

ВВЕДЕНИЕ

CFM (англ. Customer feedback management — сервис управления обратной связью) — это веб-приложения, которые позволяют компаниям управлять структурированными предложениями и жалобами пользователей. Сервисы управления обратной связью используют данные от пользователей для оценки качества обслуживания клиентов и повышения удовлетворенности клиентов. Программное обеспечение должно предоставлять широкую функциональность анализа данных и генерации отчетов, которая включает в себе функции статистического анализа, а также функции централизованного управления панелями респондентов. Help Desk — это ресурс, предназначенный для предоставления клиенту или конечному пользователю информации и поддержки, связанных с продуктами и услугами компании или учреждения. Применение автоматизированной централизованной обработки данных дает возможность учитывать индивидуальные потребности заказчиков, а также уменьшить штат сотрудников для решения данных проблем. Оперативность обработки данных позволяет осуществлять раннее выявление рисков и их уменьшение. CFM системы широко применяются в различных сферах, в том числе управления кадрами, IT, маркетинга, продаж и клиентского обслуживания. Стоит отметить, что с каждым годом увеличивается объем и расширяется сфера использования данных систем. Активное внедрение систем данного рода позволяет организации в целом слышать своих ключевых потребителей, учиться и отвечать на их нужды, а также в более короткие сроки выпускать обновленные версии продукта с учетом пожеланий пользователей.

Актуальность данного дипломного проекта, в первую очередь, заключается в том, что данные системы позволяют сократить время и человеческие ресурсы для создания качественных программных средств. Наличие обратной связи у программного продукта — наилучший способ гарантировать, что данный продукт действительно востребован. Обратная связь с клиентами обычно используется на протяжении всего процесса разработки продукта, чтобы удовлетворить большинству пожеланиям клиентов. Также обратная связь с клиентами позволяет сохранить и приумножить целевую аудиторию данного программного продукта. Улучшение качества обслуживания клиентов должно быть основной причиной, по которой собираются отзывы клиентов. Процесс завоевания нового бизнеса и удержания существующих клиентов становится все сложнее.

Существует несколько крупных аналогов программного средства для обеспечения поддержки клиентов. К ним относятся Uservoice, Omnidesk, UserEcho и прочие. Рассмотрим несколько из них.

UserVoice — это сервис, созданный для предприятий различных размеров, которые регулярно взаимодействуют со своими клиентами, чтобы принять различные жалобы и предложения, а также предоставить обратную связь по данному продукту.

Omnidesk — российский аналог UserVoice, сервис, который предоставляет многоканальную службу поддержки. Данный сервис позволяет взаимодействовать с клиентами посредством: Twitter, Facebook e-mail и обратную связь. Особенностью данного сервиса является то, что контакты пользователя с различных каналов связи собираются и обрабатываются в аккаунте пользователя на Omnidesk, что позволяет сотрудникам службы поддержки оказывать более качественную и своевременную помощь клиентам.

В данном дипломном проекте поставлена задача разработать программное средство для обеспечения поддержки клиентов, главной целью которого будет сбор и ранжирование обратной связи пользователей. В данной работе будет реализована как публичная помощь по программному продукту, реализованная в виде helpdesk, так и анонимная помощь, с помощью которой клиент может написать на почту и получить анонимный ответ. Ранжирование обратной связи пользователей заключается в том, что клиенты видят, какие жалобы и предложения и предложения появляются по программному продукту и могут их отмечать, как наиболее актуальные. Чем более актуальна проблема, тем выше она будет находиться в списке проблем, что позволит разработчикам программного обеспечения исправлять в первую очередь наиболее важных недочётов, а также удовлетворить более широкую аудиторию своих клиентов.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

На сегодняшний день существует множество приложений, которые обладают сложной структурой. Поддержание работоспособности таких приложений является одной из самых важных и сложных задач для компаний. В связи с этим, возросла актуальность появления helpdesk сервисов. Современный рынок широко представлен web-приложениями, созданными для оказания услуг обратной связи с клиентами, основной целевой аудитории, удовлетворенность которой показывает, насколько важен и актуален продукт компании.

1.1.1 Uservoice

UserVoice HelpDesk — инструмент поддержки для отслеживания и реагирования на проблемы клиентов. В рамках UserVoice каждая компания может легко оптимизировать процессы, чтобы помочь сотрудникам обеспечить наилучший ход обслуживания клиентов.

Основной особенностью данного продукта является то, что он использует игровую механику для повышения эффективности службы поддержки. Разработчики helpdesk-сервиса UserVoice решили использовать игровую механику для того, чтобы сделать работу сотрудников поддержки более интересной, а их клиентов — более удовлетворенными. В игровой вселенной UserVoice сотрудники службы поддержки соревнуются друг с другом, зарабатывая очки (см. рис. 1.1).

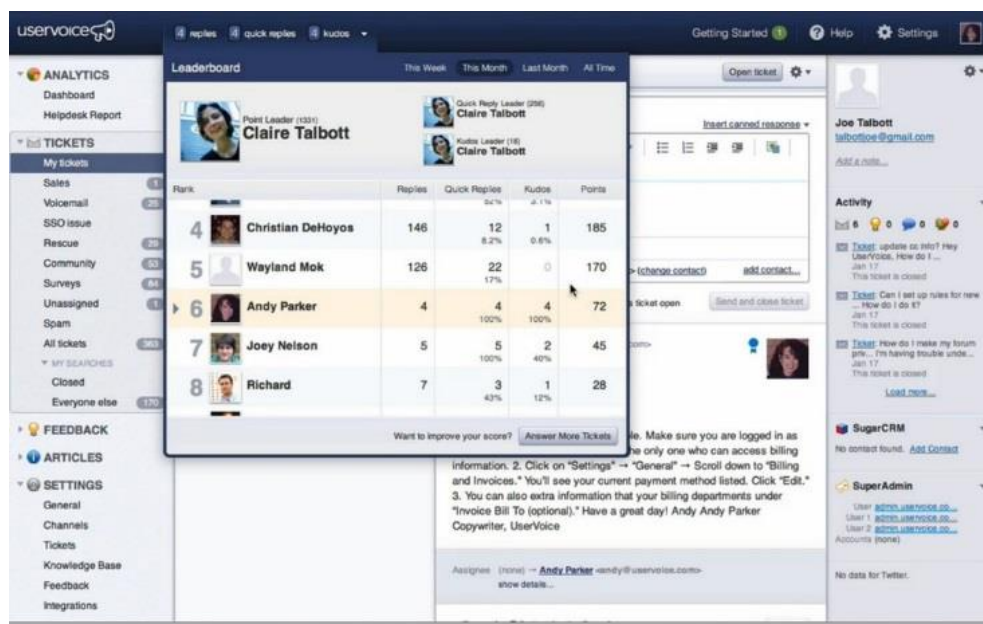


Рисунок 1.1 – Экран рейтинга Uservoice

За каждый ответ клиенту сотрудник получает бонус, причем если он ответил быстро (в течении одного часа) - очки утраиваются. А если клиент доволен ответом, он может подарить сотруднику виртуальный приз — звезду, которая стоит много очков.

Также данный сервис обладает ранжированием проблем клиентов. Клиент может увидеть все жалобы и предложения по поводу продукта и проголосовать за них. Самые актуальные будут показаны сверху, что облегчает разработчикам устранять наиболее важные проблемы (см. рис. 1.2).

How can we improve the IE developer experience?

Hot

Top Ideas

New

Category

My feedback

2,421
votes

3 votes

Stop Internet Explorer development
This browser was a pain and is still a pain, please die.
58 comments · Performance · Flag idea as inappropriate...

1,255
votes

Vote

Object.observe();
Observe changes to JS objects.
<http://wiki.ecmascript.org/doku.php?id=harmony:observe>
0 comments · JavaScript · Flag idea as inappropriate...

1,076
votes

Vote

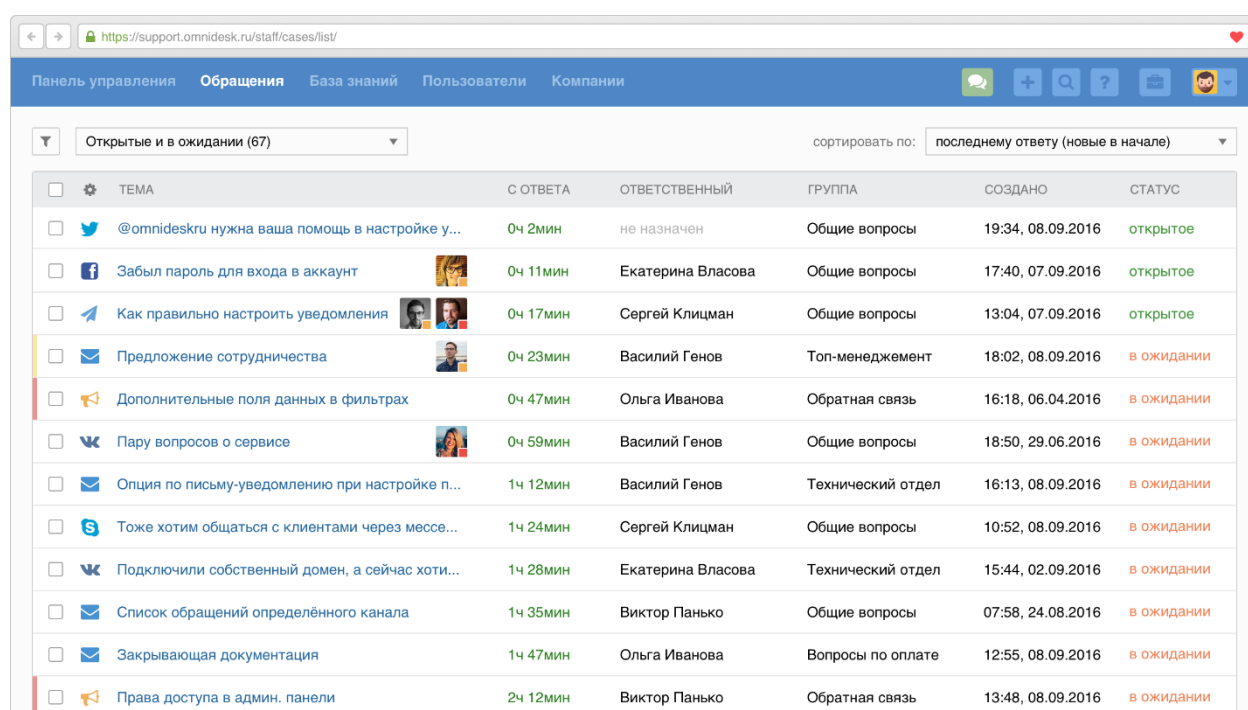
Shadow DOM (unprefixed)
Enables DOM tree encapsulation. Without it, widgets may inadvertently break pages by using conflicting CSS selectors, class or id names, or JavaScript variables.
<http://dvcs.w3.org/hg/webcomponents/raw-file/tip/spec/shadow/index.html>
0 comments · HTML · Flag idea as inappropriate...

Рисунок 1.2 – Экран ранжирования проблем Uservoice

Uservoice представляет собой web-приложение, но также имеются приложение для мобильных устройств под управлением операционной системы Android и IOS. Одним из главных недостатков является то, что данный сервис не является бесплатным, а всего лишь предоставляет бесплатный пробный период на 14 дней. Также отсутствует мультиязычность приложения, что сильно влияет на рынки, где люди не используют английский язык.

1.1.2 Омнидеск

Омнидеск также, как и UserVoice, является сервисом обратной связи, который предоставляет многоканальную службу поддержки. Данный сервис позволяет покрывать многие популярные и актуальные каналы связи с клиентами: Twitter, Facebook, e-mail и обратную связь. Особенностью данного сервиса является то, что контакты пользователя с различных каналов связи собираются и обрабатываются в аккаунте пользователя на Омнидеск, что позволяет сотрудникам службы поддержки оказывать более качественную и своевременную помощь клиентам (см. рис. 1.3).



<input type="checkbox"/>	ТЕМА	С ОТВЕТА	ОТВЕТСТВЕННЫЙ	ГРУППА	СОЗДАНО	СТАТУС
<input type="checkbox"/>	@omnideskru нужна ваша помощь в настройке у...	0ч 2мин	не назначен	Общие вопросы	19:34, 08.09.2016	открытое
<input type="checkbox"/>	Забыл пароль для входа в аккаунт	0ч 11мин	Екатерина Власова	Общие вопросы	17:40, 07.09.2016	открытое
<input type="checkbox"/>	Как правильно настроить уведомления	0ч 17мин	Сергей Клицман	Общие вопросы	13:04, 07.09.2016	открытое
<input type="checkbox"/>	Предложение сотрудничества	0ч 23мин	Василий Генцов	Топ-менеджмент	18:02, 08.09.2016	в ожидании
<input type="checkbox"/>	Дополнительные поля данных в фильтрах	0ч 47мин	Ольга Иванова	Обратная связь	16:18, 06.04.2016	в ожидании
<input type="checkbox"/>	Пару вопросов о сервисе	0ч 59мин	Василий Генцов	Общие вопросы	18:50, 29.06.2016	в ожидании
<input type="checkbox"/>	Опция по письму-уведомлению при настройке п...	1ч 12мин	Василий Генцов	Технический отдел	16:13, 08.09.2016	в ожидании
<input type="checkbox"/>	Тоже хотим общаться с клиентами через мессе...	1ч 24мин	Сергей Клицман	Общие вопросы	10:52, 08.09.2016	в ожидании
<input type="checkbox"/>	Подключили собственный домен, а сейчас хоти...	1ч 28мин	Екатерина Власова	Технический отдел	15:44, 02.09.2016	в ожидании
<input type="checkbox"/>	Список обращений определённого канала	1ч 35мин	Виктор Панько	Общие вопросы	07:58, 24.08.2016	в ожидании
<input type="checkbox"/>	Закрывающая документация	1ч 47мин	Ольга Иванова	Вопросы по оплате	12:55, 08.09.2016	в ожидании
<input type="checkbox"/>	Права доступа в админ. панели	2ч 12мин	Виктор Панько	Обратная связь	13:48, 08.09.2016	в ожидании

Рисунок 1.3 – Экран аккаунта пользователя Омнидеск

Как отмечалось выше, у сервиса Омнидеск присутствует интеграция со многими каналами связи, что также является преимуществом перед сервисом UserVoice. Онлайн система для поддержки клиентов Омнидеск реализована с помощью JIRA — сервисом, которым пользуются практически все компании, имеющие отношение к разработке. Удобство интеграции заключается в том, что она двухсторонняя: сотрудники поддержки работают с JIRA-проблемами (issues) из Омнидеска, а разработчики имеют доступ к обращениям прямо из JIRA. Также есть возможность отправки уведомлений из одного сервиса в другой.

В отличие от Uservoice, в сервисе Омнидеск отсутствует ранжирование проблем, а также игровая механика для людей, ответственных за обратную связь с клиентами. Однако стоит заметить, что в данном сервисе присутствует подобие системы оценок обслуживания клиентов (см. рис. 1.3).

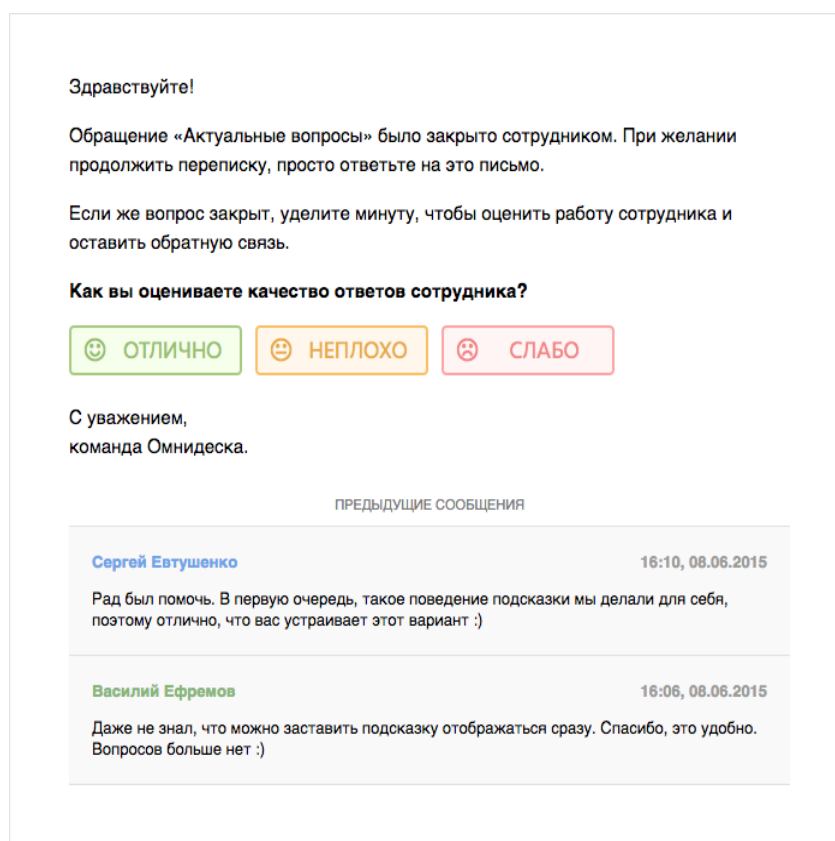


Рисунок 1.4 – Экран обслуживания пользователя Омнидеск

Сервис Омнидеск, как и Userveice, является web-приложением, но не имеется приложений под мобильные операционные системы. Несмотря на интеграцию со многими известными сервисами, данный продукт ориентирован только на русскоязычную аудиторию, что сильно мешает продвижению продукта на другие рынки.

1.2 Обзор используемых технологий

В данном разделе будут рассмотрены технологии, на которых в дальнейшем будет реализован дипломный проект.

1.2.1 Выбор технологий клиентской части приложения

Клиентская часть web-приложений существует для создания пользовательского интерфейса, а также для формирования запросов к серверу и обработки ответов от него. На сегодняшний день web-приложения получили большую популярность и активно вытесняют десктопные приложения. С развитием web-приложений появилось множество концепций и технологий по созданию клиентской части такого приложения. Сегодня большой спрос и популярность получил язык JavaScript. Существует множество фреймворков данного языка программирования, которые имеют свои достоинства и

недостатки для создания приложений. В данном разделе мы рассмотрим наиболее популярные: AngularJS, ReactJS, BackboneJS.

AngularJS является популярным фреймворком, который широко используется для создания и поддержания сложных веб-приложений. Популярность AngularJS огромна и компании, использующие его, столь же разнообразны: Domino's Pizza, Ryanair, iTunes Connect, PayPal, Google Checkout. AngularJS имеет открытый исходный код при поддержке Google. AngularJS позиционируется как расширение HTML для создания сложных веб-приложений.

AngularJS — MVC-фреймворк. Он имеет двусторонний дата-биндинг между моделями и представлениями (view). Эта привязка данных позволяет проводить автоматическое обновление с обеих сторон всякий раз, когда происходит изменение данных. Это позволяет создавать многократно используемые компоненты представления, что обеспечивает легкий обмен данными между серверной и клиентской частью.

