**Реферат**

**Дипломная работа** 46 страниц, 19 рисунков, 1 таблица, 6 источников, 2 приложения

**Ключевые слова:** система отслеживания ошибок, дефект, жизненный цикл дефекта, mvc, java ee, spring, hibernate

**Объект исследования:** система отслеживания ошибок.

**Цель работы:** проектирование и разработка системы отслеживания ошибок на платформе Java EE.

**Инструменты разработки:** IDE Eclipse, Apache Tomcat, MySQL.

**Выводы:** спроектирована архитектура и разработана система отслеживания ошибок приложений.

Содержание

[Введение 4](#_Toc452107763)

[1 Назначение разработанного приложения 6](#_Toc452107764)

[2 Сравнительный анализ существующих аналогов 7](#_Toc452107765)

[3 Обзор используемых технологий 9](#_Toc452107766)

[3.1 Базовая версия языка Java 9](#_Toc452107767)

[3.2 Корпоративная версия языка Java 10](#_Toc452107768)

[3.3 Сервлеты 10](#_Toc452107769)

[3.4 Технология JSP 11](#_Toc452107770)

[3.5 Технология JavaMail 13](#_Toc452107771)

[3.6 Фреймворк Spring 13](#_Toc452107772)

[3.7 Библиотека объектно-реляционного отображения Hibernate 14](#_Toc452107773)

[3.8 Фреймворк автоматизации сборки проектов Apache Maven 16](#_Toc452107774)

[4 Структура приложения 20](#_Toc452107775)

[4.1 Шаблон проектирования MVC 20](#_Toc452107776)

[4.2 Архитектурный слой представления 22](#_Toc452107777)

[4.3 Архитектурный слой контроллеров 23](#_Toc452107778)

[4.4 Архитектурный слой сервисов 24](#_Toc452107779)

[4.5 Архитектурный слой доступа к данным 25](#_Toc452107780)

[4.6 Архитектурный доменный слой 26](#_Toc452107781)

[4.7 Сервер базы данных 26](#_Toc452107782)

[5 Описание работы приложения 29](#_Toc452107783)

[5.1 Функциональный обзор 29](#_Toc452107784)

[5.2 Роли пользователей в системе 29](#_Toc452107785)

[5.3 Главная страница системы 31](#_Toc452107786)

[5.4 Работа с пользовательскими профилями 32](#_Toc452107787)

[5.5 Работа с дефектами 34](#_Toc452107788)

[5.5.1 Добавление дефекта 35](#_Toc452107789)

[5.5.2 Редактирование дефекта 36](#_Toc452107790)

[5.5.3 Жизненный цикл дефектов в системе 38](#_Toc452107791)

[5.6 Работа с проектами 39](#_Toc452107792)

[5.7 Работа с атрибутами дефектов 41](#_Toc452107793)

[5.7.1 Атрибут резолюции к дефектам 41](#_Toc452107794)

[5.7.2 Атрибут приоритета дефектов 42](#_Toc452107795)

[5.7.3 Атрибут типа дефектов 43](#_Toc452107796)

[5.7.4 Атрибут статуса дефектов 44](#_Toc452107797)

[Заключение 45](#_Toc452107798)

[Список используемой литературы 46](#_Toc452107799)

[Приложение A Описание сборки проекта в Maven 47](#_Toc452107800)

[Приложение Б Конфигурационные файлы Spring 51](#_Toc452107801)

# Введение

Идеальных программ не существует. Все программисты делают ошибки в своих проектах. Даже идеально протестированная программа может дать сбой. Дело в том, что программы работают в окружении других программ, написанных другими программистами.

Например, программа, которая была много раз протестирована, при переезде на другой сервер начинала работать совершенно неправильно. Выяснилось, что ошибка происходит на несколько звеньев раньше в процессе вычислений. Исходный код программы, написанный другим программистом под более старую версию операционной системы, выдавал некорректные данные. Это пример показывает, что ошибки в программе могут вызываться «внешним миром», в котором она работает.

А иногда ошибки, похожи на мины замедленного действия, которые ждут своего часа и находятся в самых неожиданных местах. Достаточно вспомнить пример с выходом пакета обновлений 3 для операционной системы Windows XP. У небольшой группы пользователей это обновление ОС вызывало постоянную перезагрузку компьютера. Выяснилось, что все пострадавшие были владельцами компьютеров HP с процессором AMD. Это показывает, какими изощренными могут быть сбои, когда программный продукт предназначен для огромного числа пользователей. И ошибка не всегда может заключаться в программе.

Учитывая, что многие фирмы, производящие ПО, стараются уменьшить цикл производства в ущерб тестированию, программистам приходится постоянно взаимодействовать со службой поддержки. Работники принимают от пользователей заявления об ошибках, регистрируют их и дальше с ними разбираются разработчики. Если же компания осознает, что необходимо проводить тщательное тестирование продукта, перед его запуском, то программистам приходится опять-таки принимать отчеты об ошибках, но теперь уже от специалистов по тестированию программного обеспечения.

Задача регистрации и обработки данных об ошибках, возникших при работе приложений, является непростой задачей. От версии к версии количество известных ошибок может уменьшаться или увеличиваться. Одно из решений данной проблемы – это использование специализированного приложения для отслеживания и управления дефектами.

Система отслеживания ошибок – прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, специалистам по тестированию и др.) учитывать и контролировать ошибки (дефекты), найденные в программах, а также следить за процессом устранения этих ошибок. Такое приложение позволяет добавить дефект и провести его через весь жизненный цикл – добавление, поиск решения, решение, проверка, закрытие.

На самом деле многие системы отслеживания ошибок позволяют вести намного более подробный учет дефектов. В чем то, они напоминают системы управления проектами. А многие из них интегрированы с такими системами.

Необходимо заметить, что системы отслеживания ошибок могут быть полезны не только для программистов и специалистов по тестированию - менеджеры проектов и обычные пользователи также могут следить за исправлением текущих дефектов без обращения в службу поддержки.

Целью дипломной работы является написание системы отслеживания ошибок на платформе Java Enterprise Edition (Java EE).

# 1 Назначение разработанного приложения

Разработанное приложение является системой отслеживания ошибок для небольших команд программистов, написанное с целью помочь разработчикам программного обеспечения (программистам, специалистам по тестированию и менеджерам) учитывать и контролировать ошибки и неполадки, найденные в программах, а также следить за процессом устранения этих ошибок.

Система позволяет аутентифицированным пользователям добавлять дефекты и проводить их через весь, зачастую непростой жизненный цикл - добавление, поиск решения, решение, проверка, закрытие или повторное открытие.

Приложение поддерживает ведение нескольких проектов и гибкую систему доступа, основанную на ролях: гость, пользователь и администратор. Так же поддерживается гибкая градация серьёзности и приоритета дефектов. Это требуется для того, чтобы уведомить проектную команду в первую очередь о дефектах, требующих незамедлительного исправления.

Так же система поддерживает технологию оповещения пользователей по электронной почте – программисту не требуется постоянно обращаться к системе, чтобы узнать о новых дефектах, за исправление которых он назначен ответственным. Система уведомляет менеджеров или специалистов по тестированию в случае, когда программист изменяет состояние дефекта.

В системе так же присутствует легко расширяемый механизм интернационализации, позволяющий "на лету" менять язык приложения, а так же постраничное разделение больших списков дефектов или проектов.

# 2 Сравнительный анализ существующих аналогов

Необходимо проанализировать существующие аналоги систем отслеживания ошибок. Среди них стоит выделить самые популярные, такие как Jira, Bugzilla и Track Studio.

Система отслеживания ошибок от компании Atlassian – **Jira** является коммерческой системой, предназначенной для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами. JIRA обладает на сегодняшний день наиболее широкой функциональностью среди систем отслеживания ошибок и имеет большое количество возможностей конфигурации. Благодаря универсальному подходу можно приспособить систему для многих непрофильных задач. Также система неплохо интегрируется с внешними системами и системами контроля версий.

Однако Jira обладает тремя существенными недостатками. Первый из них это громоздкость – развёрнутая система потребует от компании покупки и последующей поддержки отдельного мощного сервера. Для небольших компаний это непозволительная роскошь. Второй недостаток Jira – излишняя перегруженность функционалом. В случае прихода в команду молодого разработчика, не один день потребуется, чтобы разобраться, как работать с системой. Не говоря уже об администрировании – на обучение специалиста уйдёт значительно больше времени. И, наконец, третий, чаще всего решающий недостаток – стоимость лицензии. Она зависит от числа пользователей системы и на текущий момент составляет около $50 за пользователя. Хотя для проектов с открытым исходным кодом Atlassian предоставляет специальную бесплатную лицензию при соблюдении некоторых правил.

**Bugzilla** – это одна из наиболее старых и популярных систем отслеживания ошибок. Она является свободным ПО и разработку этой системы сейчас ведет Mozilla Foundation. Bugzilla используют более восьмисот компаний по всему миру. В системе нет той огромной функциональности, присущей Jira. Однако эта система включает достаточно большой набор функций, которые необходимы для контроля ошибок в небольших и средних проектах.

