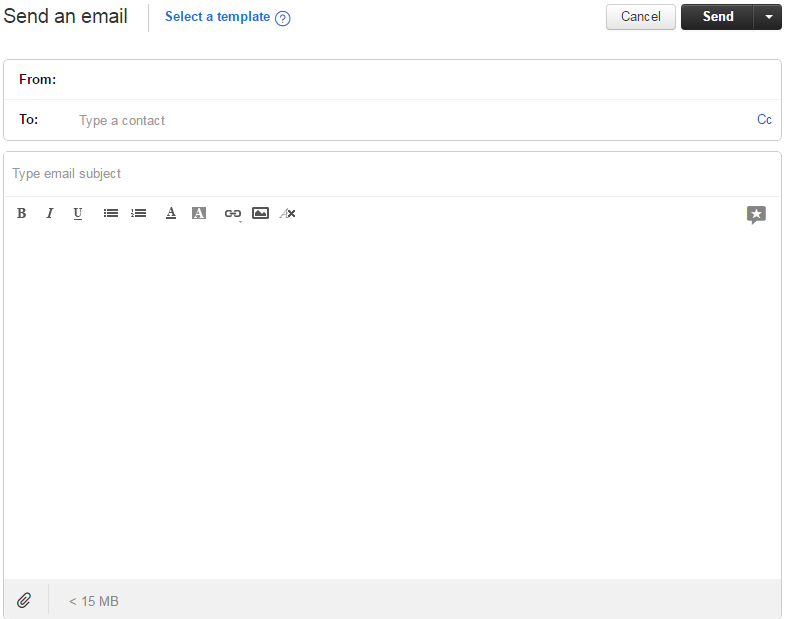
****

Рисунок 6.14 – Окно редактирования задачи

В данной главе детально был рассмотрен результат проделанной работы над настоящим программным продуктом. Были рассмотрены основные функции разработанного программного обеспечения и их работа. Так же были описаны правила использования данного программного продукта.