Данный фреймворк можно использовать при построении сложного web-приложения, нуждающемся в едином модульном фреймворке. Однако, данный фреймворк имеет ряд недостатков. Самые главные недостатки являются в сложности освоения данного фреймворка и отсутствии совместимости между первой и второй версиями. Но в тоже время, огромное количество документации делает AngularJS одним из самых используемых фреймворков в мире.

ReactJS является топовым JavaScript проектом этого года. ReactJS имеет открытый исходный код и развивается в основном с помощью Facebook при участии других крупных технологических компаний. React описывает себя как JavaScript библиотека для создания пользовательских интерфейсов.

В известном паттерне Model-View-Controller React ближе всего к пользователю. Он отвечает за представление данных, получение и обработку ввода пользователя. React — это всего лишь View приложения. React построен на парадигме реактивного программирования. Этот декларативный подход предлагает описывать данные в виде набора утверждений или формул. Изменение одного из параметров ведёт за собой автоматический пересчёт всех зависимостей. ReactJS — это отличный goto-фреймворк для создания простых веб-приложений.

Однако, данный фреймворк имеет свои недостатки. Высокий порог вхождения и отсутствие широкого и разнообразного количества документации, отсутствие множества выработанных практик делают менее привлекательным данный фреймворк.

Backbone — это простой фреймворк, который вписывается в один JavaScript файл. Backbone был разработан Джереми Ашкенасом с помощью CoffeeScript. Backbone особенно популярен среди команд, которые ищут простую структуру для небольших веб-приложений, без применения больших фреймворков как Angular или Ember. При изменении модели представление просто обновит себя самостоятельно.

Работая с Backbone, данные представляются как модели, которые могут быть созданы, проверены, удалены, и сохранены на сервере. Всякий раз, когда в интерфейсе изменяется атрибуты модели, модель вызывает событие «change» и все представления, которые отображают состояние модели, могут быть уведомлены об изменении атрибутов модели, с тем чтобы они могли отреагировать соответствующим образом — например, перерисовать себя с учетом новых данных. В готовом приложении на Backbone отсутствует необходимость писать код, ищущий элемент с определенным id в DOM и обновлять HTML вручную. При изменении модели представление просто обновит себя самостоятельно.

Backbone не предоставляет структуры. Это всего лишь набор простых инструментов. Для создания структуры необходимо заполнить много пустых мест. Конечно, многие из этих мест заполняются сторонними плагинами, но это значит, что необходимо принять много решений при их выборе. Виды в Backbone напрямую манипулируют DOM, поэтому их сложно тестировать и сложнее повторно использовать.

Выполнив детальный разбор наиболее востребованных технологий клиентской части приложения, а также оценив порог вхождения и наличие обширной и понятной документации, можно увидеть, что для данного дипломного проекта наиболее актуальным и удобным вариантом является фреймворк AngularJS.

1.2.2 Выбор технологий для разработки серверной части приложения

Существует множество различных языков программирования и технологий для разработки серверной части приложения. В последнее время широко распространены следующие технологии: фреймворк Spring, реализованный на языке Java, а также Node.js.

Spring Framework — универсальный фреймворк с открытым исходным кодом для Java-платформы. Spring имеет собственную MVC-платформу веб-приложений, которая не была первоначально запланирована. Разработчики Spring решили написать её как реакцию на то, что они восприняли как неудачность конструкции (тогда) популярного Apache Struts, а также других доступных веб-фреймворков. В частности, по их мнению, было недостаточным разделение между слоями представления и обработки запросов, а также между слоем обработки запросов и моделью. Spring MVC является фреймворком, ориентированным на запросы. В нем определены стратегические интерфейсы для всех функций современной запросно-ориентированной системы. Цель каждого интерфейса — быть простым и ясным, чтобы пользователям было легко его заново имплементировать, если они того пожелают. MVC прокладывает путь к более чистому front-end-коду. Все интерфейсы тесно связаны с Servlet API. Эта связь рассматривается некоторыми как неспособность разработчиков Spring предложить для веб-приложений абстракцию более высокого уровня. Однако эта связь оставляет

особенности Servlet API доступными для разработчиков, облегчая все же работу с ним.

Spring предоставляет бóльшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба. особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

Node.js — программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Однако данная платформа имеет ряд недостатков. Отсутствуют основные инструменты — существует множество альтернатив, достоинства и недостатки которых не ясны из документации, приходится лично пробовать и выбирать. Очень много заброшенных проектов. Для многих стандартных задач нет какого-то готового и законченного решения. Очень слабая интеграция между инструментами. Приходится при необходимости писать обертки руками.

Исходя из сравнения вышеозначенных фреймворков видно, что фреймворк Spring наиболее целесообразен для данного дипломного проекта из-за наличия широкой документации и простоты написания серверной части приложений.

1.2.3 Выбор СУБД

В настоящее время существует множество реляционных баз данных, которые имеют свои достоинства и недостатки, предназначены под различные платформы. В данном разделе рассмотрим наиболее актуальные и подходящие реляционные базы данных, основываясь на технологиях, выбранных для данного дипломного проекта.

SQLite является компактной встраиваемой реляционной базой данных. Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, а предоставляет библиотеку, с которой программа компонуется. Такой подход уменьшает накладные расходы, время отклика и

упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором выполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется. SQLite поддерживает динамическое типизирование данных. Возможные типы полей: INTEGER, REAL, TEXT, BLOB.

MySQL является свободной реляционной системой управления базами данных. MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц.

Для данного дипломного проекта можно увидеть, что реляционная база данных MySQL наиболее подходит для данного дипломного проекта, нежели SQLite. MySQL имеет преимущество в следующих компонентах:

- Дополнительные приложения позволяют довольно легко работать с базами данных.
- MySQL поддерживает большинство функционала SQL.
- Большое количество функций, обеспечивающих безопасность, которые поддерживаются по умолчанию.
- MySQL легко работает с большими объемами данных и легко масштабируется.
- Упрощение стандартов позволяет MySQL значительно увеличить производительность.

1.3 Выводы

Данный дипломный проект будет реализован с помощью серверной и клиентской частей. Клиентская часть будет написана с использованием фреймворка AngularJS, реализованном на языке JavaScript. Серверная часть будет написана с использованием фреймворка Spring, реализованном на языке Java. В качестве СУБД была выбрана MySQL.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

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

В при изображена диаграмма использования разрабатываемого программного продукта. В системе можно выделить четыре актера: пользователь, исполнитель запросов и непосредственно программное средство.

Все процессы, происходящие в системе, сводятся к запросу к программному средству. Пользователь может отправлять запросы, ранжировать проблемы и предложения, оценивать исполнителей запросов. В обязанности исполнителя запросов входит обработка запросов и удовлетворение запросов пользователей.

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:

- модуль API веб-сервиса;
- модуль аутентификации;
- модуль прямой и обратной связи;
- модуль ранжирования проблем и предложений;
- модуль маршрутизации;
- модуль авторизированной системы учёта заявок от клиентов;
- модуль администрирования и авторизации;
- модуль доступа к данным;
- база данных веб-сервиса.

На стороне веб-сайта можно выделить следующие функциональные блоки:

- модуль пользовательского интерфейса;
- модуль взаимодействия с веб-сервисом;
- модуль визуализации;

Структурная схема, где представлены все вышеперечисленные блоки приведена на чертеже ГУИР.400201.187 С1. В системе каждый модуль выполняет свою задачу и взаимодействует с другими модулями посредством интерфейсов. Рассмотрим функциональные модули разрабатываемой системы.

Модуль API веб-сервиса. Данный модуль является самым важным модулем программного продукта. Данный модуль принимает запросы от модуля маршрутизации, затем обращается, по необходимости, к другим модулям веб-сервиса, производит предварительную обработку ответа и посылает ответ модулю маршрутизации.

Модуль аутентификации. Данный модуль отвечает за авторизацию пользователей. Авторизация 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 прямой и обратной связи. Данный модуль во многом раскрывает суть данного дипломного проекта: обеспечить связь между разработчиком программного средства и пользователем, у которого есть вопросы и предложения по данному продукту. Данный модуль предполагает обеспечить клиентам связь с разработчиками в публичном виде, где запрос и ответ будут видеть все пользователи. Также будет реализован способ связи по электронной почте, при котором запрос и ответ на него будет видеть пользователь, который отправил данный запрос.

Модуль ранжирования проблем и предложений. Данный модуль позволяет решать наиболее актуальные проблемы продукта. Ранжирование обратной связи пользователей заключается в том, что клиенты видят, какие жалобы и предложения и предложения появляются по программному продукту и могут их отмечать, как наиболее актуальные. Чем более актуальна проблема, тем выше она будет находиться в списке проблем, что позволит разработчикам программного обеспечения исправлять в первую очередь наиболее важных недочётов

Модуль маршрутизации. 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 авторизованной системы учёта заявок от клиентов. Данный модуль предназначен для реализации helpdesk — ресурса, предназначенного для предоставления клиенту или конечному пользователю информации и поддержки, связанных с продуктами и услугами компании или учреждения.

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

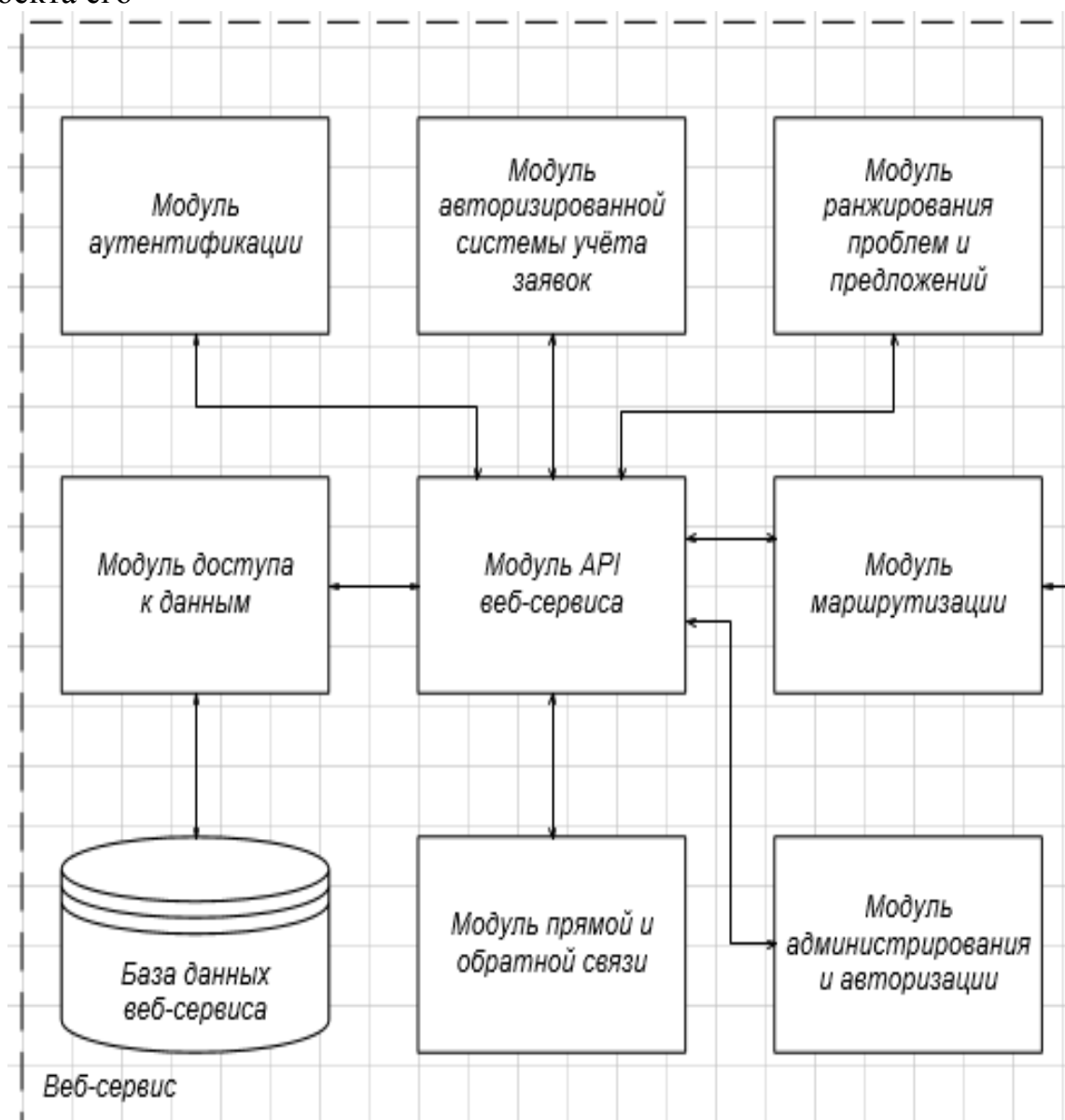


Рисунок 2.1 – Серверная часть приложения

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. *Модуль визуализации.* 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.

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 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 отправляются на веб-сервер в асинхронном режиме с использованием ajax.

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

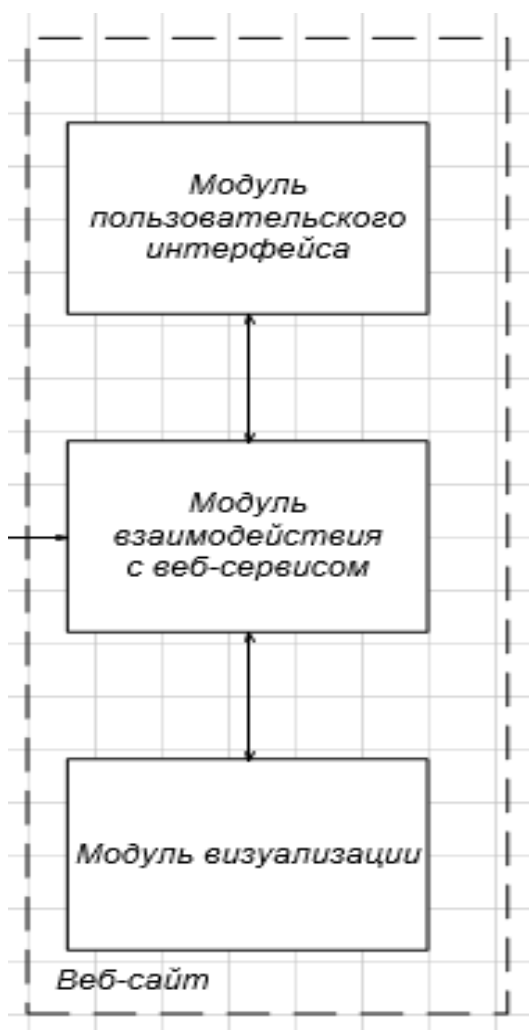


Рисунок 2.2 – Клиентская часть приложения

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

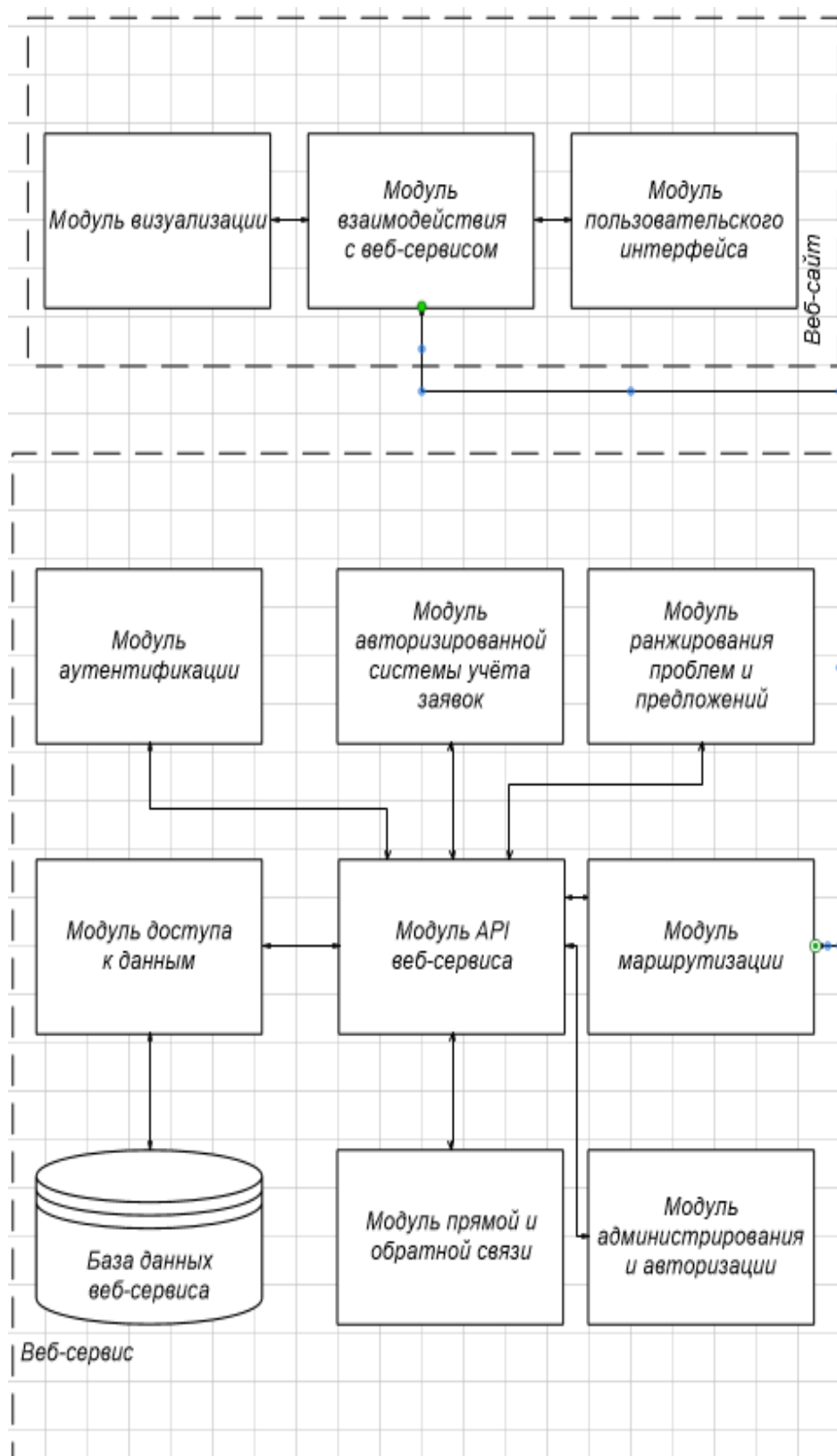


Рисунок 2.3 – Структурная схема приложения

2.2 Структура базы данных

Следующим этапом разработки программного продукта является построение базы данных. В приложении Г представлена структура базы данных программного средства отслеживания задач.

Далее будут рассмотрены таблицы базы данных разрабатываемой системы.

User. Данная таблица предназначена для хранения информации о пользователях системы. Первичным ключом здесь является поле *user_id* (id пользователя). Также присутствуют поля *f_name* (имя пользователя), *s_name* (фамилия пользователя), *user_login* (логин пользователя), *user_password* (пароль пользователя), *user_role* (роль пользователя).