К недостаткам Bugzilla можно отнести сложность установки и администрирования, зависимость от модулей Perl и несколько неприглядный интерфейс. Один из разработчиков системы указывал на ещё один недостаток - это выбор Perl в качестве языка программирования: проблема «читабельности» кода усложняет поддержку, Perl далеко не лучшим образом работают с памятью.

И, наконец, последняя система от российской компании ГРАН – **Track Studio**. Это не классическая система отслеживания ошибок, а комплексная система позволяющая управлять проектами и требованиями к приложениям. В отличие от JIRA, оптимизированной для работы с внешними клиентами, Track Studio позволяет эффективно организовать работу внутри компании. Система позволяет эффективно управлять большим числом проектов, хорошо поддерживает базы данных Oracle.

Однако и Track Studio не лишена недостатков – у приложения достаточно сложный интерфейс и высокая стоимость. Цены на систему начинаются от $500, поэтому её имеет смысл использовать только при разработке крупных проектов, когда возникает потребность задействовать все возможности по управлению проектами, входящие в состав Track Studio.

Как видно из сравнения, разработанная система отслеживания ошибок имеет ряд преимуществ перед своими аналогами: система не громоздкая, бесплатная, имеет простой и не перегруженный функционалом интерфейс, и при этом обладает большинством базовых возможностей существующих систем.

# 3 Обзор используемых технологий

При написании приложения использовано множество технологий, основные из которых базовая и промышленная (корпоративная) версии языка Java, фреймворк Spring, библиотека объектно-реляционного отображения Hibernate, а также фреймворк автоматизации сборки проектов Apache Maven.

Кратко рассмотрим особенности каждой из указанных технологий.

## 3.1 Базовая версия языка Java

Объектно-ориентированный язык Java, разработанный в компании Sun Microsystems в 1995 году для оживления графики на стороне клиента с помощью апплетов, в настоящее время используется для создания переносимых на различные платформы и операционные системы программ. Язык Java нашел широкое применение в веб-приложениях, добавив на статические и клиентские веб-страницы динамическую графику, улучшив интерфейсы и реализовав вычислительные возможности. Но объектно-ориентированная парадигма и кросс-платформенность привели к тому, что уже буквально через несколько лет после создания язык практически покинул клиентские страницы и перебрался на серверы. На стороне клиента его место заняли языки JavaScript, Adobe Flash и проч.

При создании язык Java предполагался более простым, чем его синтаксический предок C++. Сегодня с появлением новых версий возможности языка Java существенно расширились и во многом перекрывают функциональность C++. Java уже не уступает по сложности предшественникам и называть его простым нельзя.

Отсутствие указателей (наиболее опасного средства языка C++) нельзя считать сужением возможностей, а тем более – недостатком, это просто требование безопасности. Возможность работы с произвольными адресами памяти через безтиповые указатели позволяет игнорировать защиту памяти.

Отсутствие в Java множественного наследования легко заменяется на более понятные конструкции с применением интерфейсов. Системная библиотека классов языка Java содержит классы и пакеты, реализующие и расширяющие базовые возможности языка, а также сетевые средства, взаимодействие с базами данных, графические интерфейсы и многое другое. Методы классов, включенных в эти библиотеки, вызываются JVM (Java Virtual Machine) во время интерпретации программы.

В Java все объекты программы расположены в динамической памяти — куче данных (heap) и доступны по объектным ссылкам, которые, в свою очередь, хранятся в стеке (stack). Это решение исключило непосредственный доступ к памяти, но усложнило работу с элементами массивов и сделало ее менее эффективной по сравнению с программами на C++. В свою очередь, в Java предложен усовершенствованный механизм работы с коллекциями, реализующими основные динамические структуры данных. Необходимо отметить, что объектная ссылка языка Java содержат информацию о классе объекта, на который она ссылается, так что объектная ссылка – это не указатель, а дескриптор (описание) объекта. Наличие дескрипторов позволяет JVM выполнять проверку совместимости типов на фазе интерпретации кода, генерируя исключение в случае ошибки. В Java изменена концепция организации динамического распределения памяти: отсутствуют способы программного освобождения динамически выделенной памяти. Вместо этого реализована система автоматического освобождения памяти (сборщик мусора), выделенной с помощью оператора new. Программист может только рекомендовать системе освободить выделенную динамическую память.

В отличие от C++, Java не поддерживает множественное наследование, перегрузку операторов, беззнаковые целые, прямое индексирование памяти и, как следствие, указатели. В Java существуют конструкторы, но отсутствуют деструкторы (применяется автоматическая сборка мусора), не используется оператор goto и слово const, хотя они являются зарезервированными словами языка.

## 3.2 Корпоративная версия языка Java

Java Enterprise Edition (Java EE) – набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий. Java EE является промышленной технологией и в основном используется в высокопроизводительных проектах, в которых необходима надежность, масштабируемость, гибкость. Технологии, входящие в состав Java EE существенно сокращают стоимость и сложность разработки, развертывания многоуровневых серверных приложений, а также управления ими.

Ввиду многочисленности технологий, входящих в состав платформы, ниже представлены только основные технологии, использованные при разработке системы:

– Сервлеты (Англ. Servlet) – спецификация сервлетов определяет набор программных интерфейсов для обслуживания HTTP-запросов;

– Java Server Pages (JSP) – технология динамической генерации веб-страниц на стороне сервера;

– JSP Standard Tag Library (JSTL) – расширение спецификации JSP, добавляющее библиотеку JSP тегов для общих нужд;

– JavaMail – технология для получения и отправки электронной почты.

## 3.3 Сервлеты

Сервлеты (Англ. Servlet) – является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP (Англ. HyperText Transfer Protocol). Технология сервлетов является оболочкой протокола http и поддерживает его как транспорт передачи данных от клиента серверу и обратно. Контейнер сервлетов поддерживает также протокол HTTPS (Англ. HyperText Transfer Protocol Secure) для защищаемых запросов.

Сервлеты в промышленном программировании используются для:

– приема входящих данных от клиента;

– взаимодействия с логикой работы системы;

– динамической генерации ответа клиенту.

Все сервлеты реализуют общий интерфейс Servlet из пакета javax.servlet. Для обработки http-запросов в web в качестве базового класса используется HttpServlet из пакета javax.servlet.http.

Жизненный цикл сервлета состоит из следующих шагов:

а) В случае отсутствия сервлета в контейнере;

1) Класс сервлета загружается контейнером;

2) Контейнер создает экземпляр класса сервлета;

3) Контейнер вызывает метод init(). Этот метод инициализирует сервлет и вызывается в первую очередь, до того, как сервлет сможет обслуживать запросы. За весь жизненный цикл метод init() вызывается только один раз;

б) Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает метод service() для каждого запроса. Этот метод определяет тип пришедшего запроса и распределяет его в соответствующий этому типу метод для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих методов. Если поступил запрос, метод для которого не реализован, вызывается метод родительского класса и обычно завершается возвращением ошибки инициатору запроса;

в) В случае если контейнеру необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации. Подобно методу init(), этот метод тоже вызывается единожды за весь цикл сервлета.

## 3.4 Технология JSP

Java Server Pages (JSP) – технология, позволяющая веб-разработчикам создавать содержимое, которое имеет как статические, так и динамические компоненты. Страница JSP содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP-элементы, которые конструируют динамическое содержимое. Кроме этого могут использоваться библиотеки JSP-тегов, а также EL (Expression Language), для внедрения Java-кода в статичное содержимое JSP-страниц.

Код JSP-страницы транслируется в Java-код сервлета с помощью компилятора JSP-страниц Jasper, и затем компилируется в байт-код виртуальной машины java (JVM). Контейнеры сервлетов, способные исполнять JSP-страницы, написаны на кросс-платформенном языке Java. JSP-страницы загружаются на сервере и управляются из структуры специального Java server packet, который называется Java EE Web Application. Обычно страницы упакованы в файловые архивы .war и .ear.

JSP является платформо-независимой, переносимой и легко расширяемой технологией для разработки веб-приложений.

JSP страницы имеют расширение .jsp и размещаются там же, где и обычные Web-страницы. Структура таких страниц может состоять из пяти конструкций: HTML, комментарии, скриптовые элементы, директивы и действия. JSP-страница при компиляции преобразуется в сервлет со статическим содержимым, которое направляется в поток вывода, связанный с методом service. Поэтому при первом запросе этот процесс может вызвать небольшую задержку. Комментарии в документе или программе не являются причиной замедления программы, так как транслятор и исполнитель их игнорируют. Скриптовые элементы позволяют указать код на языке Java, который впоследствии станет частью конечного сервлета, директивы дают возможность управлять всей структурой сервлета, а действия служат для задания существующих используемых компонентов, а также для контроля над поведением движка JSP. Для упрощения работы со скриптами имеются заранее определённые переменные, такие как request, response, pageContext, session, out, application, config, page, exception.