Таблица *Request* используется для хранения проблем и предложений, которые интересуют клиентов некоторого программного продукта. В качестве первичного ключа используется поле *req_id* (искусственное число). Внешний ключ *req_status* ссылается на таблицу *request_status*, тем самым указывая текущий статус задачи. Внешний ключ *req_priority* ссылается на таблицу *request_priority*, тем самым указывая текущий приоритет задачи. Внешний ключ *req_executor* ссылается на таблицу *executor* и указывает исполнителя задачи. Внешний ключ *creator_id* ссылается на таблицу *User* и указывает текущего создателя задачи. Внешний ключ *req_answer* ссылается на таблицу *answer* и указывает ответ на текущий запрос. Также присутствуют поля *req_name* (имя запроса), *req_description* (описание запроса), *req_mark* (оценка запроса, с помощью которой выполняется ранжирование проблем).

Таблица *request_status* используется для хранения статуса запроса в рамках проекта. Поле *status_id* используется в качестве первичного ключа. Ключ *status_name* используется для того, чтобы указать текущий статус данного запроса. Существует 5 видов текущего статуса запроса: open, distributed, processing, checking, close.

Таблица *request_priority* используется для хранения приоритета запроса в рамках проекта. Поле *priority_id* используется в качестве первичного ключа. Ключ *priority_name* используется для того, чтобы указать текущий приоритет данного запроса. Существует 4 вида текущего приоритета запроса: low, medium, high, critical.

Lifecycle. Данная таблица используется для хранения жизненного цикла запроса. Поле *req_id* здесь является первичным ключом. Также присутствуют поля, описывающие сам жизненный цикл определенного запроса: opened (дата открытия запроса), distributed (дата распределения запроса), processing (дата обработки запроса), checking (дата проверки запроса), close (дата закрытия запроса).

Таблица *answer* предназначена для хранения прикрепленных к запросу ответов. Внешний ключ *req_id* ссылается на таблицу *Request* и является первичным ключом. Также присутствует поле *answer_content*, которое хранит сам ответ на данный запрос.

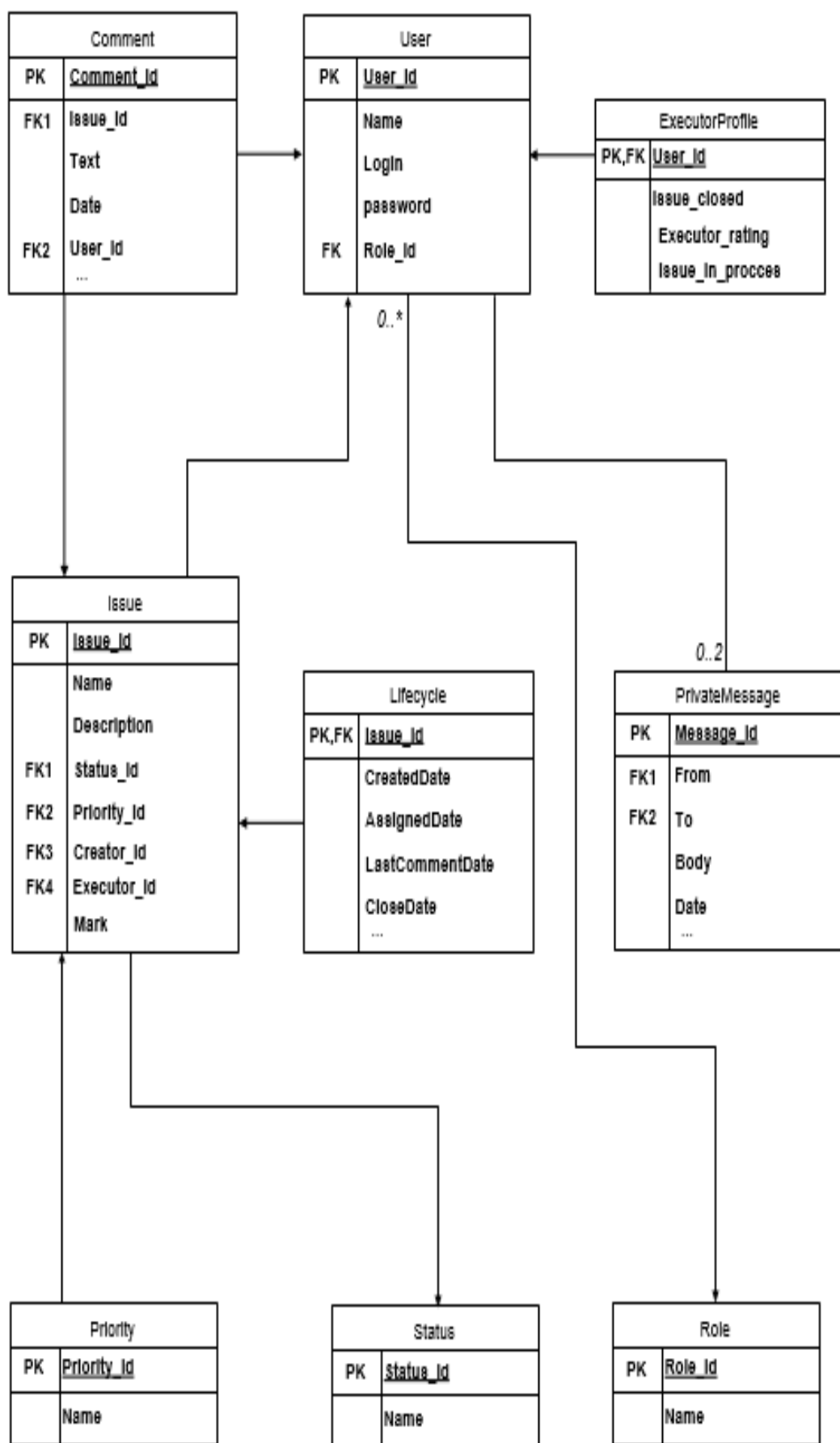


Рисунок 2.4 – Упрощенная модель данных

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.

2.3 Инфраструктура приложения

В данном приложении 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.

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.

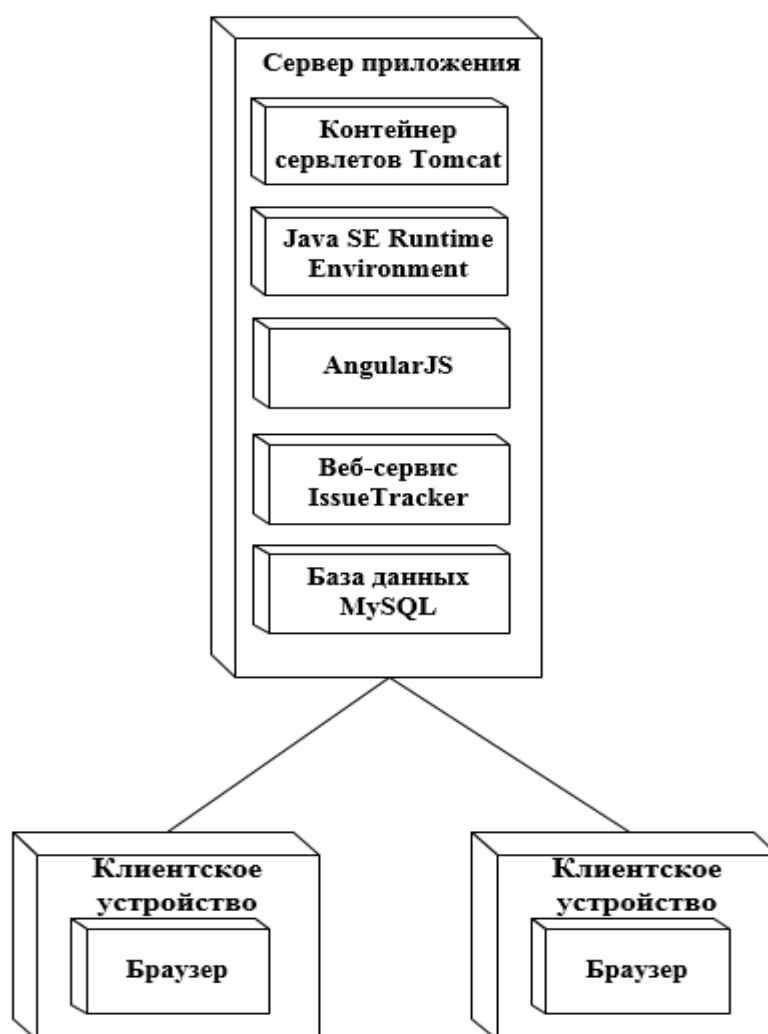


Рисунок 2.5 – Диаграмма развертывания разрабатываемого программного средства

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 примет 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 сообщений. Данный класс реализует следующие методы: *getListMessages* (метод принимает идентификатор пользователя и 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).

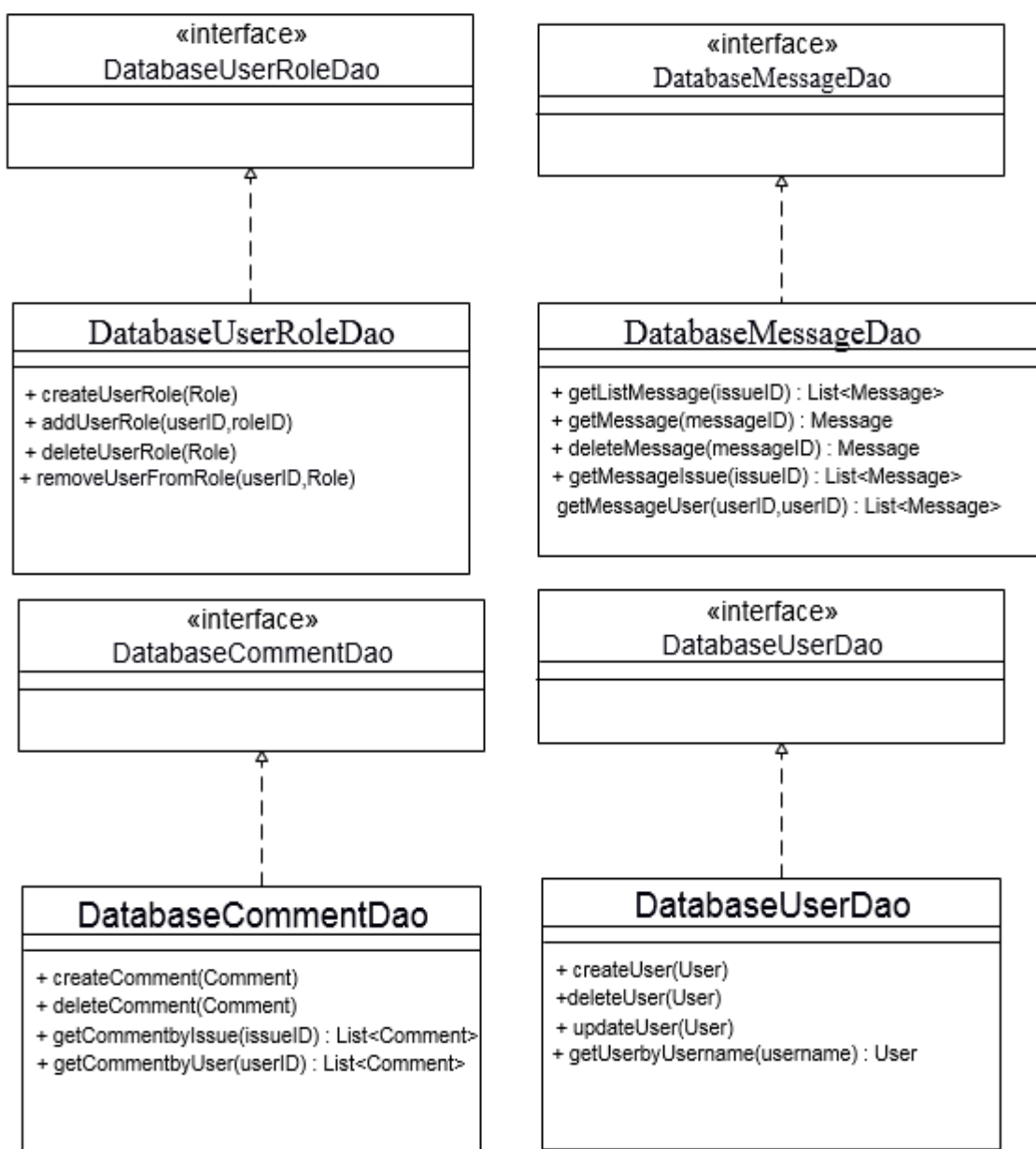


Рисунок 3.1 – Диаграмма классов доступа к данным

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 примет экземпляра класса *Status* и удаляет статус задачи).

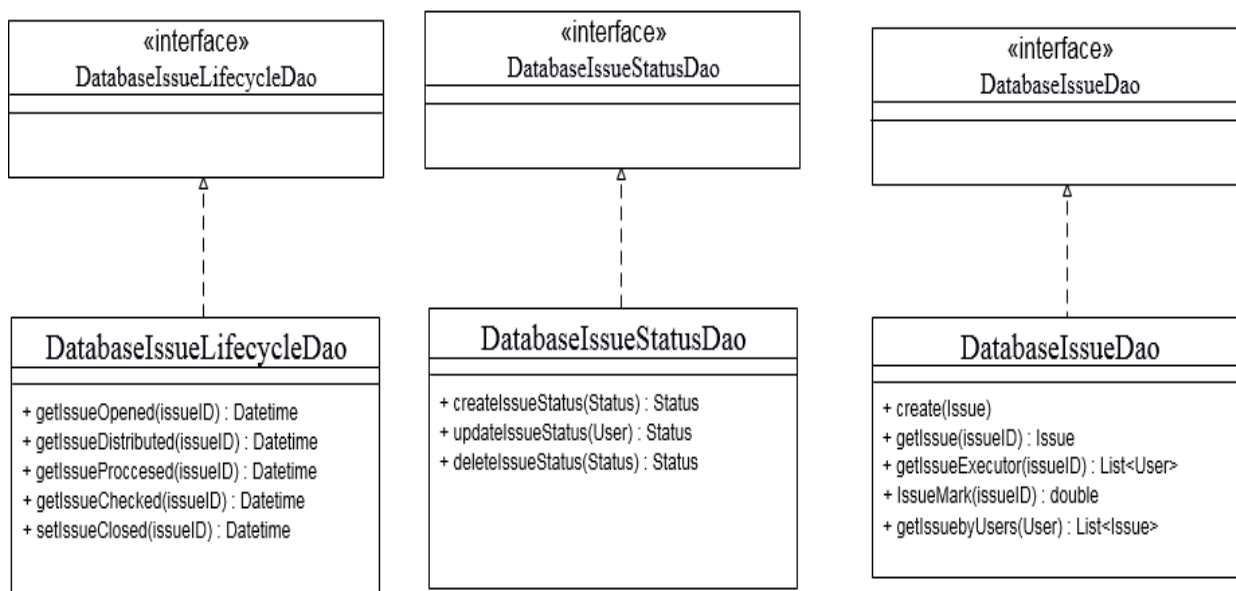


Рисунок 3.2 – Диаграмма классов DatabaseIssueDao, DatabaseIssueStatusDao, DatabaseIssueLifecycleDao