Процессы, выполняемые с файлом JSP при первом вызове:

– браузер делает запрос к странице JSP;

– JSP-engine анализирует содержание файла JSP и создает сервлет с кодом, основанным на исходном тексте файла JSP, при этом engine транслирует статическое содержимое в методы вывода и помещает его в метод \_jspService(). Полученный сервлет будет ответственен за генерацию статических элементов, определенных во время разработки. Динамические элементы транслируются в java-код;

– код сервлета компилируется в class-файл и загружается в контейнер. В итоге сервлет на основе JSP установлен и готов к работе;

– выполняется метод init() сервлета;

– вызывается метод \_jspService(), и сервлет логически исполняется, формируя экземпляр response;

– комбинация статического HTML и графики вместе с результатами исполнения динамических элементов, определенных в оригинале JSP, пересылаются браузеру через выходной поток объекта ответа HttpServletResponse. Следующие обращения к файлу JSP просто вызовут метод \_jspService() сервлета. Сервлет используется до тех пор, пока сервер не будет остановлен или сервлет не будет выгружен из контейнера.

## 3.5 Технология JavaMail

JavaMail – это технология, предназначенная для получения и отправки электронной почты с использованием протоколов SMTP, POP3 и IMAP. JavaMail является частью платформы Java EE, но также доступна в качестве дополнительного пакета для использования в приложениях базовой библиотеки Java.

В состав JavaMail не входит почтовый сервер, однако для подобных задач могут использоваться свободные библиотеки Apache James и Java Email Server (POP3 и SMTP), или библиотека SubEthaSMTP для создания SMTP сервера.

## 3.6 Фреймворк Spring

Spring Framework (Spring) – это универсальный фреймворк с открытым исходным кодом для Java-платформы. Несмотря на то, что Spring Framework не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена программной модели Enterprise JavaBeans. Spring Framework предоставляет большую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring Framework 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.

Spring Framework может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Эти фреймворки делятся на структурные элементы типовых комплексных приложений:

– Контейнер инверсии зависимостей: конфигурирование компонентов приложений и управление жизненным циклом Java-объектов;

– Фреймворк аспектно-ориентированного программирования: работает с функциональностью, которая не может быть реализована возможностями объектно-ориентированного программирования на Java без потерь;

– Фреймворк доступа к данным: работает с системами управления реляционными базами данных на Java-платформе, используя JDBC и ORM-средства и обеспечивая решения задач, которые повторяются в большом числе приложений на платформе Java;

– Фреймворк управления транзакциями: координация различных API управления транзакциями и инструментарий настраиваемого управления транзакциями для объектов Java;

– Фреймворк MVC: каркас, основанный на HTTP и сервлетах, предоставляющий множество возможностей для расширения и настройки;

– Фреймворк удалённого доступа: конфигурируемая передача Java-объектов через сеть в стиле RPC, поддерживающая RMI, CORBA и базирующиеся на http протоколы, включая web-сервисы (SOAP);

– Фреймворк аутентификации и авторизации: конфигурируемый инструментарий процессов аутентификации и авторизации, поддерживающий много популярных и ставших индустриальными стандартами протоколов, инструментов, практик через дочерний проект Spring Security (ранее известный как Acegi).

## 3.7 Библиотека объектно-реляционного отображения Hibernate

Hibernate – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (Англ. object-relational mapping – ORM). Она представляет собой свободное программное обеспечение с открытым исходным кодом, распространяемое на условиях стандартной общественной лицензии ограниченного применения GNU. Данная библиотека предоставляет легкий в использовании каркас (фреймворк) для отображения объектно-ориентированной модели данных в традиционные реляционные базы данных.

Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования по обеспечению хранения объектов в реляционной базе данных. Разработчик может использовать Hibernate как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой данных.

Hibernate не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос приложения на любые базы данных SQL.

Hibernate обеспечивает прозрачную поддержку сохранности данных для «POJO» (то есть для стандартных Java-объектов). Единственное строгое требование для сохраняемого класса – наличие конструктора по умолчанию (без параметров). Для корректного поведения в некоторых приложениях требуется также уделить внимание методам equals() и hashCode().

Сопоставление (проецирование) Java-классов с таблицами БД осуществляется с помощью конфигурационных XML-файлов или Java-аннотаций. При использовании файла XML Hibernate может генерировать скелет исходного кода для классов длительного хранения (persistent). В этом нет необходимости, если используется аннотация. Hibernate может использовать файл XML или аннотации для поддержки схемы базы данных.

Обеспечиваются возможности по организации отношения между классами "один-ко-многим" и "многие-ко-многим". В дополнение к управлению связями между объектами Hibernate также может управлять рефлексивными отношениями, где объект имеет связь "один-ко-многим" с другими экземплярами своего собственного типа данных.

Hibernate поддерживает отображение пользовательских типов значений. Это делает возможными такие сценарии:

– Переопределение типа по умолчанию SQL, Hibernate выбирает при отображении столбца свойства;

– Проецирование перечисляемого типа Java на поле БД, будто они являются обычными свойствами;

– Проецирование одного свойства в несколько колонок.

Коллекции объектов данных, как правило, хранятся в виде коллекций Java-объектов, таких, как набор (Англ. Set) и список (Англ. List). Поддерживаются обобщенные классы (Англ. Generics), введенные в Java 5. Hibernate может быть настроен на «ленивые» (отложенные) загрузки коллекций. Отложенные загрузки является вариантом по умолчанию, начиная с Hibernate 3.

Связанные объекты могут быть настроены на каскадные операции. Например, родительский класс может быть настроен на каскадное сохранение и/или удаление своего потомка. Это может сократить время разработки и обеспечить целостность. Функция проверки изменения данных позволяет избежать ненужной записи действий в базу данных, выполняя SQL-обновление только при изменении полей персистентных объектов.

Успех библиотеки Hibernate подтолкнул JCP к разработке спецификации JDO, ставшей одной из стандартных технологий ORM на платформе Java EE.

Hibernate обеспечивает использование SQL-подобного языка Hibernate Query Language (HQL), который позволяет выполнять SQL-подобные запросы, записанные рядом с объектами данных Hibernate. Запросы критериев предоставляются как Объектно-ориентированная альтернатива к HQL.

Hibernate может использоваться как в самостоятельных приложениях Java, так и в программах Java EE, выполняемых на сервере (например, сервлет или компоненты EJB). Также он может включаться как дополнительная возможность к другим языкам программирования.

## 3.8 Фреймворк автоматизации сборки проектов Apache Maven

Apache Maven (Maven) – фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (Англ. Project Object Model), являющемся подмножеством XML. Проект Maven издаётся сообществом Apache Software Foundation, где формально является частью Jakarta Project.

Maven обеспечивает декларативную, а не императивную (в отличие от средства автоматизации сборки Apache Ant) сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов, описанные в спецификации, Maven выполняет посредством их обработки последовательностью встроенных и внешних модулей.

Maven используется для построения и управления проектами, написанными на Java, C#, Ruby, Scala, и других языках.

Информация для сборки проекта, поддерживаемого Apache Maven, содержится в XML-файле с названием pom.xml. При запуске Maven проверяет, содержит ли конфигурационный файл все необходимые данные и все ли данные синтаксически правильно записаны.

Минимальная конфигурация включает версию конфигурационного файла, имя проекта, его автора и версию. С помощью pom.xml конфигурируются зависимости от других проектов, индивидуальные фазы процесса построения проекта, список модулей, реализующих порядок сборки.

Крупные проекты могут быть поделены на несколько модулей, или проектов, каждый со своим собственным POM. Операции над модулями могут выполняться через общий корневой POM единой командой.

POM файлы проектов могут наследовать конфигурацию от других файлов конфигурации. В то же время все файлы конфигурации обязательно наследуются от главного POM файла по умолчанию. Главный POM файл обеспечивает конфигурацию по умолчанию, например, стандартная структура каталогов, используемые по умолчанию модули, привязка к фазам жизненного цикла и прочее.

Maven поддерживает принцип "соглашения прежде конфигурации". Поскольку проект её придерживается – отсутствует необходимость указывать пути к файлам в явном виде, что упрощает содержимое pom.xml. Однако, почти все стандарты, на которые опирается Maven, могут быть изменены индивидуальной конфигурацией.

Maven использует принцип Maven-архетипов (Англ. Archetypes). Архетип – это инструмент шаблонов, каждый из которых определён паттерном или моделью, по аналогии с которой создаются производные.

Стандартная структура каталогов – одна из реализаций принципа архетипов в Maven. Следующая структура показывает важнейшие каталоги для проекта на Java:

– Корневой каталог проекта: файл pom.xml и все дальнейшие подкаталоги;

– Каталог src: все исходные файлы;

– Каталог src/main: исходный код приложения;

– Каталог src/main/java: исходный код на языке java;

– Каталог src/main/resources: другие файлы, которые используются при компиляции или исполнении, например файлы свойств;

– Каталог src/test: исходные файлы, необходимые для организации автоматического тестирования;

– Каталог src/test/ java: java-классы для автоматического тестирования;

– Каталог target: все создаваемые в процессе работы Maven файлы;

– Каталог target/classes: скомпилированные Java-классы.

Жизненный цикл Maven проекта – это список поименованных фаз, определяющий порядок действий при его построении. Стандартные жизненные циклы могут быть дополнены функционалом с помощью Maven-плагинов. Плагины позволяют вставлять в стандартный цикл новые шаги (например, распределение на сервер приложений) или расширять существующие шаги.

Maven базируется на plugin-архитектуре, которая позволяет применять плагины для различных задач (compile, test, build, deploy, checkstyle, pmd, scp-transfer) для данного проекта, без необходимости их в явном виде инсталлировать. Это возможно за счет того, что информация поступает плагину через стандартный вход, а результаты пишутся в его стандартный выход. Теоретически, это позволяет, кому угодно писать плагины для взаимодействия со средствами построения проекта (компиляторы, средства тестирования, и т. п.) для любого другого языка. В реальности, поддержка других языков кроме Java сейчас минимальна. Существует плагин для .NET-фреймворка, а также плагины для C/C++ maven-native и maven-nar.

Количество доступных плагинов в настоящее время очень велико и включает: плагины, позволяющие непосредственно из Maven запускать web-приложение для тестирования его в браузере; плагины, позволяющие тестировать или создавать банки данных; плагины, позволяющие генерировать Web Services. Задачей разработчика в такой ситуации является найти и применить наиболее подходящий набор плагинов.

В файле pom.xml задаются зависимости, которые имеет управляемый с помощью Maven проект. Менеджер зависимостей основан на нескольких основных принципах:

– Репозитории. Maven ищет необходимые файлы в локальных каталогах или в локальном Maven-репозитории. Если зависимость не может быть локально разрешена, Maven подключается к указанному Maven-репозиторию в сети и копирует в локальный репозиторий. По умолчанию Maven использует Maven Central Repository, но разработчик может конфигурировать и другие публичные Maven-репозитории, такие, как Apache, Ibiblio, Codehaus или Java.Net;

– Транзитивные зависимости. Необходимые библиотеки подгружаются в проект автоматически. При разрешении конфликта версий используется принцип «ближайшей» зависимости, то есть выбирается зависимость, путь к которой через список зависимых проектов является наиболее коротким;

– Исключение зависимостей. Файл описания проекта предусматривает возможность исключить зависимость в случае обнаружения цикличности или отсутствия необходимости в определённой библиотеке;

– Поиск зависимостей. Поиск зависимостей (open-source-библиотек и модулей) ведётся по их координатам (groupId, artifactId и version). Эти координаты могут быть определены с помощью специальных поисковых машин, например, Maven search engine. Например, по поисковому признаку «pop3», поисковая машина предоставляет результат с groupId="com.sun.mail" и artifactId="pop3";

– Менеджеры репозиториев. Репозитории реализуются с помощью менеджеров репозиториев Maven (Maven Repository Manager), таких как Apache Archiva, Nexus, Artifactory, Codehaus Maven Proxy или Dead Simple Maven Proxy.

Область распространения зависимости позволяет включать зависимости только на определённую стадию построения проекта. Существует 6 возможных областей:

– compile. Область по умолчанию. Зависимость доступна во всех путях поиска классов в проекте. Распространяется на зависимые проекты;

– provided. Область аналогична compile, за исключением того, что JDK или контейнер сам предоставит зависимость во время выполнения программы;

– runtime. Зависимость не нужна для компиляции, но нужна для выполнения;

– test. Зависимость не нужна для нормальной работы приложения, а нужна только для компиляции и запуска тестов;

– system. Область аналогична provided за исключением того, что содержащий зависимость JAR указывается явно. Артефакт не ищется в репозитории;

– import (начиная с версии Maven 2.0.9) используется только с зависимостью типа pom в секции dependencyManagement. Зависимости текущего POM заменяются на зависимости из указанного POM.

Проект, поддерживающийся с помощью Maven, должен удовлетворять некоторым условиям для возможности его чтения утилитой, последующего анализа и возможности сборки. Это накладывает некоторые ограничения на структуру директорий и требует дополнительных действий, если проект изначально имеет отличную структуру.

Для того, чтобы Maven распознал проект, как подлежащий обработке, он должен содержать установленную структуру каталогов. Все файлы с исходным кодом должны находиться по относительному пути "\src\main\java".

Файл конфигурации web-проекта web.xml должен находиться в директории "\src\main\webapp\WEB-INF".

Файл конфигурации Maven-проекта pom.xml должен находиться в корневом каталоге проекта. В нём, согласно его предназначению, могут быть указаны удаленный репозиторий, плагины для создания архивов, плагин компилятора и прочее.

# 4 Структура приложения

Концепция слоев – одна из общеупотребительных моделей, используемых разработчиками программного обеспечения для разделения сложных систем на более простые части. Описывая систему в терминах архитектурных слоев, удобно воспринимать составляющие ее подсистемы в виде "слоеного пирога". Слой более высокого уровня пользуется службами, предоставляемыми нижележащим слоем, но тот не "осведомлен" о наличии соседнего верхнего слоя. Более того, обычно каждый промежуточный слой "скрывает" нижний слой от верхнего: например, слой 4 пользуется услугами слоя 3, который обращается к слою 2, но слой 4 не знает о существовании слоя 2. Не в каждой архитектуре слои настолько "непроницаемы", но в большинстве случаев дело обстоит именно так.

Расчленение системы на слои предоставляет целый ряд преимуществ.

– Отдельный слой можно воспринимать как единое самодостаточное целое, не особенно заботясь о наличии других слоев;

– Можно выбирать альтернативную реализацию базовых слоев;

– Зависимость между слоями можно свести к минимуму;

– Созданный слой может служить основой для нескольких различных слоев более высокого уровня.

Схема расслоения обладает и определенными недостатками.

– Слои способны удачно инкапсулировать многое, но не все: модификация одного слоя подчас связана с необходимостью внесения каскадных изменений в остальные слои;

– Наличие избыточных слоев нередко снижает производительность системы. При переходе от слоя к слою моделируемые сущности обычно подвергаются преобразованиям из одного представления в другое. Несмотря на это, инкапсуляция нижележащих функций зачастую позволяет достичь весьма существенного преимущества.

Система состоит из нескольких архитектурных слоёв и строится на базе шаблона проектирования MVC.

## 4.1 Шаблон проектирования MVC

MVC – шаблон проектирования архитектуры приложения, при котором модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы каждый компонент был максимально независим и его изменение не требовало модификации остальных компонентов. Основной целью данного шаблона является отделение бизнес-логики от графического представления данных в приложении. Благодаря этому улучшается модульность кода, что позволяет легко использовать компоненты повторно. Самым большим преимуществом MVC является возможность выбора представления данных без изменения бизнес-логики. А именно, реализуются следующие возможности:

– к модели присоединяется несколько представлений, без изменения самой модели. К примеру, это позволяет одним пользователям отображать данные в виде таблицы с возможностью редактирования, а другим в режиме просмотра;

– не изменяя представление, можно изменить поведение приложения, путем переопределения контроллера. К примеру, изменить реакцию интерфейса на нажатие кнопок мыши или ввод данных;

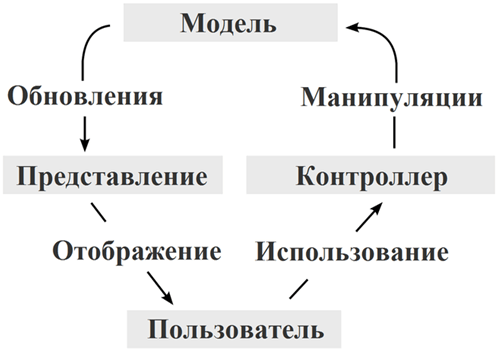
– позволяет внедрить разделение специализации разработчиков: часть разработчиков реализуют лишь пользовательский интерфейс, а другие специализируются на построении бизнес логики. Разработчики даже могут не знать об используемом представлении, поскольку компоненты независимы.

Концепция MVC предполагает обособление методов доступа к данным, представление и реакцию на действия пользователя в три отдельных компонента (Рисунок 4.1):

– Модель (Англ. Model). Модель инкапсулирует обращение и прочие функции для работы с данными. Модель изменяет свое состояние, реагируя на запросы, и не имеет представления о том, как данные будут отображены;

– Представление (Англ. View). Отвечает за визуализацию информации. Зачастую в качестве представления используется форма графических элементов. Также часто применяется отображение в табличном, графическом и прочих форматах;

– Контроллер (Англ. Controller). Является посредником между системой и пользователем. Контролирует ввод информации пользователя и влияет на модель и представления для отображения нужной информации и реакции на действия пользователя.