Класс *DatabaseCommentDao* (см. рис. 3.1) предоставляет интерфейс по управлению комментариями над задачами. Класс реализует следующие методы: *createComment* (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>
```

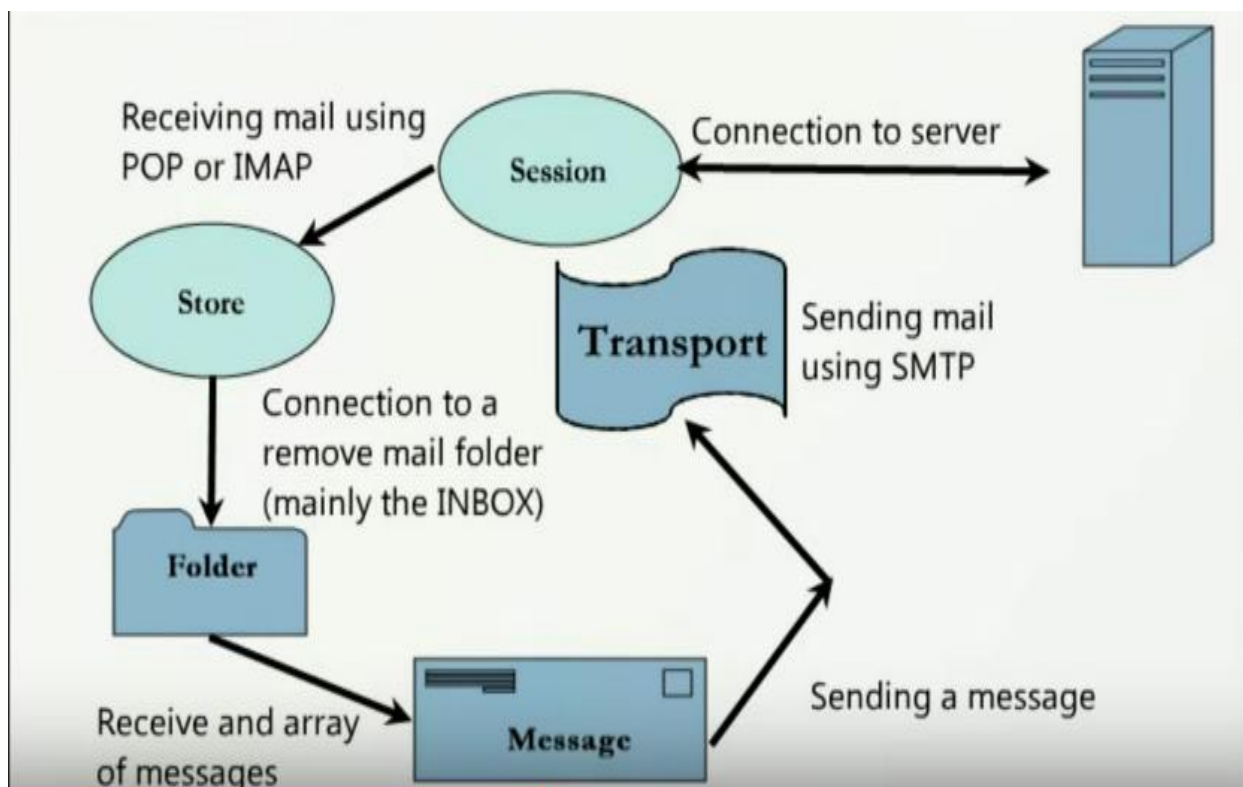


Рисунок 3.3 – Схема работы JavaMail

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, является 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.

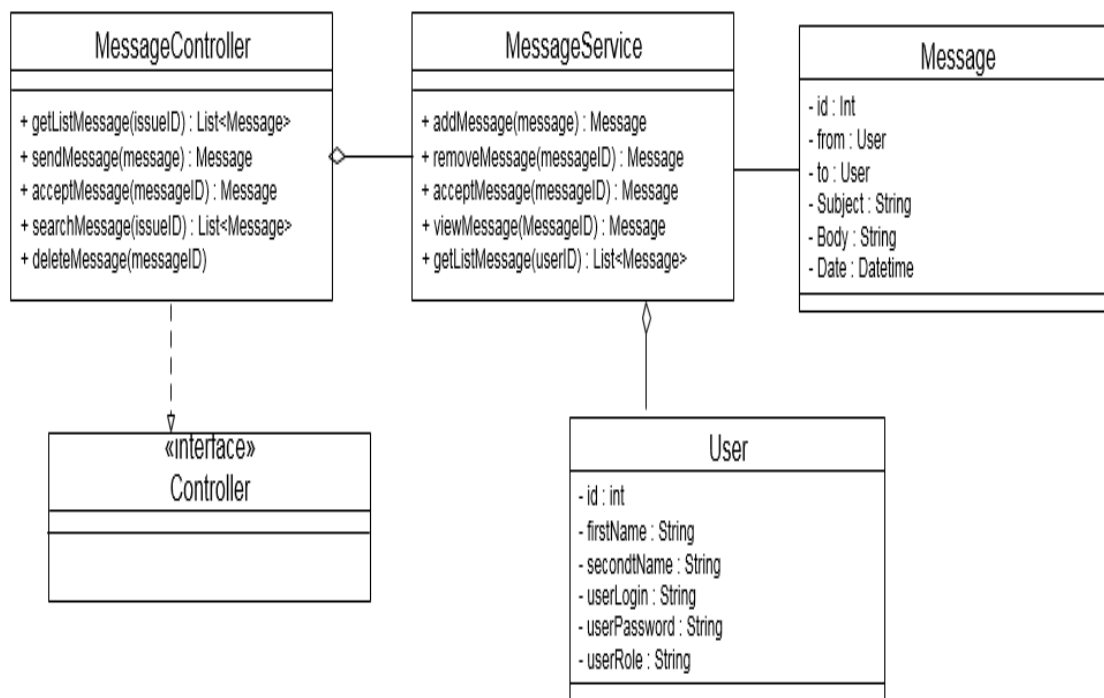


Рисунок 3.4 – Диаграмма классов модуля отправки уведомлений

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 включает 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 в 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.

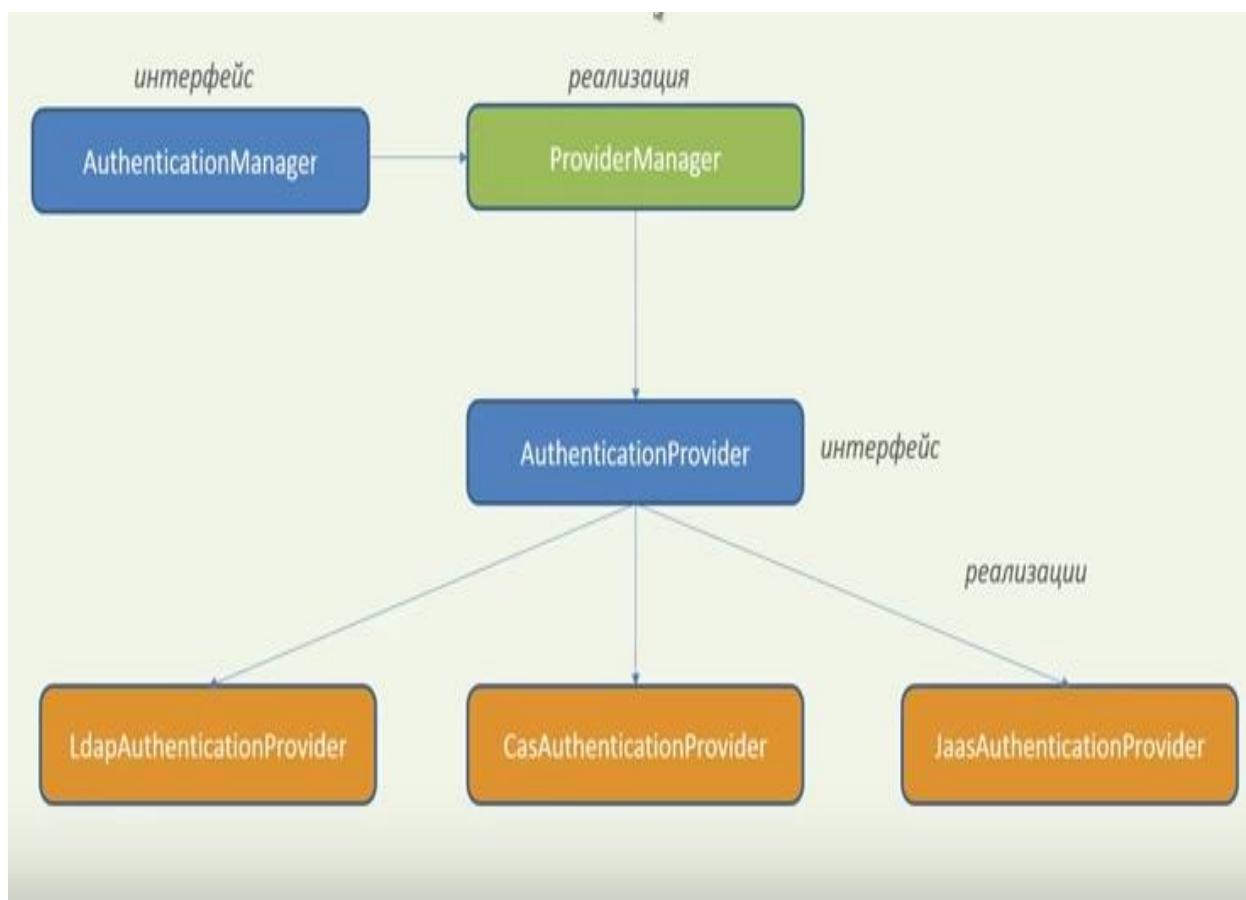


Рисунок 3.5 – Схема работы аутентификации Spring Security

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: `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 *updateUser* 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.

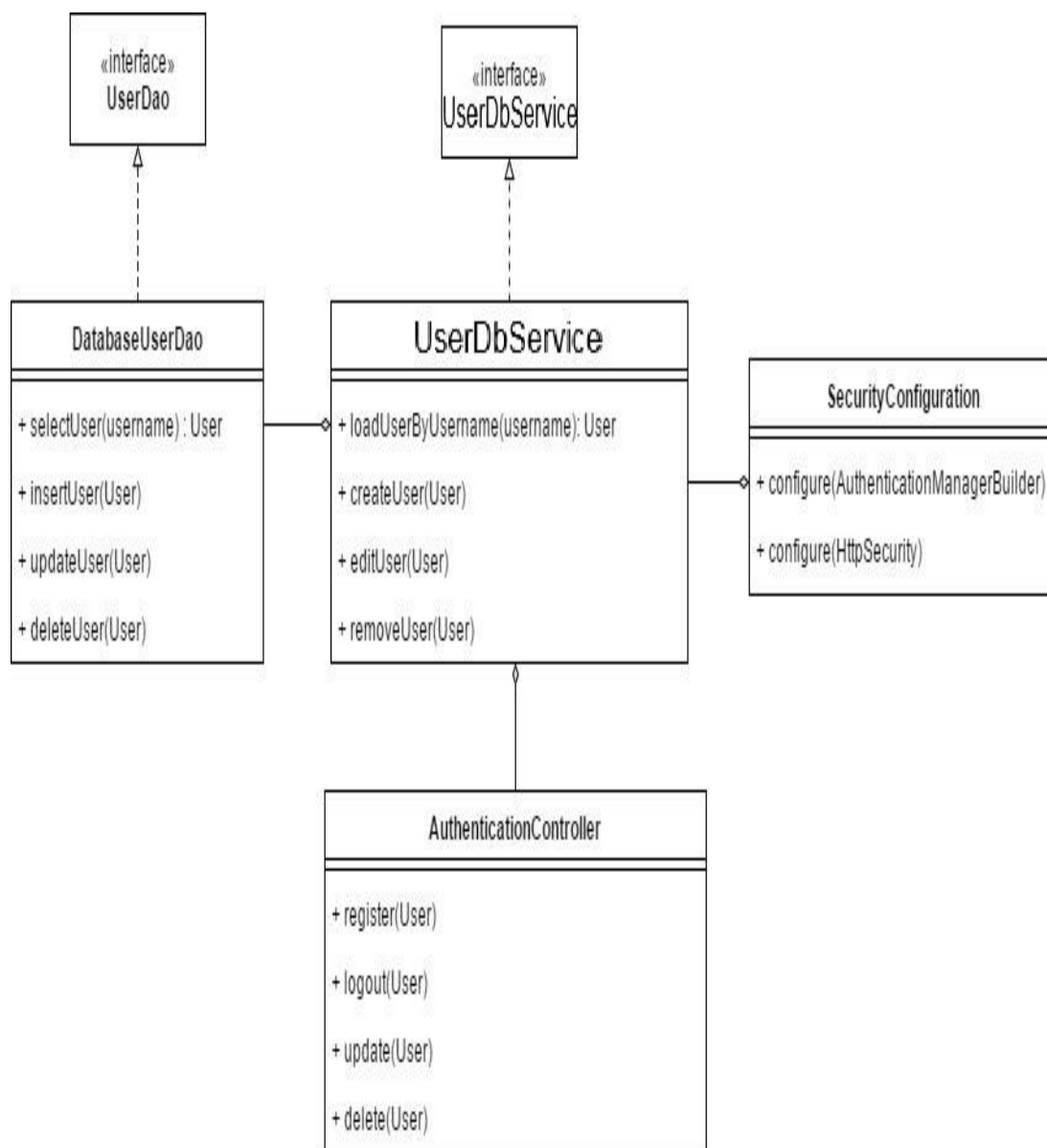


Рисунок 3.6 – Диаграмма классов модуля аутентификации

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 *removeComment* принимает идентификатор комментариев 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.

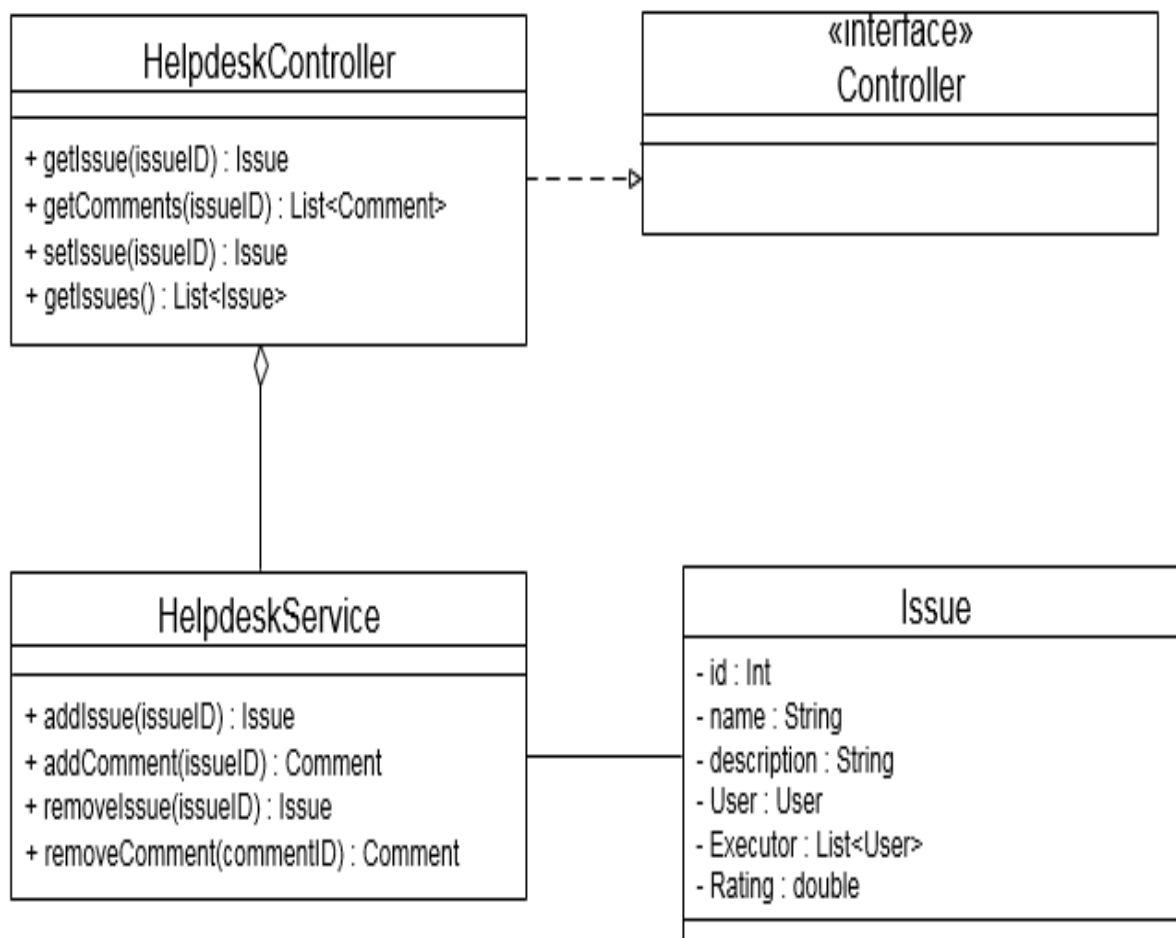


Рисунок 3.7 – Диаграмма классов модуля авторизированной системы учёта заявок от клиентов

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 возвращает 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: *createComment* (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 и 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 *removeComment* принимает идентификатор комментариев 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

возвращает 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.

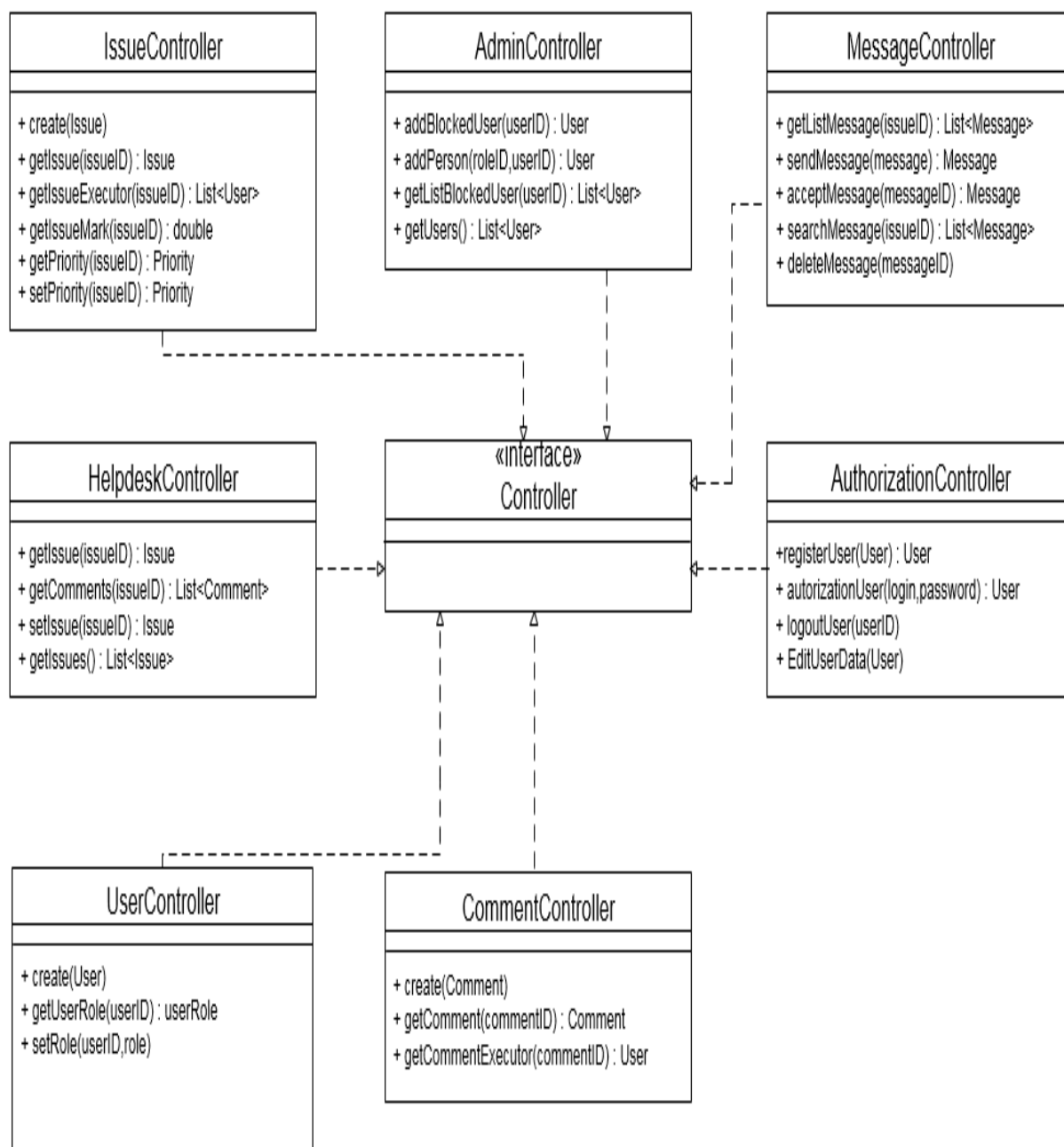


Рисунок 3.8 – Диаграмма классов модуля API веб-сервиса

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 аннотацией 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 помощью 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 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 возвращает 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 соответствующие 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).

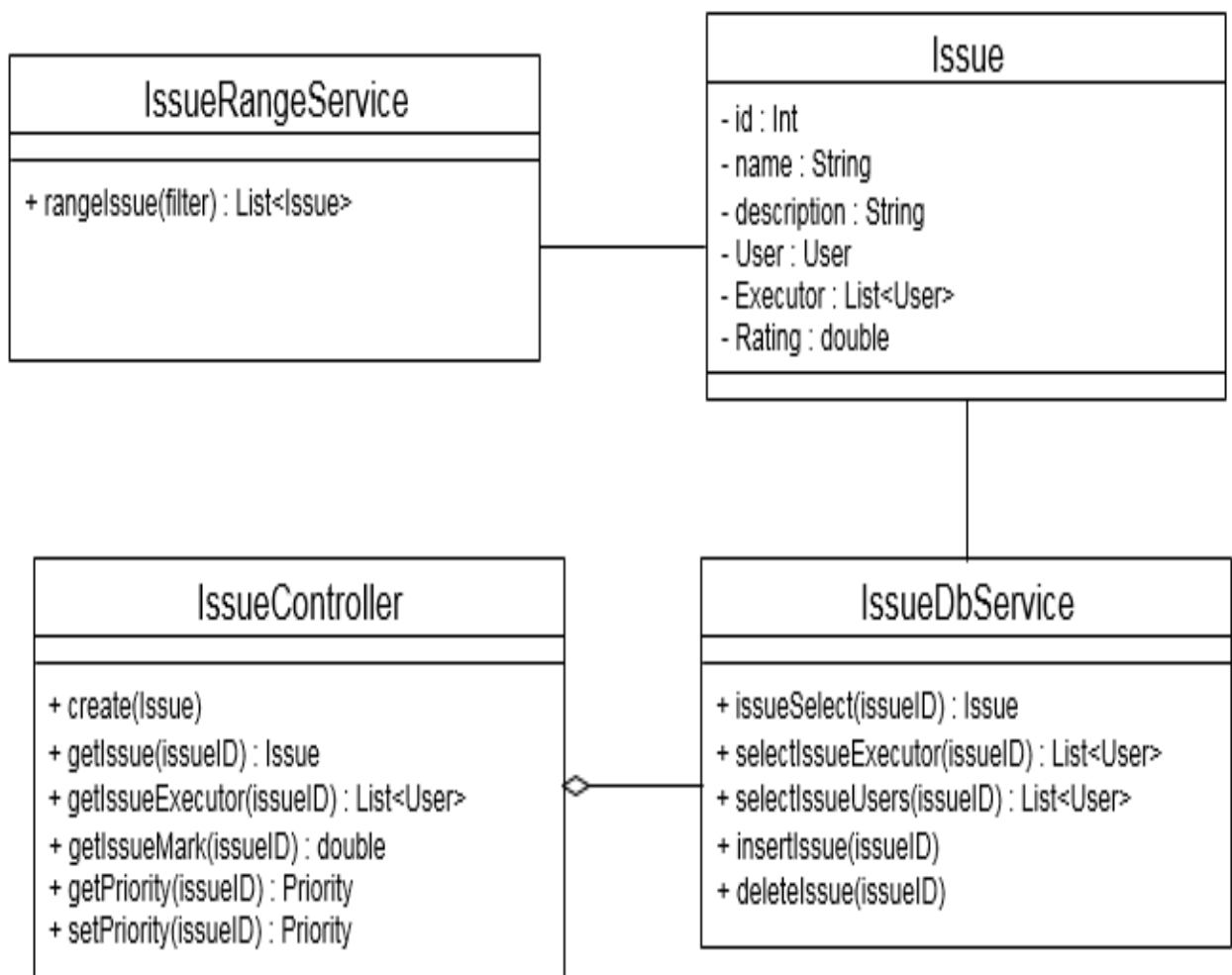


Рисунок 3.9 – Диаграмма классов модуля ранжирования проблем и предложений

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.

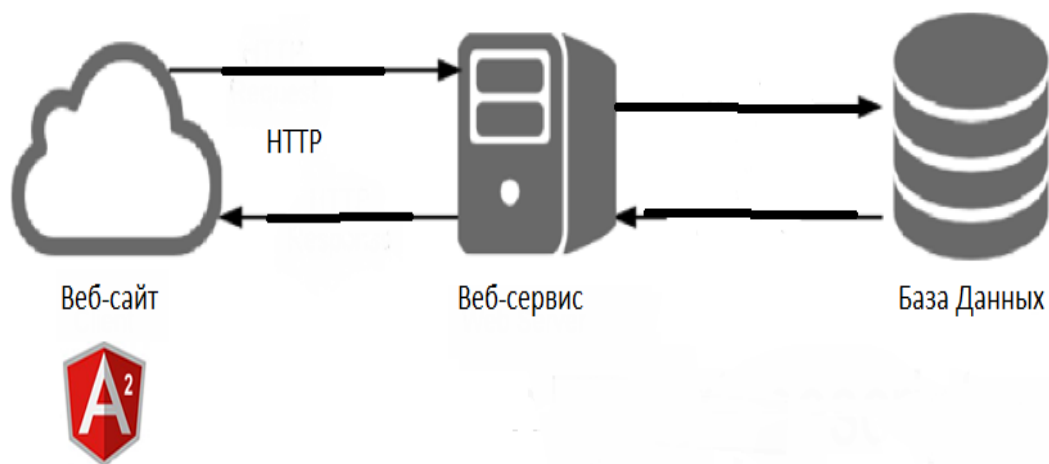


Рисунок 3.10 – Схема взаимодействия в веб-сервисом

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 скобок 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.

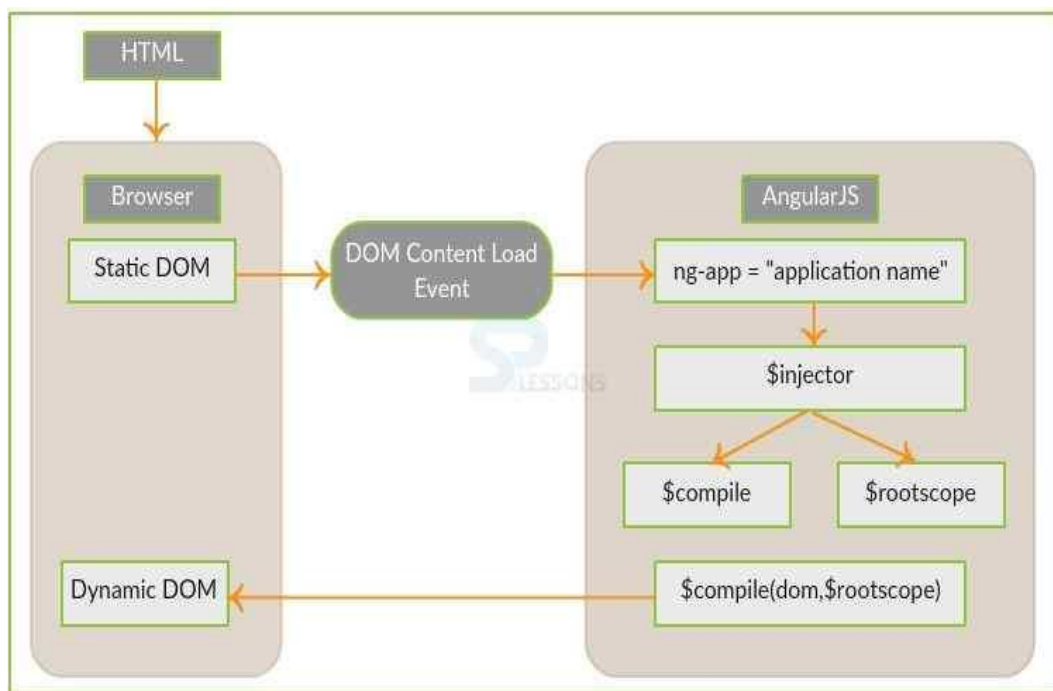


Рисунок 3.11 – Рендеринг страниц в Angular

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.

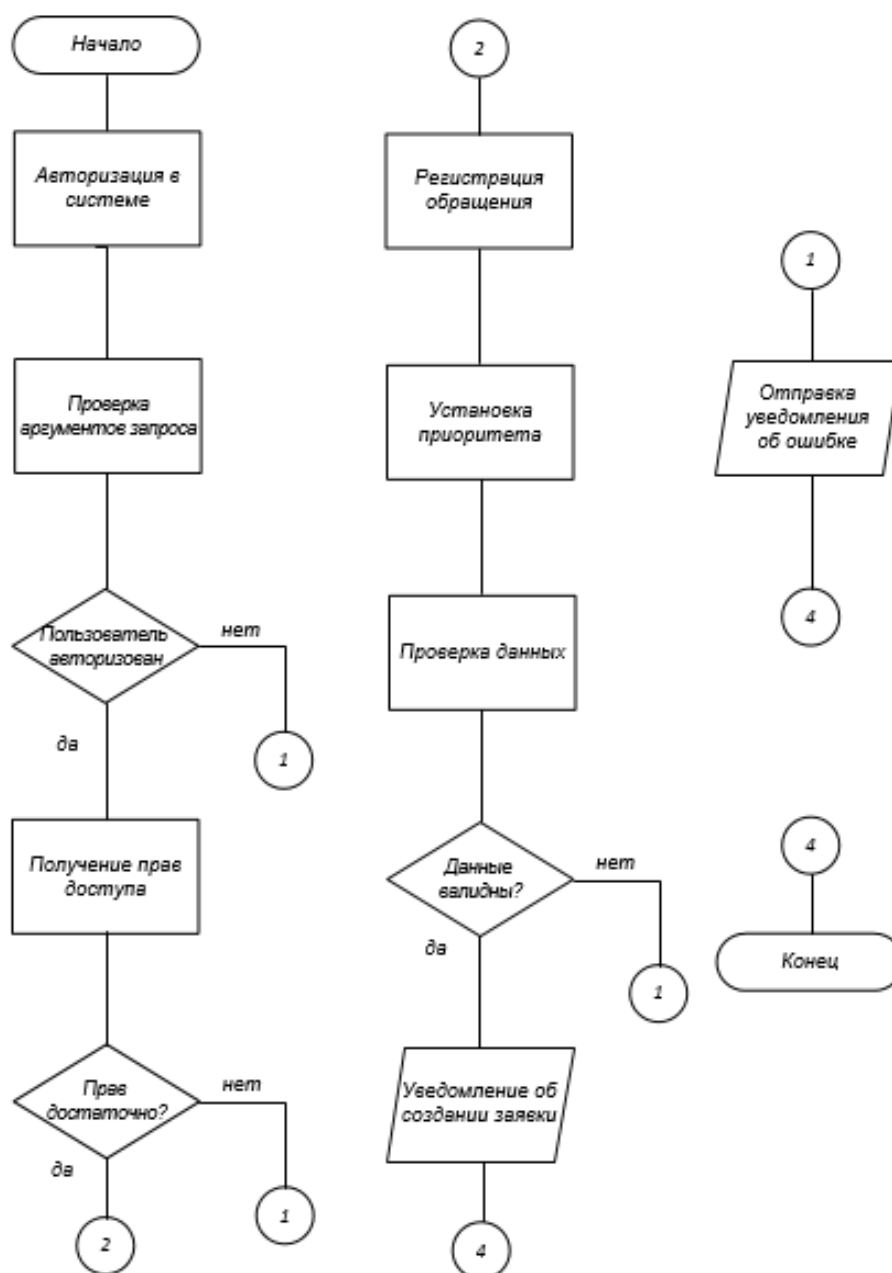


Рисунок 4.1 – Алгоритм создания заявки

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.

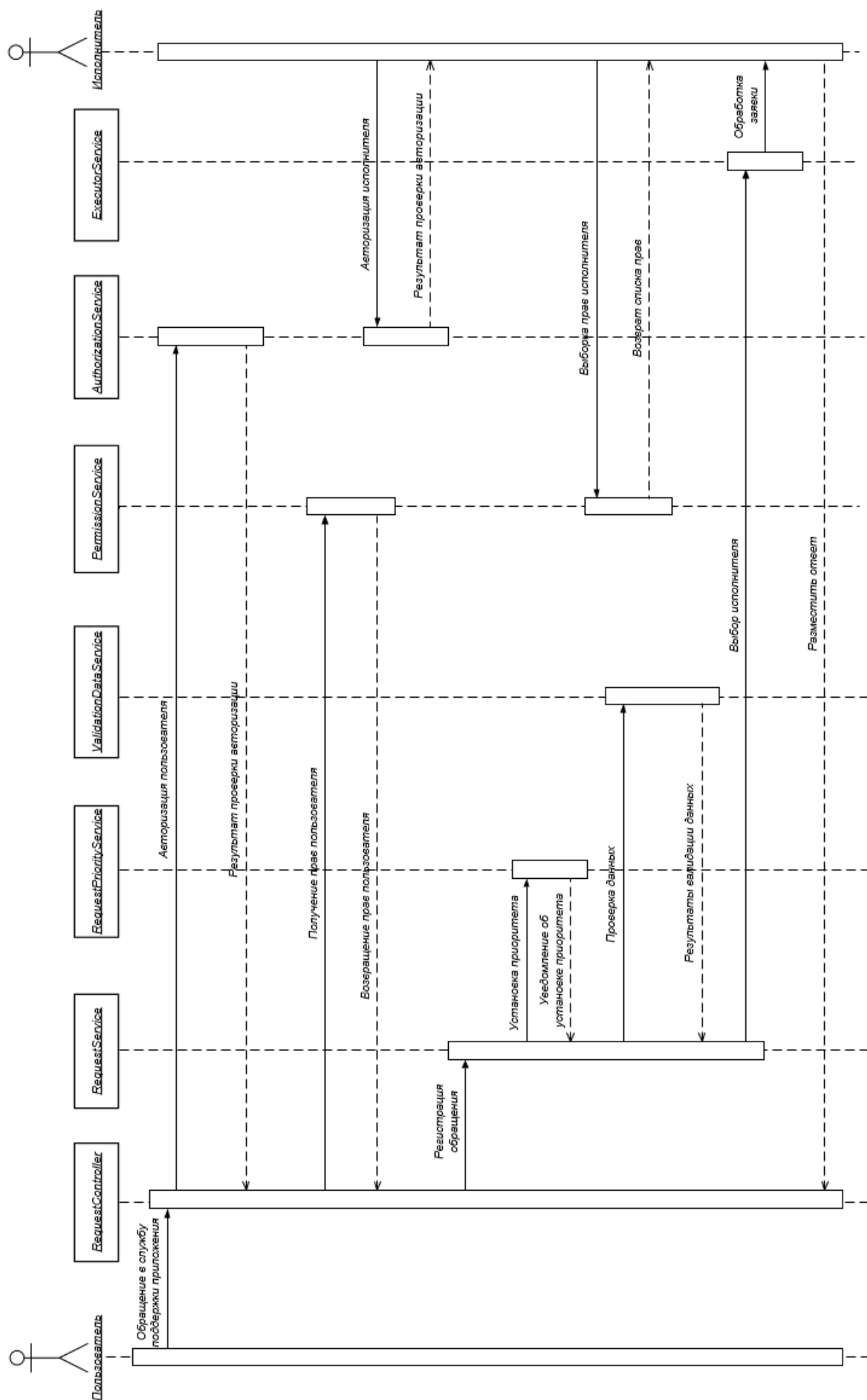


Рисунок 4.2 — Диаграмма последовательности обработки запроса

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 параметризовать 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 отмечены 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 отправляется 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.

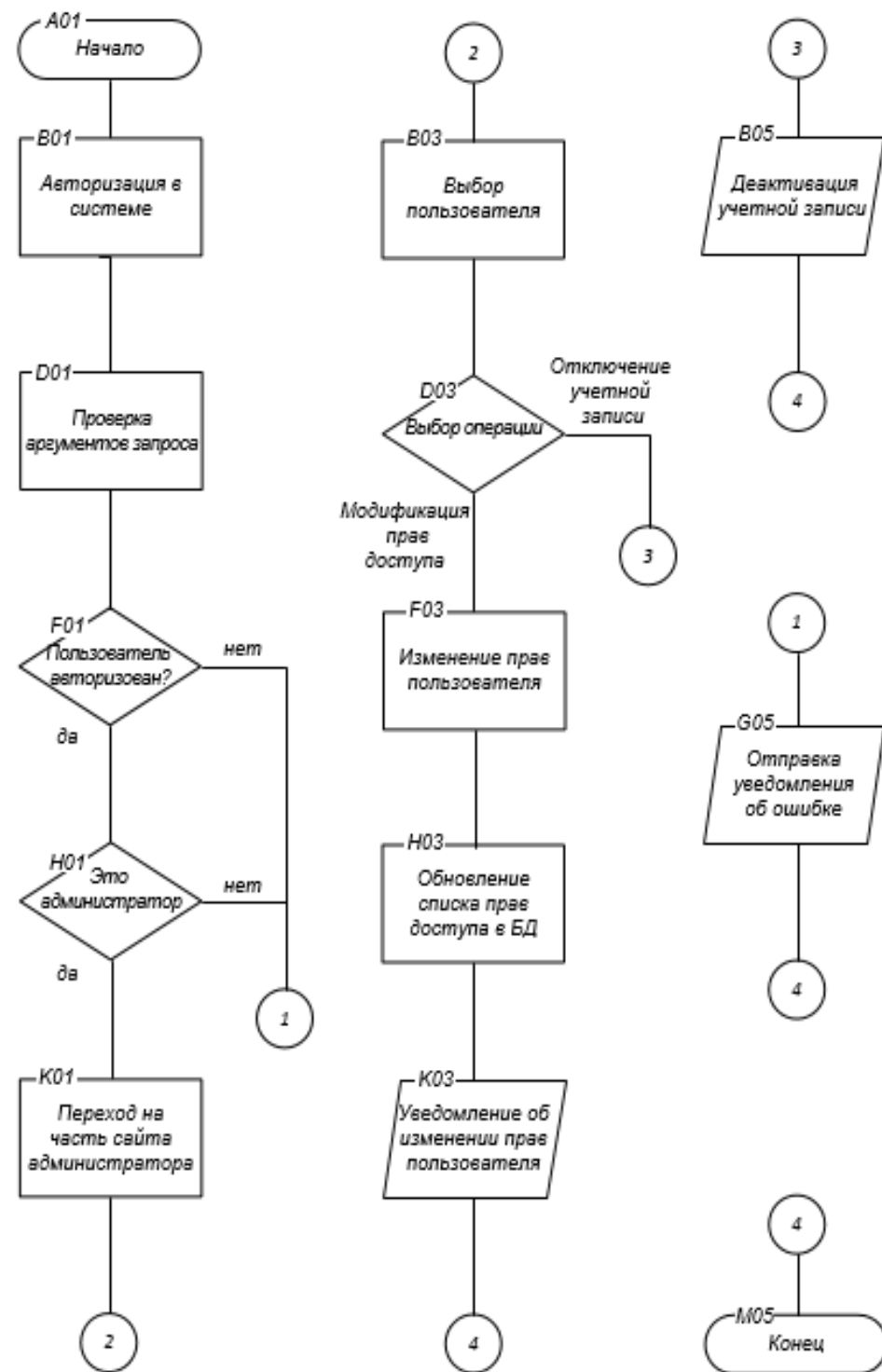


Рисунок 4.4 – Диаграмма активности процесса смены статуса задачи

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.user
name, 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 заявка 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.

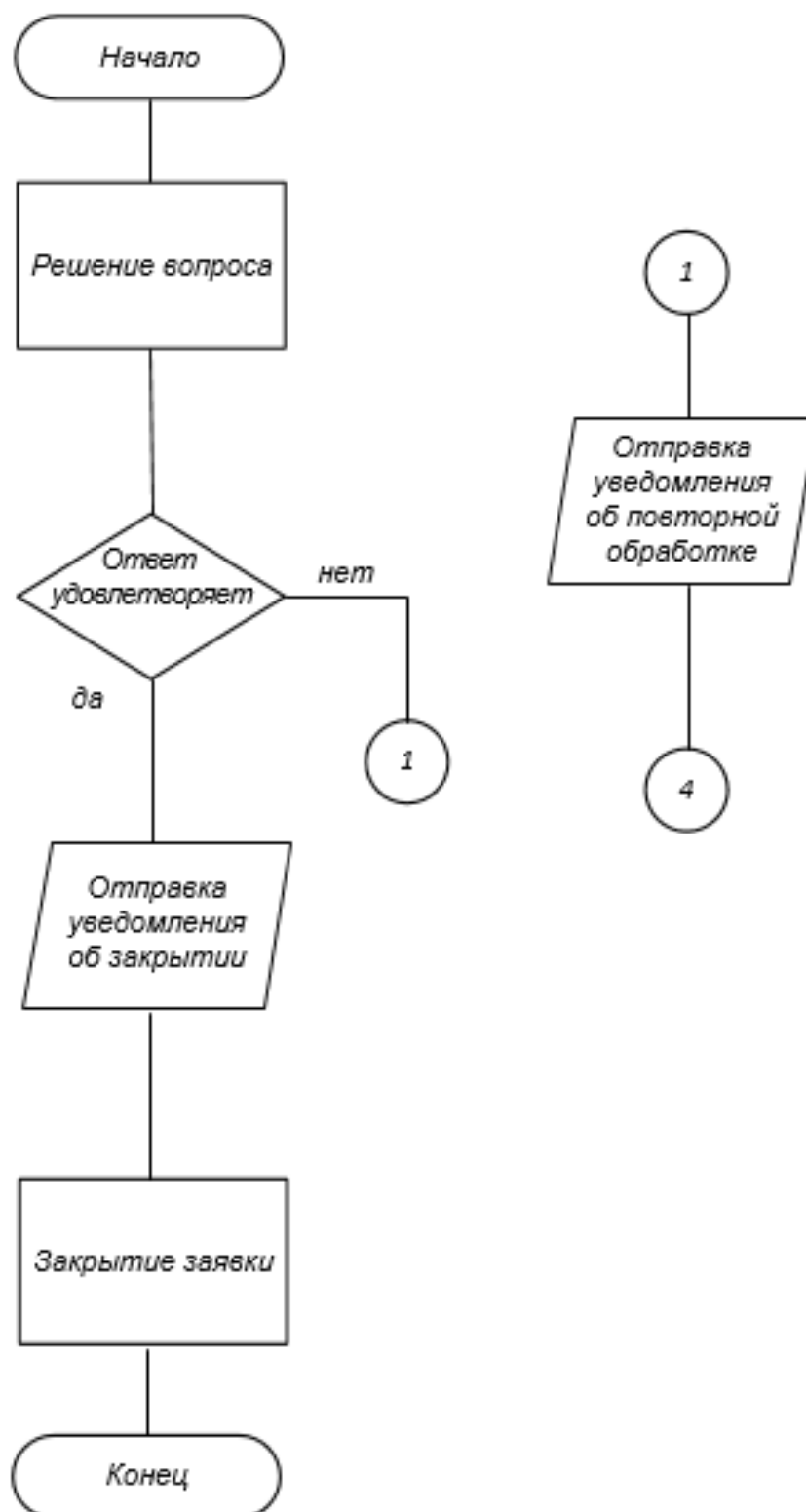


Рисунок 4.5 – Алгоритм закрытия задачи

Ниже приведен фрагмент кода, показывающий, что заявка 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.issueSta
tus, 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.

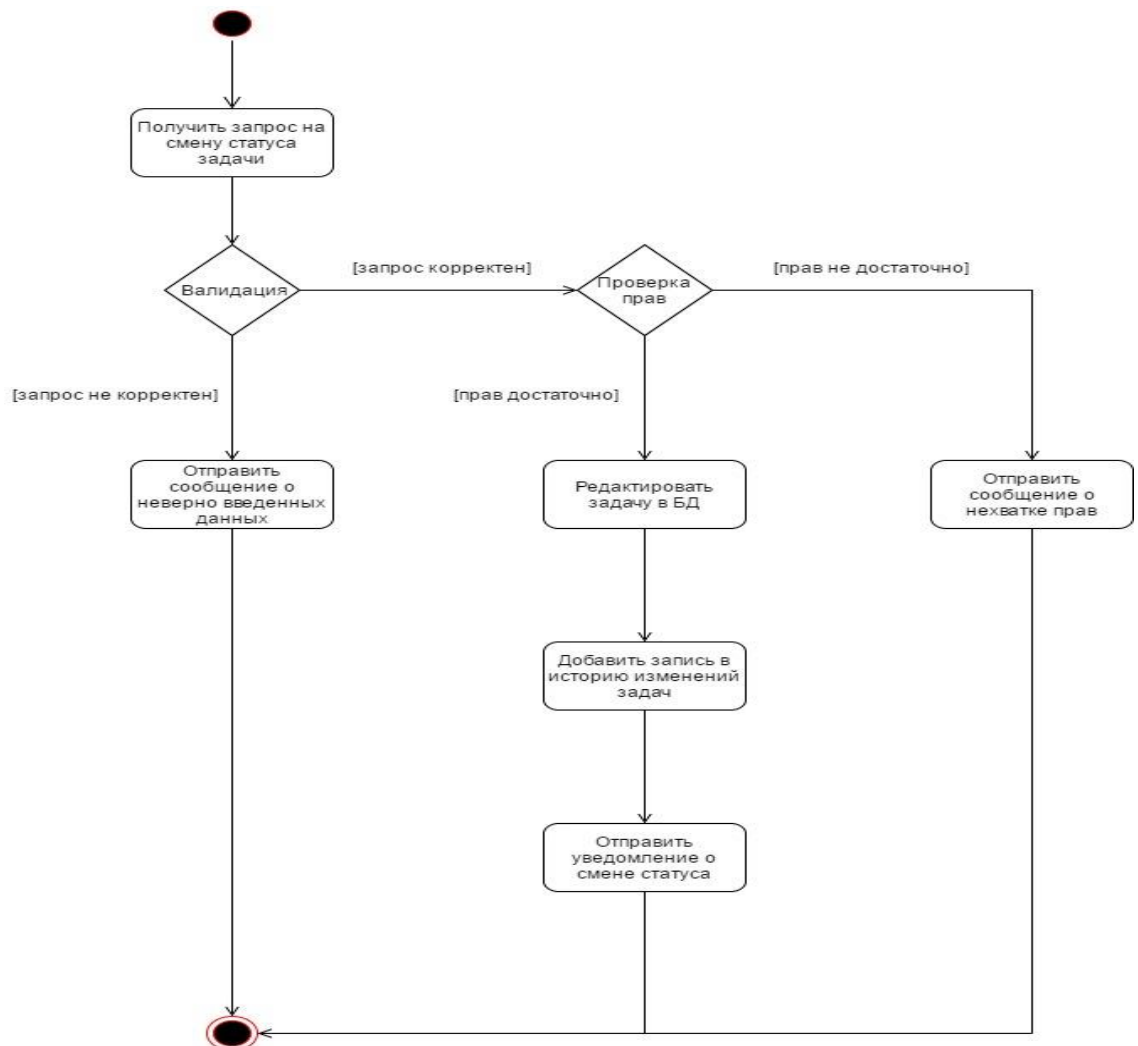


Рисунок 4.6 – Диаграмма активности процесса смены статуса задачи

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.

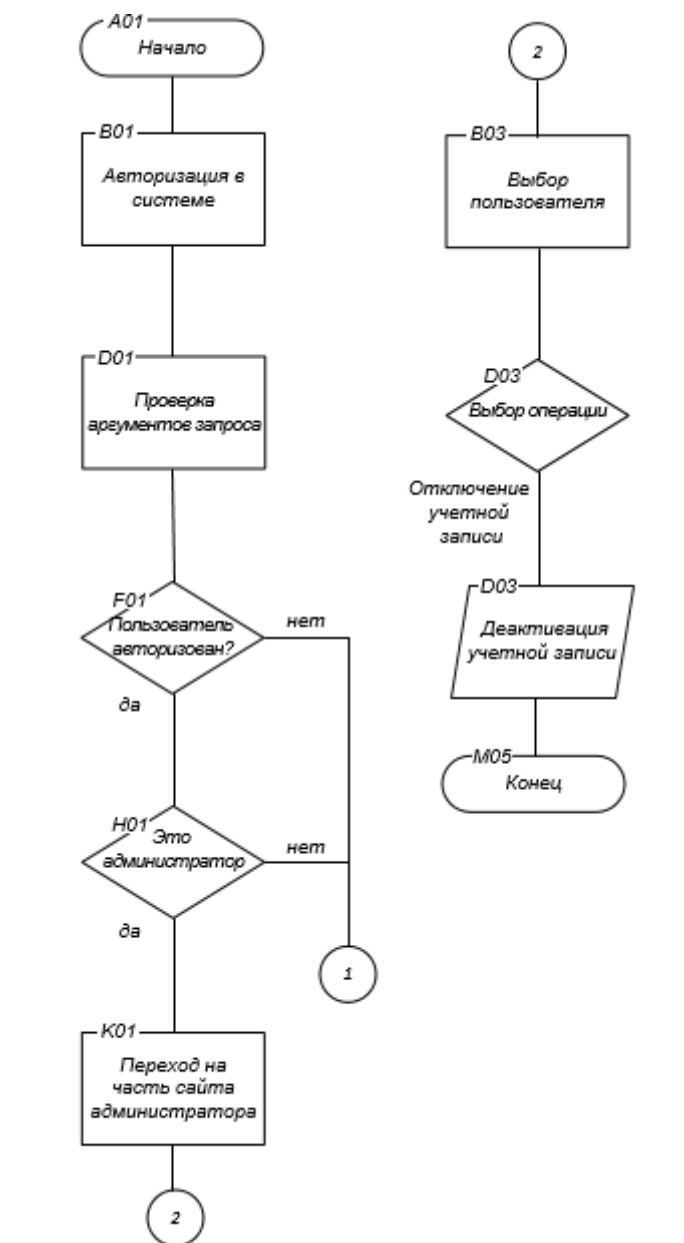


Рисунок 4.7 – Алгоритм блокировки пользователя

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 выявления 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 стабильностью 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 отработывали 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	1 Ввести 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» Ожидаемый 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 «X» 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, позволяющий проверить на корректность отдельные модули исходного кода 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, 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 называемый моук-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 случае 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 случае 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 средства 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.

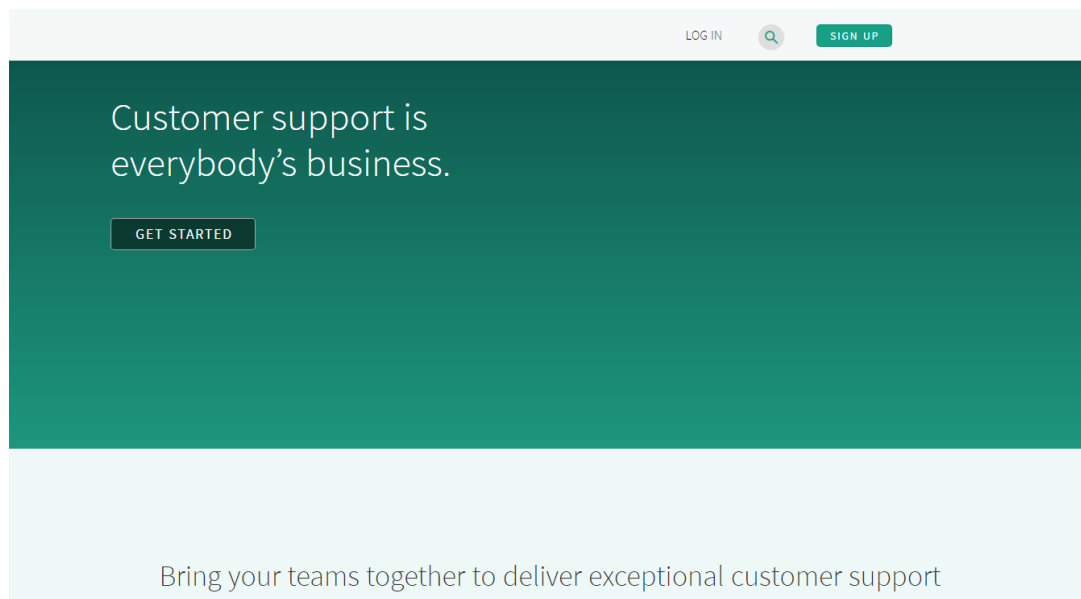


Рисунок 6.1 – Стартовая страница

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).

The image shows a modal window titled 'Authentication' with a close button (X) in the top right corner. Inside the modal, there are two input fields: 'Login' and 'Password'. Below these fields is a blue button labeled 'Log in'.

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.

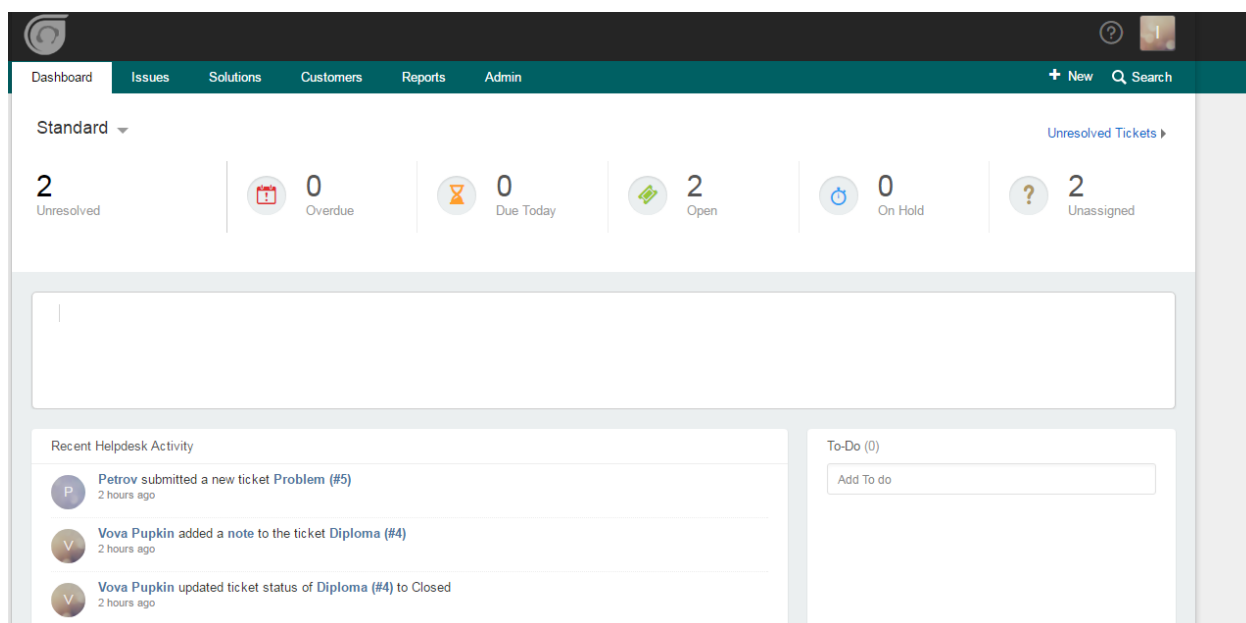


Рисунок 6.3 – Главная страница

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 являются 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.

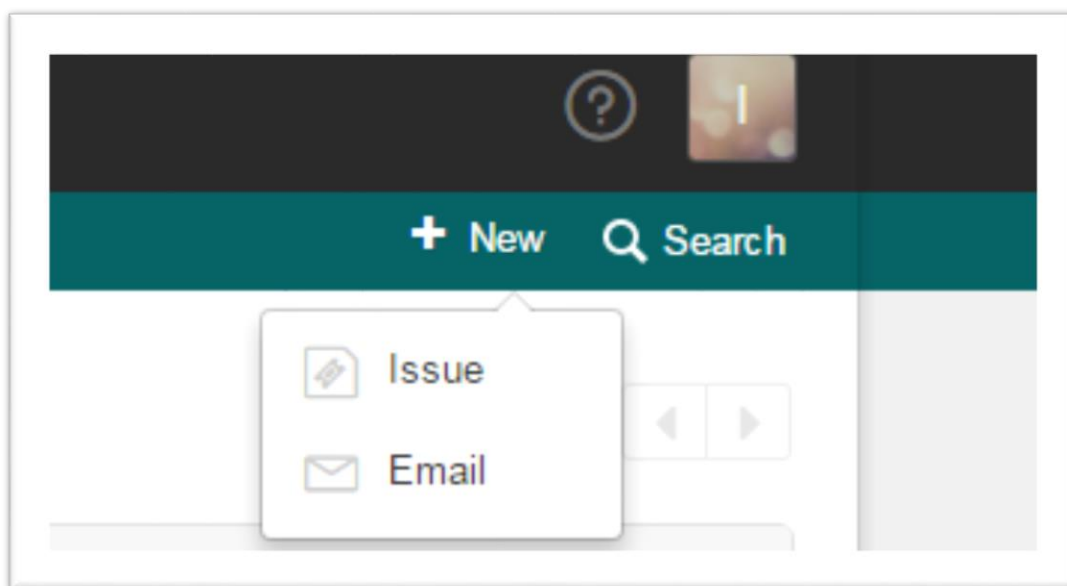


Рисунок 6.4 – Окно выбора новой задачи

Create a new issue | [Select a template ?](#) Cancel Submit

The requester will receive an email notification that a new ticket has been created. It can be disabled in Admin > Email notifications.

Requester ★ [Add new requester](#)

Subject ★

Type

Status ★

Priority ★

Group

Executor

Рисунок 6.5 – Окно создания новой задачи

6.2.4 Окно задач

На рисунке 6.6 представлен вид главного окна проекта.

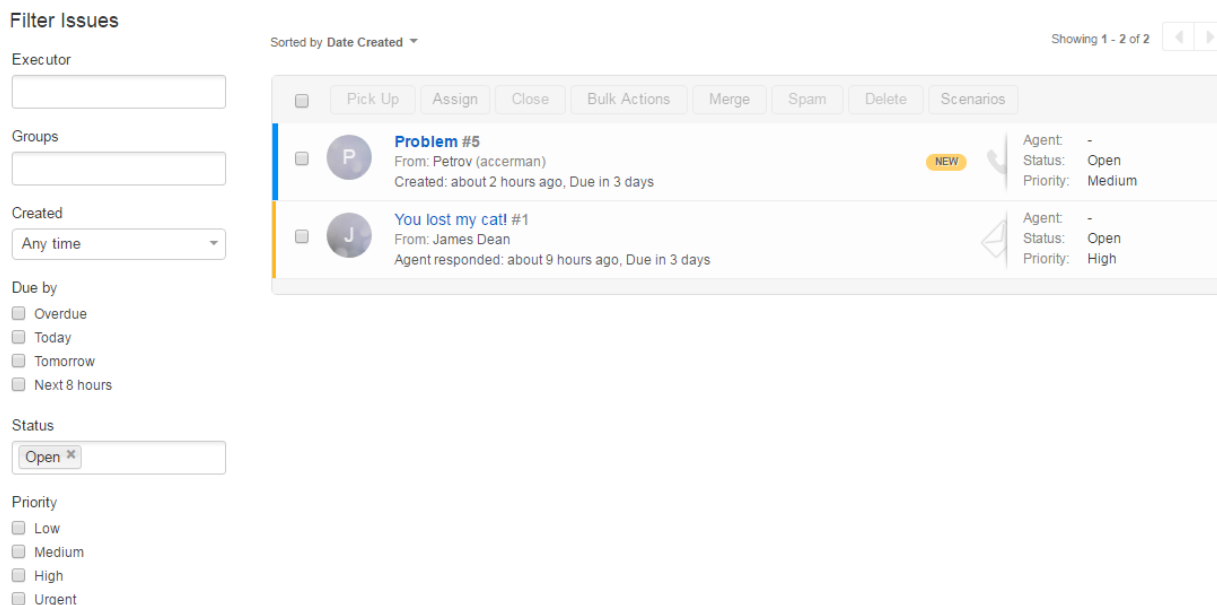


Рисунок 6.6 – Главное окно задач

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.

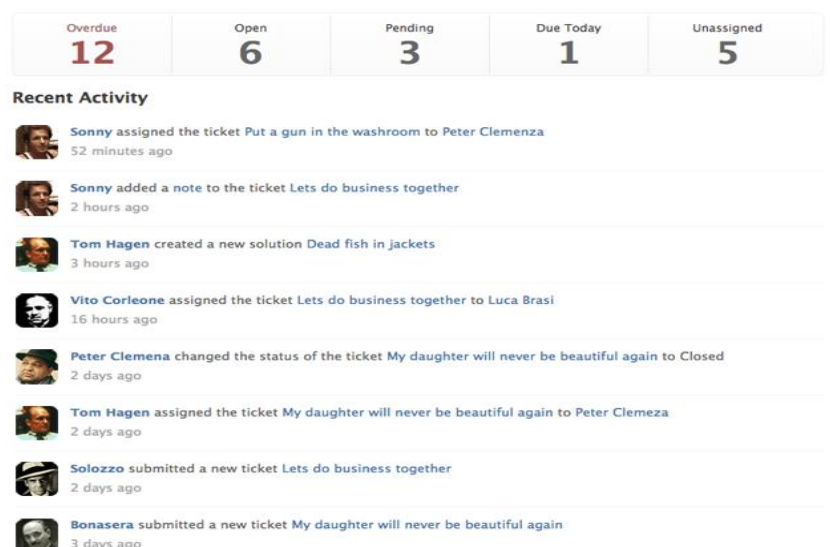


Рисунок 6.7 – Окно активности пользователей

6.2.5 Удаление задачи

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

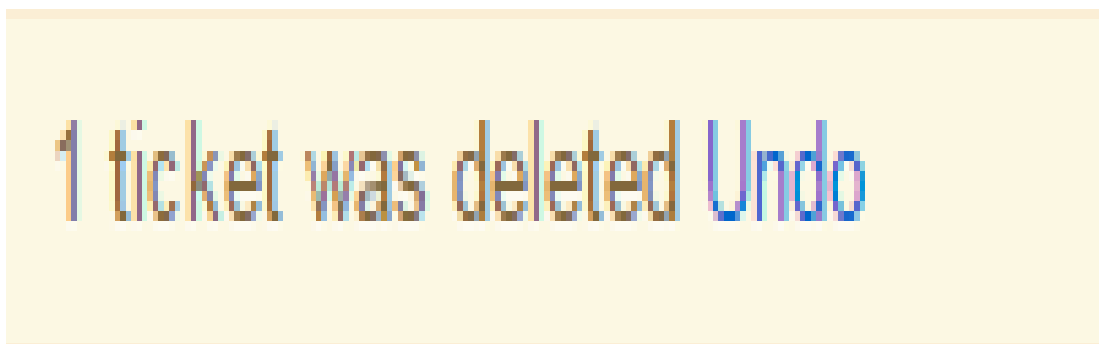


Рисунок 6.8 – Уведомление удаления задачи

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).

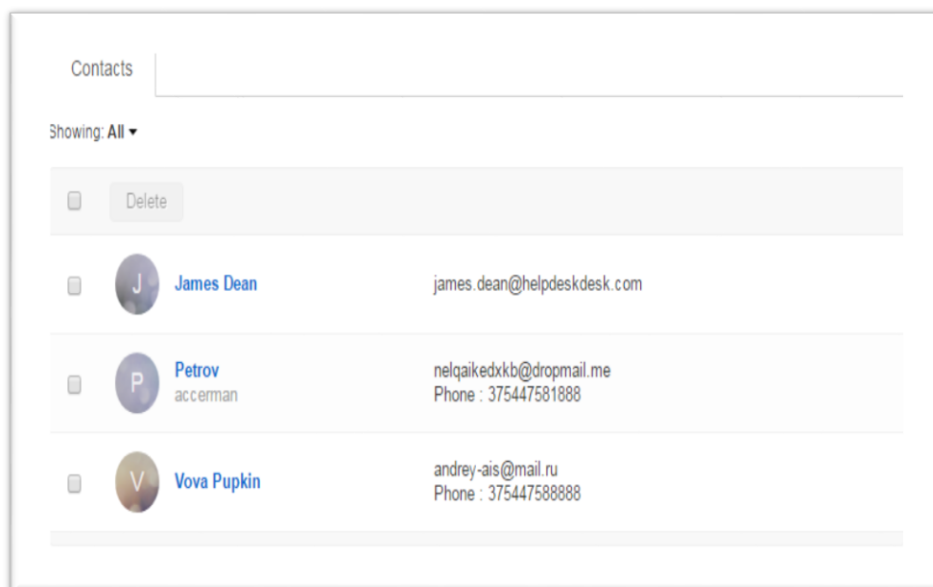


Рисунок 6.9 – Окно пользователей

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).

User Roles		New Role
Account Administrator	Has complete control over the help desk including access to Account or Billing related information, and receives Invoices.	1 Agent
Administrator	Can configure all features through the Admin tab, but is restricted from viewing Account or Billing related information.	No Agent
Supervisor	Can perform all agent related activities, access reports and see unresolved tickets dashboard.	No Agent
Executor	Can log, view, reply, update and resolve tickets and manage contacts.	No Agent

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.

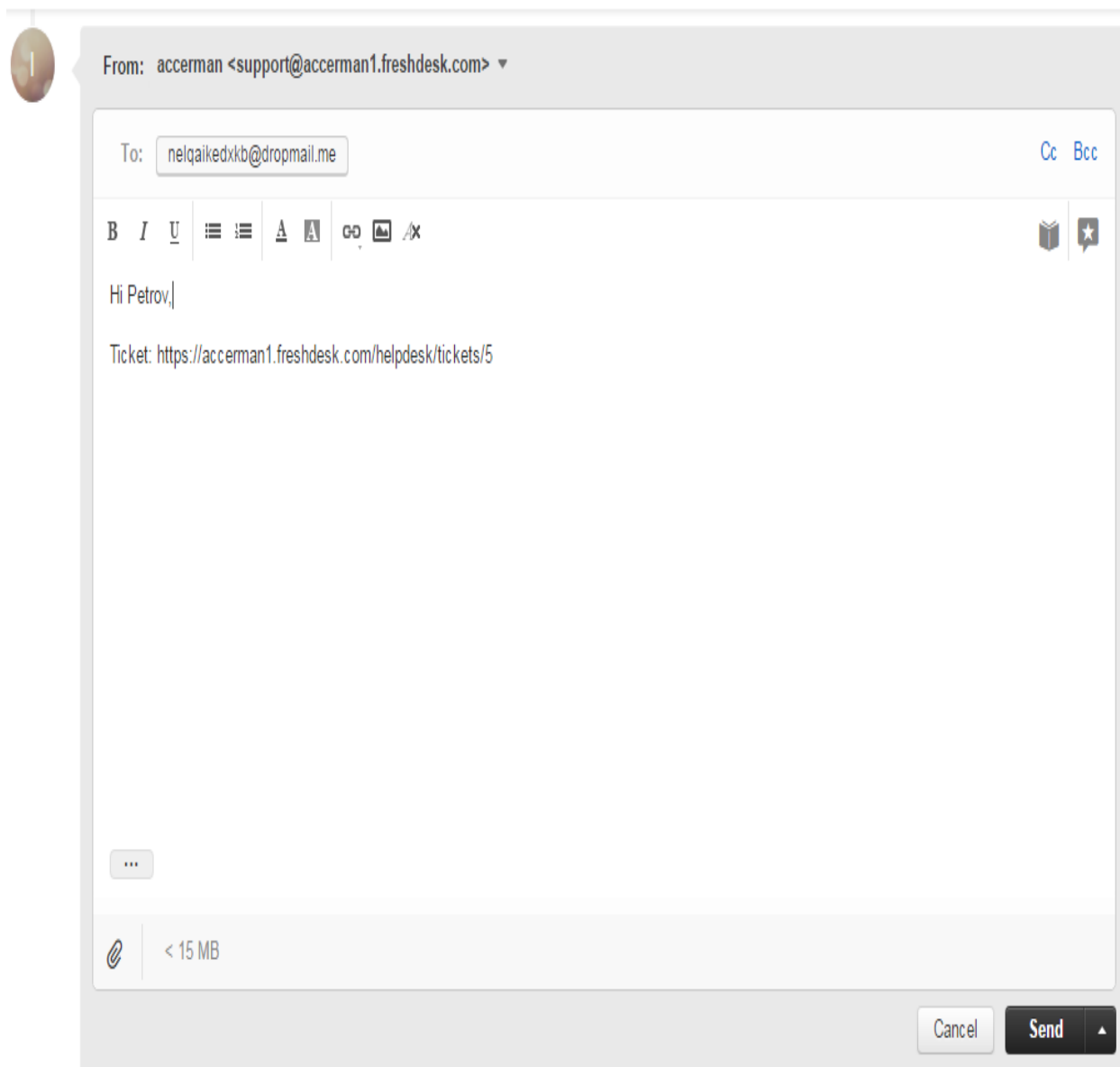


Рисунок 6.11 – Окно добавления комментария

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).

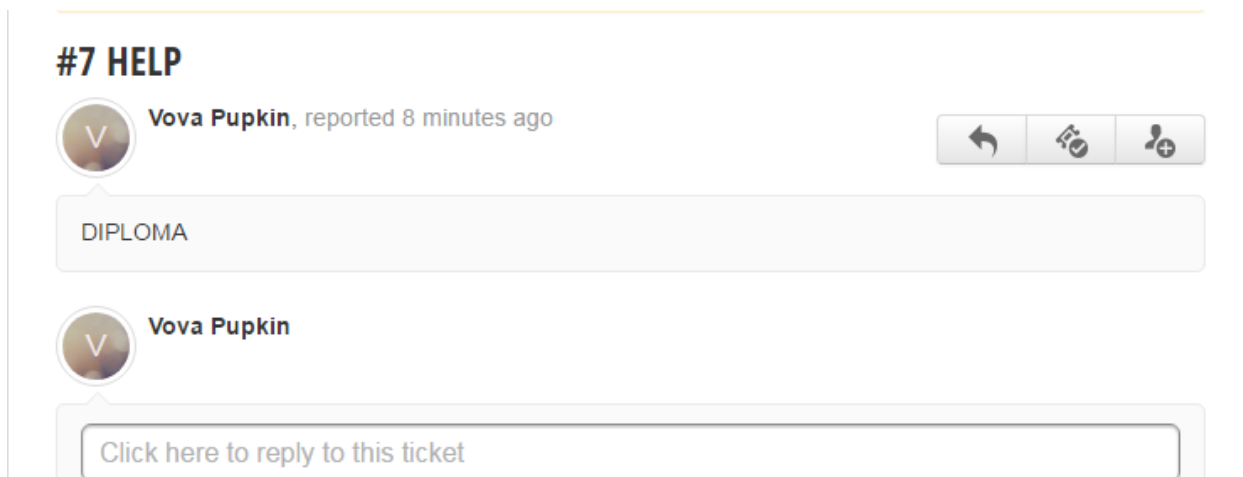


Рисунок 6.12 – Окно закрытия задачи

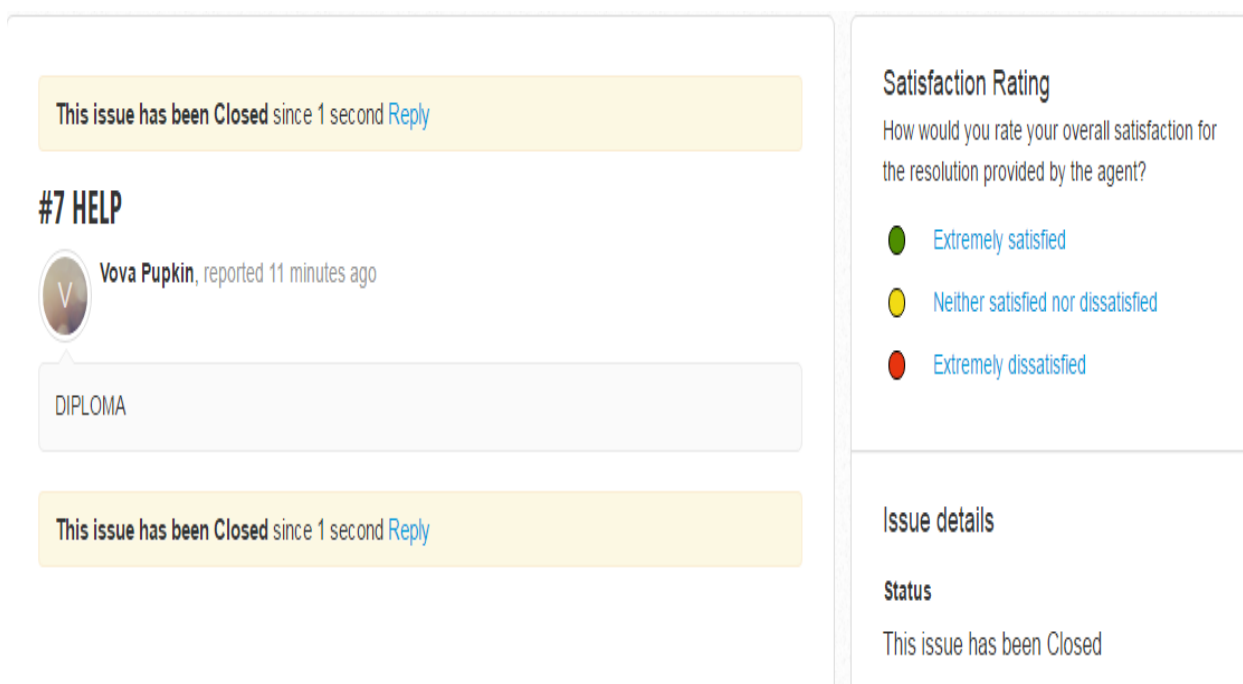


Рисунок 6.13 – Окно уведомления об закрытии задачи

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.

Send an email

Select a template ?

Cancel

Send

From:

To: Type a contact Cc

Type email subject

B I U ☰ ☷ A A G-D 🖼️ ✂️

★

📎 < 15 MB

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.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО СРЕДСТВА ОТСЛЕЖИВАНИЯ ЗАДАЧ

7.1 Характеристика программного продукта

Целью дипломного проекта является разработка программного средства для обеспечения поддержки клиентов, главной целью которого будет сбор и ранжирование обратной связи пользователей. В данной работе будет реализована как публичная помощь по программному продукту, реализованная в виде helpdesk, так и анонимная помощь, с помощью которой клиент может написать на почту и получить анонимный ответ. Ранжирование обратной связи пользователей заключается в том, что клиенты видят, какие жалобы и предложения и предложения появляются по программному продукту и могут их отмечать, как наиболее актуальные. Предполагаемыми пользователями данного программного продукта являются IT-компании любого размера, которые в своем рабочем процессе используют подобные продукты для обеспечения обратной связи с клиентами.

Разработка данного продукта осуществляется для продажи копиями. Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течении 4 лет; планируется продать 100 лицензий в 2017 году, 200 лицензий в 2018 году, 200 лицензий в 2019 году, 200 лицензий в 2020 году.

Экономическая целесообразность инвестиций в разработку и реализацию представленного программного продукта определяется на основе расчёта следующих показателей:

- чистый дисконтированный доход;
- срок окупаемости инвестиций;
- рентабельность инвестиций;

7.2 Расчёт сметы затрат и отпускной цены программного продукта

Основная заработная плата (Z_o) исполнителей определяется по формуле:

$$Z_o = \sum_{i=1}^n T_{qi} \cdot T_q \cdot \Phi_{\pi i} \cdot K, \quad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПС;

T_{qi} – часовая тарифная ставка i -го исполнителя (руб.);

T_q – количество часов работы в день (ч);

K – коэффициент премирования (1,2).

$\Phi_{\Sigma i}$ – эффективный фонд рабочего времени i -го исполнителя (дней);

Примем тарифную ставку 1-го разряда равной 190,00 рублей. Среднемесячная норма рабочего времени составляет 168 часов.

Часовой тарифный оклад руководителя проекта с 12 разрядом составляет $190 \cdot 3,25 / 168 = 3,68$ рубля.

Часовой тарифный оклад инженера-программиста 10 разряда составляет $190 \cdot 2,48 / 168 = 2,80$ рубля.

Результаты расчёта основной заработной платы исполнителей представлены в таблице 7.1.