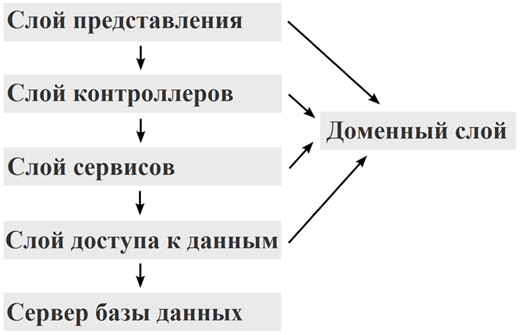


**Рисунок 4.1 – Схема взаимодействия базовой концепции MVC**

Важной особенностью является то, что контроллер и представление зависят от модели. Обратное неверно, модель не имеет зависимости ни от контроллера, ни от представления. Таким образом, реализуется одно из преимуществ такого обособления: построение модели можно производить независимо от представления данных. Помимо всего прочего, такой подход позволяет реализовать несколько представлений одной и той же модели.

Однако применение шаблона на практике приводит к тому, что трехуровневая базовая модель распадается на многоуровневую модель. Число и назначение слоев может существенно отличаться в зависимости от разработанного архитектурного решения.

В разработанном приложении наиболее удобной оказалась следующая методика разделения слоёв: представление и контроллер остались без существенных изменений базовой концепции, а слой модели разделён на составляющие: уровень сервисов, обрабатывающих объект запроса, уровень с доменными объектами и уровень организации взаимодействия с базой данных (Рисунок 4.2).



**Рисунок 4.2 –Реализация разделения слоёв в системе**

Далее подробно рассмотрим реализацию каждого из слоёв системы.

## 4.2 Архитектурный слой представления

Слой представления (Англ. View layer) – предназначен для отображения пользовательского интерфейса и охватывает все, что имеет отношение к общению пользователя с системой. В системе он представлен jsp-файлами и представлен в каталоге WEB-INF/views. Файлы стилей, шрифтов и иной статический контент размещены в каталоге /resources.

Для каждой из страниц системы заведено отдельное представление со своим назначением:

– добавление дефектов, приоритетов, проектов, резолюций, типов и пользователей;

– редактирование данных дефектов, приоритетов, проектов, резолюций, статусов, типов, профилей других пользователей и своего профиля в системе;

– отображение списков дефектов, приоритетов, проектов, резолюций, статусов, типов и пользователей;

– поиск дефектов и пользователей по указанным критериям.

Для всех текстовых полей выполняется проверка корректности или пустоты полей, посредством языка JavaScript, с выводом соответствующих предупреждений в случае ошибки.

Для страниц отображения списков дефектов и проектов используется пагинация и сортировка данных посредством библиотек языка JavaScript: jQuery и AJAX.

На каждой странице системы действует механизм "горячего" переключения языка для всех строковых сообщений – русский, белорусский или английский. Функционал реализуется посредством конфигурирования spring-фреймворка при помощи сущностей *LocaleChangeInterceptor* и *CookieLocaleResolver*. Файлы свойств с сообщениями отдельные для каждого из языков и хранятся в каталоге src/main/resource.

## 4.3 Архитектурный слой контроллеров

Слой контроллеров (Англ. Controller layer) – является связующим звеном между представлением и моделью данных. В системе он представлен java-классами, которые располагаются в пакете by.gsu.bugtracker.web.servlets.

В типовом Java EE приложении используются классы, унаследованные от HttpServlet и переопределяющие соответствующие методы, однако использование возможностей фреймворка Spring значительно сокращает объём кода: классы контроллеров помечаются аннотацией @Controller без наследования и переопределения методов doGet и doPost. Соответствие ссылок и контроллеров/методов-обработчиков осуществляется посредством аннотации @RequestMapping.

Различные возможные действия объединены в 7 классов контроллеров:

– UsersServlet обрабатывает добавление, просмотр, редактирование и поиск профилей пользователей;

– RootServlet предназначен для отображения и пагинации списков дефектов, а также поиска и просмотра отдельных дефектов. Иными словами, этот контроллер обрабатывает все действия неавторизованных пользователей;

– PropertyServlet рассчитан на обработку действий по добавлению или изменению свойств дефектов: приоритетов, типов, резолюций и статусов;

– ProjectsServlet обрабатывает возможные действия с проектами: отображение и пагинация списков проектов, добавление проектов и версий сборок, а так же изменение данных проектов;

– IssuesServlet предназначен для работы с дефектами: добавление и изменение дефектов, прикрепление комментариев и файлов к дефектам;

– EditProfileServlet нужен только для редактирования своих данных в профиле;

– GlobalExceptionController требуется для отображения страницы с ошибкой при появлении непредвиденной ошибки работы сервера.

## 4.4 Архитектурный слой сервисов

Слой сервисов (Англ. Service layer) – является связующим звеном между слоем доступа к данным и контроллерами системы. Этот слой предназначен для выполнения операций с методами, в которых используются несколько реализаций классов доступа к данным. Также методы этого слоя поддерживают транзакционные операции (операции с отменяемой последовательностью действий). В системе слой сервисов представлен интерфейсами, которые располагаются в пакете by.gsu.bugtracker.service, и реализациями этих интерфейсов в пакете by.gsu.bugtracker.service.impl.

Классы-реализации интерфейсов слоя сервисов помечаются аннотацией @Service для объявления Spring, что этот класс представляет сервис – использование данной аннотации позволит фреймворку искать сервисы автоматически.

Некоторые методы сервисов помечаются аннотацией @Transactional. Перед исполнением таких методов начинается транзакция (отменяемая последовательность действий) и в случае, если на какой-либо промежуточной операции возникло исключение RuntimeException, транзакция "откатывается" к начальному состоянию.

Классы слоя сервисов включают в себя как обобщённые наборы действий для всех сущностей, так и более специализированные и характерные только для одной. Ниже перечислены назначения реализаций:

– AbstractGenericService является параметризованным абстрактным классом для обработки общих наборов действий для сущностей, таких как получение сущности по идентификатору, получение всех сущностей в системе, создание, редактирование, удаление и получение количества сущностей в системе;

– IssueService содержит специализированные действия по работе с дефектами, которые не входят в общий набор: создание и изменение дефекта с уведомлением пользователя, получение сортированного списка дефектов, методы для получения статуса в зависимости от текущего статуса дефекта и метод прикрепления файла к дефекту;

– MyMailSender предназначен для уведомления пользователей. В системе реализуется уведомление пользователей по электронной почте, а именно уведомление пользователя, который создал дефект и ответственного за исправление, при создании и редактировании дефекта;

– ProjectService выполняет специализированные действия по работе с проектами: получение списка версий сборки конкретного проекта, получение сортированного списка проектов и создание новой версии сборки;

– UserService обрабатывает специализированные действия по работе с пользователями, какие как получение пользователя по электронной почте и получение результата поиска пользователей.

## 4.5 Архитектурный слой доступа к данным

Слой доступа к данным (Англ. Data Access Object layer или DAO layer) – предоставляет интерфейс к какому-либо типу хранилища данных или механизму хранения. Этот слой является связующим звеном между слоем сервисов и базой данных приложения. В системе слой доступа к данным представлен интерфейсами, которые располагаются в пакете by.gsu.bugtracker.dao, и реализациями этих интерфейсов в пакете by.gsu.bugtracker.dao.persistence.

Классы слоя сервисов помечаются аннотацией @Repository для объявления Spring, что этот класс функционирует как репозиторий и требует наличия прозрачной трансляции исключений. Практическим смыслом трансляции исключений является то, что слой сервиса будет иметь дело с общей иерархией исключений от Spring (DataAccessException) вне зависимости от используемых технологий доступа к данным в слое данных.

Все запросы к хранилищу данных в методах реализуются через поле класса EntityManager – оно является менеджером сущностей и помечается аннотацией @PersistenceContext. Эта аннотация предназначена для автоматического связывания менеджера сущностей с классами сущностей доменного слоя. Непосредственно конфигурирование менеджера сущностей реализуется посредством настройки фабрики сессий при помощи сущности *LocalContainerEntityManagerFactoryBean*. Сами запросы к базе данных написаны на языке HQL.

Hibernate Query Language (HQL) – платформо-независимый объектно-ориентированный язык запросов библиотеки Hibernate. HQL используется для написания запросов к сущностям, хранящимся в реляционной базе данных. HQL во многом похож на SQL, но в отличие от последнего, оперирует запросами, составленными по отношению к классам сущностей доменного слоя, в отличие от прямых запросов к таблицам базы данных.

Классы слоя доступа к данным включают в себя как обобщённые наборы действий для всех сущностей, так и более специализированные и характерные только для одной. Ниже перечислены назначения реализаций:

– AbstractJpaDAO является параметризованным абстрактным классом для обработки общих наборов действий для сущностей, таких как получение сущности по идентификатору, получение всех сущностей в системе, создание, редактирование, удаление и получение количества сущностей в системе, а также работа со свойствами сущностей;

– PIssueDao содержит специализированные действия по работе с дефектами, которые не входят в общий набор: получение списка дефектов, в которых пользователь назначен ответственным за исправление, получение сортированного списка дефектов, методы для получения статуса в зависимости от текущего статуса дефекта и метод прикрепления файла к дефекту;

– PProjectJpaDao выполняет специализированные действия по работе с проектами: получение списка версий сборки конкретного проекта, получение сортированного и несортированного списка проектов, а также получение количества проектов в системе;

– PUserDao обрабатывает специализированные действия по работе с пользователями, какие как получение пользователя по логину/паролю или электронной почте, получение результата поиска пользователей.