Таблица 7.1 — Расчёт основной заработной платы.

Исполнитель	Раз - ряд	Тариф- ный коэффици- циент	Месяч- ная тарифная ставка, руб.	Часовая тарифная ставка, руб.	Плано- вый фонд рабочего времени	Заработ- ная плата, руб.
Руководитель проекта	12	3,25	617,50	3,68	30	883,20
Инженер- программист	10	2,48	471,20	2,80	90	2016,00
Премия (20%)						579,84
Основная заработная плата						3479,04

Дополнительная заработная плата ($З_d$) рассчитывается по формуле:

$$З_d = \frac{З_o \cdot Н_d}{100}, \quad (7.2)$$

где $Н_d$ – норматив дополнительной заработной платы, 15%.

Размер дополнительной заработной платы исполнителей составит:

$$З_d = \frac{3479,04 \cdot 15}{100} = 521,86 \text{ руб.}$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ($P_{\text{соц}}$) определяется в соответствии с действующими законодательными актами по формуле:

$$P_{\text{соц}} = \frac{3_o + 3_d}{100} \cdot N_{\text{соц}}, \quad (7.3)$$

где $N_{\text{соц}}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, $34 + 0,6\%$.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составит:

$$P_{\text{соц}} = \frac{3479,04 + 521,86}{100} \cdot 34,6 = 1384,31 \text{ руб.}$$

Расходы по статье «Машинное время» ($P_{\text{мв}}$) включают оплату машинного времени, необходимого для разработки и отладки ПС и определяется по формуле:

$$P_{\text{мв}} = C_m \cdot T_{\text{ч}} \cdot C_p, \quad (7.4)$$

где C_m – цена одного машино-часа, руб.;

$T_{\text{ч}}$ – количество часов работы в день, ч.;

C_p – длительность проекта, дн.

Стоимость одного машино-часа на предприятии составляет 1,50 рублей. Разработка проекта займёт 90 дней. Определим затраты по статье «Машинное время»:

$$P_{\text{мв}} = 1,50 \cdot 8 \cdot 90 = 1080,00 \text{ руб.}$$

Расходы по статье «Прочие затраты» ($P_{\text{пз}}$) включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяется в процентах к основной заработной плате:

$$P_{\text{пз}} = \frac{3_o \cdot N_{\text{пз}}}{100}, \quad (7.5)$$

где $N_{\text{пз}}$ – норматив прочих затрат в целом по организации, 50% .

$$P_{\text{пз}} = \frac{3479,04 \cdot 50}{100} = 1739,52 \text{ руб.}$$

Общая сумма расходов по всем статьям на ПО ($C_{\text{п}}$) представляет полную

себестоимость ПО:

$$C_{\Pi} = P_{\text{мв}} + Z_o + Z_d + P_{\text{пз}} + P_{\text{соц}}, \quad (7.6)$$

$$C_{\Pi} = 1080,00 + 3479,04 + 521,86 + 1739,52 + 1384,31 = 8204,73 \text{ руб.}$$

Прогнозируемая прибыль рассчитывается по формуле:

$$\Pi_o = \frac{C_{\Pi} \cdot Y_p}{100}, \quad (7.7)$$

где Y_p – уровень рентабельности, 25%.

$$\Pi_o = \frac{8204,73 \cdot 25}{100} = 2051,18 \text{ руб.}$$

Прогнозируемая цена без налогов (цена предприятия C_{Π}) рассчитывается по формуле:

$$\Pi_{\Pi} = C_{\Pi} + \Pi_o, \quad (7.8)$$

$$\Pi_{\Pi} = 8204,73 + 2051,18 = 10255,91 \text{ руб.}$$

Налог на добавленную стоимость (НДС) рассчитывается по формуле:

$$\text{НДС} = \frac{\Pi_{\Pi} \cdot H_{\text{дс}}}{100}, \quad (7.9)$$

где $H_{\text{дс}}$ – ставка налога на добавленную стоимость, 20%.

$$\text{НДС} = \frac{10255,91 \cdot 20}{100} = 2051,18 \text{ руб.}$$

Прогнозируемая отпускная цена ($\Pi_{\text{от}}$) рассчитывается по формуле:

$$\Pi_{\text{от}} = C_{\Pi} + \Pi_o + \text{НДС}, \quad (7.10)$$

$$\Pi_{\text{от}} = 8204,73 + 2051,18 + 2051,18 = 12307,09 \text{ руб.}$$

Все расчёты сметы затрат и отпускной цены можно свести в таблицу 7.2.

Таблица 7.2 – Смета затрат и отпускная цена.

Наименование статей	Условные обозначения	Значение, руб.	Методика расчёта
Основная заработная плата исполнителей	$З_o$	3479,04	$З_o = \sum_{i=1}^n T_{\text{чи}} \cdot T_{\text{ч}} \cdot \Phi_{\text{эi}} \cdot K$
Дополнительная заработная плата исполнителей	$З_d$	521,86	$З_d = \frac{З_o \cdot H_d}{100}$
Отчисления в фонд социальной защиты населения	$P_{\text{соц}}$	1384,31	$P_{\text{соц}} = \frac{З_o + З_d}{100} \cdot H_{\text{соц}}$
Машинное время	$P_{\text{мв}}$	1080,00	$P_{\text{мв}} = \Pi_{\text{м}} \cdot T_{\text{ч}} \cdot H_{\text{соц}}$
Прочие прямые расходы	$P_{\text{пз}}$	1739,52	$P_{\text{пз}} = \frac{З_o \cdot H_{\text{пз}}}{100}$
Полная себестоимость	$C_{\text{п}}$	8204,73	$C_{\text{п}} = P_{\text{мв}} + З_o + З_d + P_{\text{пз}}$
Прогнозируемая прибыль	Π_o	2051,18	$\Pi_o = \frac{C_{\text{п}} \cdot Y_p}{100}$
Прогнозируемая цена без налогов	$\Pi_{\text{п}}$	10255,91	$\Pi_{\text{п}} = C_{\text{п}} + \Pi_o$
Налог на добавленную стоимость (НДС)	НДС	2051,18	$\text{НДС} = \frac{\Pi_{\text{п}} \cdot H_{\text{дс}}}{100}$
Прогнозируемая отпускная цена	$\Pi_{\text{от}}$	12307,09	$\Pi_{\text{от}} = C_{\text{п}} + \Pi_o + \text{НДС}$

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 4 лет; планируется продать

100 лицензий в 2017 году, 200 лицензий в 2018 году, 200 лицензий в 2019 году, 200 лицензий в 2020 году. На основании маркетингового исследования отпускная цена одной копии лицензии составила 50 рублей.

Прибыль от продажи одной лицензии программного продукта определяется по формуле:

$$\Pi_{\text{ед}} = Ц - \text{НДС} - \frac{З_p}{N}, \quad (7.11)$$

где Ц — отпускная цена одной копии лицензии программного продукта;