## 4.6 Архитектурный доменный слой

Доменный слой – это слой для размещения всех классов, описывающих простые сущности системы. Иногда для обозначения этих классов используют термин POJO (Plain Old Java Object) – простой Java-класс, не унаследованный от какого-то специфического класса и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели. В системе доменный слой представлен классами, которые располагаются в пакете by.gsu.bugtracker.domain.

Все сущности системы, которые хранятся в базе данных, унаследованы от класса GenericDomainObject, который хранит в себе поле идентификатора, переопределённые методы hashCode, equals и toString. Данный класс помечен аннотацией @MappedSuperclass, которая определяет класс, описанные свойства и методы которого будут применены в классах-наследниках, но этот класс не будет отображаться на отдельную таблицу. Классы наследники помечены аннотациями @Entity и @Table – они нужны для указания JPA на хранение этих классов в хранилище данных. Поля классов наследников помечаются аннотациями @Column, а в некоторых случаях @Size, @Enumerated, @Pattern, @JoinColumn, @ManyToOne, @OneToMany – они нужны для указания на столбцы таблиц в хранилище, указания корректных размеров строк для проверки, указания на перечисление, для проверки данных регулярным выражением, для указания объединений таблиц, а также типов связей между таблицами.

## 4.7 Сервер базы данных

При разработке системы использована база данных MySQL 5.5, однако система поддерживает большинство реляционных баз данных посредством указания сущностей *LocalContainerEntityManagerFactoryBean и dataSource* в конфигурационных файлах Spring, а также указания требуемого драйвера для подключения к базе данных в файле pom.xml в системе сборки Maven.

Инициализация всех требуемых таблиц (Рисунок 4.3) для работы системы, благодаря использованию библиотеки Hibernate, происходит автоматически при первом запуске сервера.

Однако требуется первичное заполнение начальными данными таблиц приоритетов, резолюций, статусов и типов дефектов, а также создания хотя бы одного пользователя с правами администратора. Скрипт для первичного заполнения данными:

INSERT INTO priorities (NAME) VALUES ('Критический');

INSERT INTO priorities (NAME) VALUES ('Важный');

INSERT INTO priorities (NAME) VALUES ('Обычный');

INSERT INTO priorities (NAME) VALUES ('Незначительный');

INSERT INTO resolutions (NAME) VALUES ('Дефект исправен');

INSERT INTO resolutions (NAME) VALUES ('Задача не имеет смысла');

INSERT INTO resolutions (NAME) VALUES ('Нельзя исправить');

INSERT INTO resolutions (NAME) VALUES ('Не воспроизводится');

INSERT INTO statuses (NAME) VALUES ('Новый');

INSERT INTO statuses (NAME) VALUES ('Назначен исполнитель');

INSERT INTO statuses (NAME) VALUES ('В процессе исправления');

INSERT INTO statuses (NAME) VALUES ('Решён');

INSERT INTO statuses (NAME) VALUES ('Закрыт');

INSERT INTO statuses (NAME) VALUES ('Повторно открыт');

INSERT INTO types (NAME) VALUES ('Косметическая ошибка');

INSERT INTO types (NAME) VALUES ('Основная ошибка');

INSERT INTO types (NAME) VALUES ('Ошибка функциональности');

INSERT INTO types (NAME) VALUES ('Ошибка производительности');

INSERT INTO users (EMAILADDRESS, FIRSTNAME, LASTNAME, PASSWORD, USERROLE)

VALUES ('gordon.frimen.bm@mail.ru', 'Гордон', 'Фримен',

'gordon.frimen.bm', 'ADMINISTRATOR');

# cheme2_750px.png

**Рисунок 4.3 - Визуальная схема таблиц базы данных со связями**

# 5 Описание работы приложения

В описании работы приложения перечислены общий функциональный обзор системы, роли пользователей, работа с дефектами и их атрибутами, а также проектами.

## 5.1 Функциональный обзор

Пользователи приложения имеют возможность:

– добавить дефект, указав требуемые параметры и описание;

– назначить человека, который будет ответственен за исправление дефекта;

– добавить комментарий к дефекту;

– закрыть дефект после его исправление и верификации;

– открыть дефект снова, в случае его повторного обнаружения;

– выполнить поиск;

– получать оповещения, о каких либо изменениях в дефекте, который они добавили либо на который они назначены как ответственные за исправление/проверку;

– аутентифицироваться в приложении;

– редактировать свои собственные данные;

– редактировать данные других пользователей;

– добавлять и редактировать Статусы, Приоритеты, Типы, Проекты.

## 5.2 Роли пользователей в системе

В приложении есть несколько ролей определяющих возможности пользователя:

– Администратор это роль, которая позволяет пользователю просматривать любой дефект, добавлять дефект, редактировать любой дефект, закрывать дефект, повторно открывать дефект, добавлять комментарии к дефекту, выполнять поиск, получать оповещения, добавлять и редактировать данные пользователей, добавлять и редактировать любые второстепенные данные;

– Пользователь это роль, которая позволяет пользователю просматривать любой дефект, добавлять дефект, редактировать любой дефект, закрывать дефект, повторно открывать дефект, добавлять комментарии к дефекту, выполнять поиск, получать оповещения, редактировать свои собственные данные;

– Гость это роль, которая позволяет пользователю просматривать любой дефект, выполнять поиск.

В приложении может быть неограниченное количество пользователей с ролями Гость, Пользователь и Администратор. Роли назначает пользователь с ролью Администратор. Администратор не может назначить пользователю роль Гость. У пользователей может быть только одна роль. До тех пор, пока пользователь не аутентифицируется в приложении, ему автоматически назначается роль Гость. После аутентификации пользователю назначается роль, ранее определенная пользователем с ролью Администратор. Таблица привилегий пользователей в зависимости от роли в приложении (Таблица 5.1):

**Таблица 5.1 – Таблица привилегий ролей пользователей**

|  |  |  |  |
| --- | --- | --- | --- |
| Привилегии / Роли | Гость | Пользователь | Администратор |
| Просмотр дефекта | Доступно | Доступно | Доступно |
| Редактирование дефекта | Не доступно | Доступно | Доступно |
| Добавление дефекта | Не доступно | Доступно | Доступно |
| Добавление комментариев | Не доступно | Доступно | Доступно |
| Добавление файлов | Не доступно | Доступно | Доступно |
| Закрытие дефекта | Не доступно | Доступно | Доступно |
| Повторное открытие  дефекта | Не доступно | Доступно | Доступно |
| Назначение дефекта | Не доступно | Доступно | Доступно |
| Поиск дефектов | Доступно | Доступно | Доступно |
| Поиск пользователей | Не доступно | Не доступно | Доступно |
| Получение оповещений | Не доступно | Доступно | Доступно |
| Редактирование собственных пользовательских данных | Не доступно | Доступно | Доступно |
| Просмотр данных всех пользователей | Не доступно | Не доступно | Доступно |
| Редактирование данных всех пользователей | Не доступно | Не доступно | Доступно |
| Добавление пользователя | Не доступно | Не доступно | Доступно |
| Добавление проектов | Не доступно | Не доступно | Доступно |
| Редактирование проектов | Не доступно | Не доступно | Доступно |
| Редактирование статусов | Не доступно | Не доступно | Доступно |
| Добавление резолюций | Не доступно | Не доступно | Доступно |
| Редактирование резолюций | Не доступно | Не доступно | Доступно |
| Добавление приоритетов | Не доступно | Не доступно | Доступно |
| Редактирование приоритетов | Не доступно | Не доступно | Доступно |
| Добавление типов | Не доступно | Не доступно | Доступно |
| Редактирование типов | Не доступно | Не доступно | Доступно |

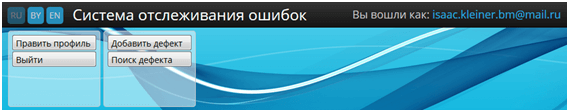
## 5.3 Главная страница системы

Начальная страница системы содержит следующие элементы:

– меню пользователя;

– подменю администратора;

– список дефектов.



**Рисунок 5.1 – Меню пользователя с ролью Пользователь.**

Меню пользователя (Рисунок 5.1) включает в себя:

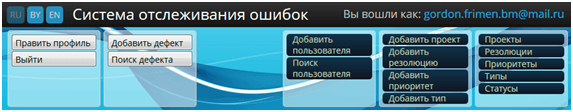
– Форму аутентификации пользователя;

– Ссылку на страницу редактирования пользовательских данных;

– Кнопку Добавить дефект ведущую на страницу создания дефекта;

– Кнопку Поиск дефекта ведущую на страницу поиска дефекта.