 НДС — сумма налога на добавленную стоимость;
 N — количество лицензий, которые купят клиенты;
 З_р — сумма расходов на разработку и реализацию.
 Сумма налога на добавленную стоимость рассчитывается по формуле:

$$\text{НДС} = \frac{Ц \cdot H_{\text{дс}}}{100 + H_{\text{дс}}}, \quad (7.12)$$

где H_{дс} — ставка налога на добавленную стоимость, равняется 20 %.
 Рассчитаем сумму налога на добавленную стоимость:

$$\text{НДС} = \frac{50 \cdot 20}{100 + 20} = 8,34 \text{ руб.}$$

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

$$З_p = C_{\text{п}} + \frac{C_{\text{п}} \cdot 15}{100} = 8204,73 + \frac{8204,73 \cdot 15}{100} = 9435,44 \text{ руб.}$$

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

$$\Pi_{\text{ед}} = 50 - 8,34 - \frac{9435,44}{700} = 31,73 \text{ руб.}$$

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

$$\text{ЧП}_{\text{ед}} = \Pi_{\text{ед}} \cdot \left(1 - \frac{H_{\text{п}}}{100}\right), \quad (7.13)$$

где H_n — ставка налога на прибыль, 18%.

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

$$ЧП_{ед} = 31,73 \cdot \left(1 - \frac{18}{100}\right) = 26,02 \text{ руб.}$$

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

$$ЧП = ЧП_{ед} \cdot N, \quad (7.14)$$

Прибыль по проекту за каждый год продаж составляет:

$$ЧП_1 = 26,02 \cdot 100 = 2602 \text{ руб.}$$

$$ЧП_2 = 26,02 \cdot 200 = 5204 \text{ руб.}$$

$$ЧП_3 = 26,02 \cdot 200 = 5204 \text{ руб.}$$

$$ЧП_4 = 26,02 \cdot 200 = 5204 \text{ руб.}$$

7.4 Расчет показателей экономической эффективности разработки и реализации 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:

$$\alpha = (1 + E_n)^{t-t_p}, \quad (7.15)$$

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

t_p — расчетный год, $t_p = 1$;

t — порядковый номер года.

На 01.05.2017 г. ставка рефинансирования составляет 15%. Используя формулу (7.15) рассчитаем коэффициенты дисконтирования:

$$2017 \text{ г.}; \quad t_p = 1; \quad \alpha = (1 + 0,15)^{1-1} = 1$$

$$2018 \text{ г.}; \quad t_p = 2; \quad \alpha = (1 + 0,15)^{1-2} = 0,87$$

$$\begin{aligned}
2019 \text{ г.}; \quad t_p &= 3; \quad \alpha = (1 + 0,15)^{1-3} = 0,76 \\
2020 \text{ г.}; \quad t_p &= 4; \quad \alpha = (1 + 0,15)^{1-4} = 0,65
\end{aligned}$$

Расчет показателей эффективности инвестиций по разработке продукта представлен в таблице 7.3.

Таблица 7.3 — Результаты расчета эффективности инвестиционного проекта

Показатели	Ед. изм.	Годы			
		2017	2018	2019	2020
Коэффициент приведения		1	0,87	0,76	0,65
Прирост чистой прибыли	руб.	2602	5204	5204	5204
Прирост чистой прибыли с учётом фактора времени	руб.	2602	4527,48	3955,04	3382,60
Затраты на разработку и реализацию (Z_p)	руб.	9435,44	-	-	-
Затраты на разработку и реализацию с учётом фактора времени	руб.	9435,44	-	-	-
Превышение результата над затратами	руб.	-6833,44	-1629,44	3574,56	8778,56
То же с нарастающим итогом	руб.	-6833,44	-2305,96	1649,08	5031,68

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта ($P_{и}$) по формуле:

$$P_{и} = \frac{\Pi_{\text{чср}}}{Z_p} \cdot 100\%, \quad (7.16)$$

где $\Pi_{\text{чср}}$ – среднегодовая величина чистой прибыли за расчетный период, руб., которая определяется по формуле:

$$\Pi_{\text{ср}} = \frac{\sum_{i=1}^n \Pi_{\text{ч}i}}{n}, \quad (7.17)$$

где $\Pi_{\text{ч}t}$ – чистая прибыль, полученная в году t , руб.

$$\Pi_{\text{ср}} = \frac{2602 + 5204 + 5204 + 5204}{4} = 4553,20 \text{ руб.}$$

$$P_{\text{и}} = \frac{4553,20}{9435,44} \cdot 100 = 48,2\%$$

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

- чистый дисконтированный доход за четыре года работы программы составит 5031,68 руб.;
- затраты на разработку программного продукта окупятся на третий год его использования;
- рентабельность инвестиций составит 48,2%.

Таким образом, разработка и реализация программного продукта является эффективным и инвестирование средств в разработку продукта целесообразно.

ЗАКЛЮЧЕНИЕ

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

- обработка задач с помощью авторизированной системы учёта заявок;
- оказание эффективной обратной связи;
- сбор и ранжирование обратной связи;
- создание задач, решений, 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 прямой и обратной связи, модуль ранжирования проблем и предложений, модуль маршрутизации, модуль авторизированной системы учёта заявок, модуль администрирования и авторизации, модуль доступа к данным, база данных веб-сервиса, модуль пользовательского интерфейса, модуль взаимодействия с веб-сервисом, модуль визуализации. 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 обработки запроса пользователя.

Ко всему прочему 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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Что такое Helpdesk [Электронный ресурс]. – 2017. – Режим доступа: <http://www.codenet.com/CFM/helpdesk/>.
- [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] Техничко-экономическое обоснование дипломных проектов. – 2005. – В.А. Палицын.

ПРИЛОЖЕНИЕ А
(обязательное)

Программное средство поддержки клиентов. Исходный код.

Содержимое файла DatabaseIssueDao.java