Кнопка Добавить дефект видна аутентифицированным пользователям.



**Рисунок 5.2 – Меню пользователя с ролью Администратор.**

Подменю администратора содержит (Рисунок 5.2):

– ссылки на страницы списков Проектов, Статусов, Резолюций, Приоритетов и Типов;

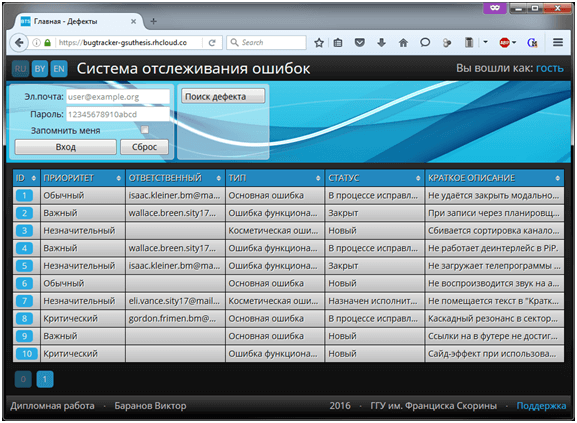
– ссылки на добавление Проектов, Резолюций, Приоритетов и Типов;

– ссылки на страницы поиска и добавления пользователя.

Подменю администратора видно только пользователю с ролью Администратора.

Список дефектов (Рисунок 5.3) представляет собой таблицу, которая содержит 10 назначенных дефектов в случае аутентифицированного пользователя, либо 10 последних добавленных дефектов. Если в приложении нет дефектов, удовлетворяющих требованию, то пользователю показано сообщение об отсутствии назначенных дефектов, либо об отсутствии дефектов в приложении соответственно.

Таблица содержит следующие столбцы: Идентификатор, Приоритет, Ответственный, Тип, Статус и Краткое описание. У пользователя есть возможность сортировать строки в таблице по любому столбцу. Значение в столбце Идентификатора представляет собой ссылку на страницу редактирования дефекта, либо на страницу просмотра дефекта в зависимости от роли пользователя.



**Рисунок 5.3 – Страница со списком дефектов.**

## 5.4 Работа с пользовательскими профилями

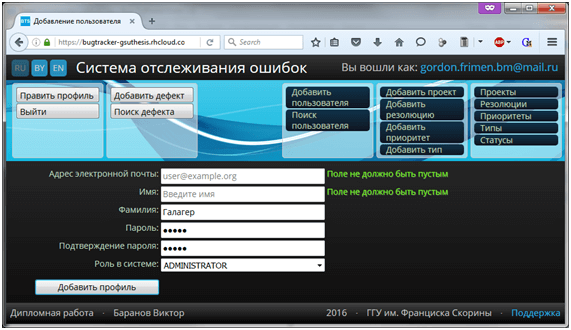
Только пользователь с ролью Администратор может добавлять пользователей. Для успешного добавления пользователя необходимо заполнить все обязательные поля в соответствии с индивидуальными правилами для каждого поля. В случае если Администратор не заполнил одно или более из обязательных полей, либо если одно или более правил для полей не выполняется, профиль пользователя не сохраняется, а приложение сообщает об этом Администратору, указав все незаполненные поля и ошибки. Если все поля заполнены корректно, то пользователь сохраняется и происходит перенаправление на страницу со списком дефектов.

Администратор может редактировать свои собственные данные и данные других пользователей, включая данные других Администраторов. Администратор может изменить пароль другого пользователя, но сам пароль просматривать не может.

Пользователь с ролью Пользователя может редактировать только свои собственные данные.

Для успешного обновления пользовательских данных необходимо заполнить все обязательные поля в соответствии с индивидуальными правилами для каждого поля, если такие имеются. В случае если не заполнено одно или более из обязательных полей, либо если одно или более правил для полей не выполняется, пользовательские данные не обновляются, а приложение сообщает об этом, указав все незаполненные поля и ошибки (Рисунок 5.4).

Если все поля заполнены корректно, то данные обновляются, и происходит перенаправление на главную страницу со списком дефектов.



**Рисунок 5.4 – Страница добавления пользователя**

Страница добавления пользователя содержит следующие поля:

– Имя пользователя – текстовое поле;

– Фамилия пользователя – текстовое поле;

– Адрес электронной почты пользователя – текстовое поле;

– Роль пользователя в системе – выпадающий список;

– Пароль пользователя – текстовое поле;

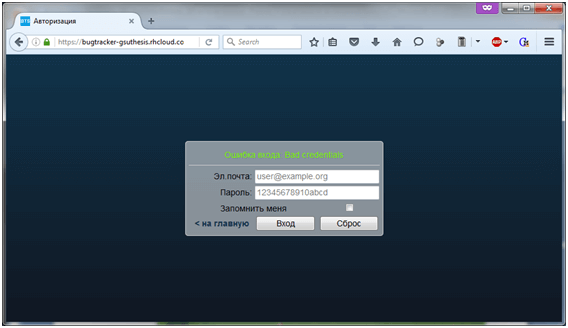
– Подтверждение пароля – текстовое поле.

Все перечисленные поля обязательны для заполнения. Содержание полей Пароль и Подтверждение пароля скрыто.

Пароль должен состоять не менее чем из 5 символов и может содержать буквы английского алфавита в верхнем и нижнем регистрах, цифры, знаки препинания, а также знаки @ % $.

Страница редактирования пользовательских данных пользователем с ролью Администратора содержит те же поля, что и при добавлении пользователя. Все поля так же обязательны для заполнения.

Для аутентификации пользователь должен ввести свой адрес электронной почты и пароль. Форма аутентификации пользователя присутствует на любой странице, в случае если пользователь не аутентифицирован. В случае ошибочного ввода данных, происходит перенаправления на отдельную страницу авторизации (Рисунок 5.5). После аутентификации форма заменяется приветственным сообщением содержащим имя пользователя и ссылку на страницу редактирования пользовательских данных и смену пароля, и кнопку выхода из системы.



**Рисунок 5.5 – Страница авторизации с указанием ошибки входа**

## 5.5 Работа с дефектами

Перед описанием работы с дефектами в системе, нужно уточнить непосредственное значение термина.

Дефект (или ошибка) – любое состояние системы, которое не соответствует ожидаемому поведению, основанному на спецификациях к проекту, требований, проектной документации, пользовательской документации, стандартов или исходя из чьего-либо восприятия, опыта и здравого смысла. Дефекты бывают разных классификаций в зависимости от вида тестирования. При обнаружении дефекта пользователи создают отчёты о дефекте.

Отчёт о дефекте (Англ. Bug report) – документ, описывающий и устанавливающий приоритет обнаруженного дефекта, а также содействующий его устранению. Как следует из самого определения, отчёт о дефекте пишется со следующими основными целями:

– предоставить информацию о проблеме – уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы;

– установить приоритет исправления проблемы – определить степень опасности проблемы для проекта;

– содействовать устранению проблемы – качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

### 5.5.1 Добавление дефекта

Добавить дефект может любой аутентифицированный пользователь. Страница добавления дефекта содержит следующие поля (Рисунок 5.6):

– Краткое описание дефекта – текстовое поле;

– Расширенное описание дефекта – текстовое поле;

– Текущий статус дефекта – выпадающий список;

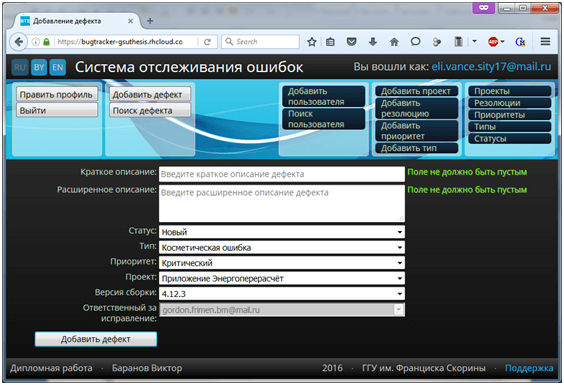
– Тип дефекта – выпадающий список;

– Приоритет дефекта – выпадающий список;

– Проект, в котором был найден дефект – выпадающий список;

– Версия сборки проекта – выпадающий список;

– Пользователь ответственный за исправление – выпадающий список.