```
package by.bsuir.helpdesk.dao.impl;

import by.bsuir.helpdesk.dao.IssueDao;
import by.bsuir.helpdesk.domain.Priority;
import by.bsuir.helpdesk.domain.Status;
import by.bsuir.helpdesk.domain.Issue;
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;

@Repository
public class DatabaseIssueDao implements IssueDao {

    private static final String GET_ISSUE_ISSUES_QUERY = "select ta_id, ta_name, ta_summary,
    ta_priority, ta_status, ta_order, ts_order, ts_name from issue inner join issue_status on
    ts_id=ta_status where ta_issue = :issueId and ta_status = :issueStatusId order by ts_order,
    ta_priority, ta_order limit :startFrom, :rowNumber;";

    private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    private static final RowMapper<Issue> issueRowMapper = ((resultSet, i) -> {
        Issue issue = new Issue();

        issue.setId(resultSet.getInt(TableColumn.ISSUE_ID));
        issue.setName(TableColumn.ISSUE_NAME);
        issue.setSummary(TableColumn.ISSUE_SUMMARY);
        issue.setPriority(Priority.valueOf(resultSet.getString(TableColumn.ISSUE_PRIORITY)));
        issue.setOrder(resultSet.getInt(TableColumn.ISSUE_ORDER));

        Status status = new Status();

        status.setName(resultSet.getString(TableColumn.ISSUE_STATUS_NAME));
        status.setOrder(resultSet.getInt(TableColumn.ISSUE_STATUS_ORDER));
        status.setId(resultSet.getInt(TableColumn.ISSUE_STATUS_ID));

        issue.setIssueStatus(status);

        return issue;
    });

    @Autowired
    public DatabaseIssueDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
    }

    @Override
    public List<Issue> getIssueIssues(int issueId, int issueStatusId, int startFrom, int limit) {
        MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();

        mapSqlParameterSource.addValue(ParameterName.ISSUE_ID, issueId);
        mapSqlParameterSource.addValue(ParameterName.ISSUE_STATUS_ID, issueStatusId);
        mapSqlParameterSource.addValue(ParameterName.START_FROM, startFrom);
        mapSqlParameterSource.addValue(ParameterName.ROW_NUMBER, limit);

        return namedParameterJdbcTemplate.query(GET_ISSUE_ISSUES_QUERY, mapSqlParameterSource,
        issueRowMapper);
    }
}
```

Содержимое файла DatabaseIssueStatusDao.java

```
package by.bsuir.helpdesk.dao.impl;

import by.bsuir.helpdesk.dao.IssueStatusDao;
import by.bsuir.helpdesk.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;

@Repository
public class DatabaseIssueStatusDao implements IssueStatusDao {

    private static final String GET_AVAILABLE_STATUS_QUERY = "SELECT ts_id as ta_status from
issue_status WHERE ts_issue = ? ORDER BY ts_order;";

    private final NamedParameterJdbcTemplate namedParameterJdbcTemplate;
    private static final RowMapper<Status> issueStatusRowMapper = ((resultSet, i) -> {
        Status status = new Status();
        status.setId(resultSet.getInt(TableColumn.ISSUE_STATUS_ID));

        return status;
    });

    @Autowired
    public DatabaseIssueStatusDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate){
        this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
    }

    @Override
    public List<Status> getAvailableIssueStatus(int issueId) {

        return namedParameterJdbcTemplate.getJdbcOperations().query(GET_AVAILABLE_STATUS_QUERY,
new Object[]{issueId}, issueStatusRowMapper);
    }
}

```

Содержимое файла DatabaseUserDao.java

```

package by.bsuir.helpdesk.dao.impl;

import by.bsuir.helpdesk.dao.UserDao;
import by.bsuir.helpdesk.domain.Company;
import by.bsuir.helpdesk.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;

@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_issue, 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_ISSUE_QUERY = "select us_first_name, us_last_name,
us_username, us_picture from users inner join issue_user on pu_username=us_username where
pu_issue_id = :issueId 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_issue, us_company_id) VALUES
(:username, :email, :firstName, :lastName, :password, :picture, :canCreateIssue, :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.setCanCreateIssue(resultSet.getInt(TableColumn.CREATE_ISSUE) != 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 static final RowMapper<User> userOnIssueRowMapper = ((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> getUsersOnIssue(int issueId, int startFrom, int limit) {
        MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();

        mapSqlParameterSource.addValue(ParameterName.ISSUE_ID, issueId);
        mapSqlParameterSource.addValue(ParameterName.START_FROM, startFrom);
        mapSqlParameterSource.addValue(ParameterName.ROW_NUMBER, limit);

        return namedParameterJdbcTemplate.query(GET_USERS_ON_ISSUE_QUERY, mapSqlParameterSource,
userOnIssueRowMapper);
    }

    @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}, userOnIssueRowMapper);
    }
}

```

Содержимое файла DatabaseUserIssueDao.java

```

package by.bsuir.helpdesk.dao.impl;

import by.bsuir.helpdesk.dao.UserIssueDao;
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;

@Repository
public class DatabaseUserIssueDao implements UserIssueDao {

    private static final String IS_ASSIGN_QUERY = "SELECT count(*) from issue_user WHERE
pu_issue_id = :issueId and pu_username = :username;";

    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    @Autowired
    public DatabaseUserIssueDao(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {
        this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;
    }

    @Override
    public boolean isUserAssignOnIssue(String username, int issueId) {
        MapSqlParameterSource mapSqlParameterSource = new MapSqlParameterSource();
        mapSqlParameterSource.addValue(ParameterName.ISSUE_ID, issueId);
        mapSqlParameterSource.addValue(ParameterName.USERNAME, username);

        return namedParameterJdbcTemplate.queryForObject(IS_ASSIGN_QUERY, mapSqlParameterSource,
Integer.class) != 0;
    }
}

```

Содержимое файла RelevantIssueComparator.java

```

package by.bsuir.helpdesk.service.util;

import by.bsuir.helpdesk.domain.Tag;
import by.bsuir.helpdesk.domain.Issue;

import java.util.Comparator;
import java.util.Set;

public class RelevantIssueComparator implements Comparator<Issue> {

    private static final int GREATER = 1;
    private static final int EQUALS = 0;
    private static final int LESS = -1;
    private Set<Tag> searchedIssues;

    public RelevantIssueComparator(Set<Tag> searchedIssues) {
        this.searchedIssues = searchedIssues;
    }
}

```

```

@Override
public int compare(Issue issue1, Issue issue2) {
    int issue1ContainsCount = 0;
    int issue2ContainsCount = 0;

    for(Tag tag: searchedIssues){

        if(issue1.getTags().contains(tag)){
            issue1ContainsCount++;
        }

        if(issue2.getTags().contains(tag)){
            issue2ContainsCount++;
        }

    }

    if(issue1ContainsCount < issue2ContainsCount){
        return LESS;
    }
    else if(issue1ContainsCount > issue2ContainsCount){
        return GREATER;
    }
    else {
        return EQUALS;
    }
}
}

```

Содержимое файла MailNotificationSender.java

```

package by.bsuir.helpdesk.service.impl;

import by.bsuir.helpdesk.domain.Notification;
import by.bsuir.helpdesk.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;

@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 IssueService.java

```

package by.bsuir.helpdesk.service.impl;

import by.bsuir.helpdesk.dao.IssueDao;
import by.bsuir.helpdesk.dao.IssueDao;
import by.bsuir.helpdesk.dao.IssueStatusDao;
import by.bsuir.helpdesk.dao.UserDao;

```

```

import by.bsuir.helpdesk.domain.Issue;
import by.bsuir.helpdesk.domain.Status;
import by.bsuir.helpdesk.domain.Issue;
import by.bsuir.helpdesk.domain.User;
import by.bsuir.helpdesk.service.IssueService;
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;

@Service
public class DefaultIssueService implements IssueService {

    private final IssueDao issueDao;

    private final IssueDao issueDao;

    private final IssueStatusDao issueStatusDao;

    private final UserDao userDao;

    @Autowired
    public DefaultIssueService(IssueDao issueDao, IssueDao issueDao, IssueStatusDao
issueStatusDao, UserDao userDao) {
        this.issueDao = issueDao;
        this.issueDao = issueDao;
        this.issueStatusDao = issueStatusDao;
        this.userDao = userDao;
    }

    @Override
    public List<Issue> getUserIssues(User user, int limit, int startFrom) {

        return issueDao.getUsersIssues(user.getUsername(), limit, startFrom);
    }

    @Override
    @PreAuthorize("isAuthenticated() and principal.canCreateIssue")
    public void createIssue(Issue issue){
        User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        issue.setCompany(user.getCompany());
        issueDao.createIssue(issue);
    }

    @Override
    @PreAuthorize("@databaseUserIssueDao.isUserAssignOnIssue(principal.username, issueId)")
    public Issue getIssue(int issueId){
        Issue issue = issueDao.getIssue(issueId);

        List<Status> statuses = issueStatusDao.getAvailableIssueStatus(issueId);

        List<Issue> issues = new ArrayList<>();

        for(Status status : statuses){
            List<Issue> issuesInStatus = issueDao.getIssueIssues(issueId, status.getId(),
ISSUE_START_FROM, ISSUE_LIMIT);
            issues.addAll(issuesInStatus);
        }

        issue.setIssues(issues);
        issue.setUsers(userDao.getUsersOnIssue(issueId, ISSUE_START_FROM, ISSUE_LIMIT));

        return issue;
    }
}

```

Содержимое файла UserService.java

```

package by.bsuir.helpdesk.service.impl;

import by.bsuir.helpdesk.ImageConstant;

```

```

import by.bsuir.helpdesk.dao.UserDao;
import by.bsuir.helpdesk.domain.Notification;
import by.bsuir.helpdesk.domain.User;
import by.bsuir.helpdesk.service.NotificationSender;
import by.bsuir.helpdesk.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;

@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);
    }
}
}

```

ПРИЛОЖЕНИЕ Б
(обязательное)

Программное средство поддержки клиентов. Спецификация.

ПРИЛОЖЕНИЕ В
(обязательное)

Программное средство поддержки клиентов. Ведомость документов.