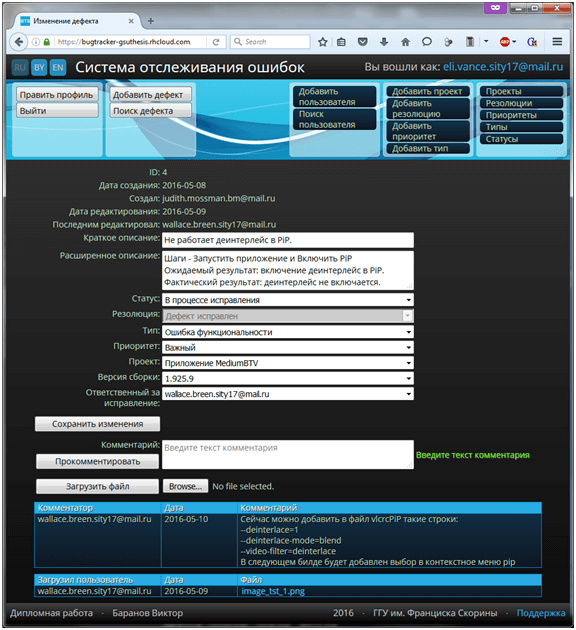


**Рисунок 5.6 – Страница добавления дефекта**

Все перечисленные поля, кроме поля Пользователя ответственного за исправление, обязательны для заполнения. В выпадающем списке ответственных перечислены все пользователи, добавленные в приложение. В качестве ответственного за исправление пользователь может указать сам себя.

### 5.5.2 Редактирование дефекта

Редактировать дефект в системе может любой аутентифицированный пользователь.



**Рисунок 5.7 – Страница редактирования дефекта**

Страница редактирования дефекта содержит следующие поля (Рисунок 5.7):

– Уникальный идентификатор дефекта – не редактируемое поле;

– Дата создания дефекта – не редактируемое поле, которое заполняется автоматически один раз при добавлении дефекта;

– Пользователь, добавивший дефект – не редактируемое поле, которое заполняется автоматически один раз при добавлении дефекта;

– Дата последнего редактирования дефекта – не редактируемое поле, которое заполняется автоматически каждый раз при редактировании дефекта;

– Пользователь, редактировавший дефект – не редактируемое поле, которое заполняется автоматически каждый раз при редактировании дефекта;

– Краткое описание дефекта – текстовое поле;

– Расширенное описание дефекта – текстовое поле;

– Текущий статус дефекта – выпадающий список;

– Резолюция, с которой закрыт дефект – выпадающий список;

– Тип дефекта – выпадающий список;

– Приоритет дефекта – выпадающий список;

– Проект, в котором был найден дефект – выпадающий список;

– Версия сборки проекта – выпадающий список;

– Пользователь ответственный за исправление дефекта – выпадающий список.

Все перечисленные редактируемые поля, за исключением полей Ответственного и Резолюции, обязательны для заполнения. Поле Резолюция обязательно заполнения только в случае закрытия дефекта и до этого момента должно оставаться не редактируемым. Поле Ответственного за исправление обязательно заполнения только в случае если статус дефекта меняется на назначен ответственный за исправление и далее по жизненному циклу дефекта. В выпадающем списке Ответственных перечислены все пользователи, добавленные в приложение. В качестве ответственного за исправление пользователь может указать сам себя.

Так же на странице редактирования дефекта есть секция с добавлением комментариев и все ранее добавленные комментарии. Комментарий может добавить любой аутентифицированный пользователь. Количество комментариев на странице редактирования дефекта неограниченно. Секция добавление комментария содержит только одно необязательное текстовое поле Текст комментария.

Пользователь не может редактировать и удалять комментарии. Комментарии на странице отображаются в режиме только для чтения и в порядке добавления, начиная от самого раннего.

Каждый добавленный комментарий содержит следующие поля:

– Пользователь, добавивший комментарий – не редактируемое поле, которое заполняется автоматически один раз при добавлении комментария;

– Дата добавления комментария – не редактируемое поле, которое заполняется автоматически один раз при добавлении комментария;

– Текст комментария – не редактируемое поле, которое заполняется автоматически один раз при добавлении комментария.

Любой аутентифицированный пользователь имеет возможность прикрепить файл к дефекту. На странице редактирования дефекта есть секция с добавлением файлов и все ранее добавленные файлы. Файл может добавить любой аутентифицированный пользователь. Количество файлов на странице редактирования дефекта неограниченно.

Пользователь не может удалять прикрепленные файлы. Прикрепленные файлы на странице отображаются в виде ссылок на сами файлы и в порядке добавления, начиная от самого раннего.

Каждый добавленный файл содержит следующие поля:

– Пользователь, добавивший файл – не редактируемое поле, которое заполняется автоматически один раз при добавлении файла;

– Дата добавления файла – не редактируемое поле, которое заполняется автоматически один раз при добавлении файла;

– Ссылка на скачивание файла, содержащая имя файла.

Пользователи, которые не аутентифицированы в системе могут видеть список файлов, но прав на скачивание у них нет.

### 5.5.3 Жизненный цикл дефектов в системе

Дефект в системе имеет следующий жизненный цикл:

– Создание дефекта;

– Редактирование дефекта;

– Закрытие дефекта;

– Повторное открытие дефекта.

Для создания дефекта пользователь должен нажать на кнопку Добавить дефект. В результате этого действия пользователь переходит на страницу добавления дефекта. Для успешного добавления дефекта пользователю необходимо заполнить все обязательные поля. В поле Статус для выбора доступно два значения: Новый и Назначен исполнитель. В случае если пользователь заполняет поле Назначен исполнитель, то он должен выбрать ответственного за исправление из выпадающего списка. Для того чтобы завершить процесс добавления дефекта пользователю необходимо нажать на кнопку Добавить дефект. Если пользователь не заполнил одно или более из обязательных полей, то дефект не должен быть сохранен, а приложение должно сообщить об этом пользователю указав все незаполненные поля. Если все поля заполнены корректно, то дефект сохраняется и пользователь перенаправляется на главную страницу.

Для редактирования пользователь должен перейти на страницу редактирования дефекта. На странице редактирования пользователь может изменить значения во всех доступных для редактирования полях. Поля доступные для редактирования обязательны для заполнения. Значение в поле Статус можно изменить с Назначен исполнитель на В процессе исправления, чтобы указать, что пользователь ответственный за исправление дефекта начал работу. Другие значения для поля Статус недоступны. Для того чтобы завершить процесс редактирования дефекта пользователю необходимо нажать на кнопку Сохранить изменения. Если пользователь не заполнил одно или более из обязательных полей, то изменения не сохраняются, а приложение сообщает об этом пользователю указав все незаполненные поля. Если все поля заполнены корректно, то все изменения сохраняются и пользователь перенаправляется на главную страницу.

Также во время редактирования дефекта пользователь может добавить комментарий или прикрепить файл.

Для закрытия пользователь должен перейти на страницу редактирования дефекта. На странице редактирования пользователь может изменить значения во всех доступных для редактирования полях. Поля доступные для редактирования обязательны для заполнения. Значение в поле Статус можно изменить на Решён или Закрыт, чтобы указать, что дефект исправлен. Так же для заполнения поля Статус доступно значение из предыдущего жизненного цикла, т.е. В процессе исправления. Другие значения для поля Статус недоступны. Дефект считается закрытым только в случае при значении Закрыт в поле Статус. Если пользователь выбирает значение Решён или Закрыт в поле Статус, то поле Резолюция становится редактируемым и обязательным для заполнения. Для того чтобы завершить процесс закрытия дефекта пользователю необходимо нажать на кнопку Сохранить изменения. Если пользователь не заполнил одно или более из обязательных полей, то изменения не сохраняются, а приложение сообщает об этом пользователю, указав все незаполненные поля. Если все поля заполнены корректно, то все изменения сохраняются и приложение сообщает об успешном завершении операции. Также во время закрытия дефекта пользователь может добавить комментарий или прикрепить файл.

После того как дефект закрыт. Все его поля, кроме поля Статус, становятся не редактируемыми. В поле Статус для выбора остаются только два значения: текущее Закрыт и доступное для выбора Повторно открыт.

Для того чтобы повторно открыть дефект пользователь должен перейти на страницу просмотра дефекта. На этой странице все поля, кроме поля Статус, не редактируемые. В поле Статус для выбора есть только два значения: текущее Закрыт и необходимое для повторного открытия Повторно открыт. Дефект считается повторно открытым только в случае, если его статус изменен на Повторно открыт. Для того чтобы завершить процесс редактирования дефекта пользователю необходимо нажать на кнопку Сохранить изменения.

После повторного открытия жизненный цикл дефекта повторяется.

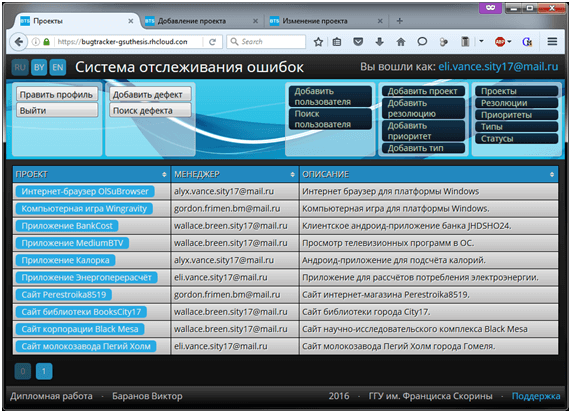
## 5.6 Работа с проектами

Список проектов, так же как и страницы добавления и редактирования проектов, доступны лишь пользователям с ролью Администраторов.

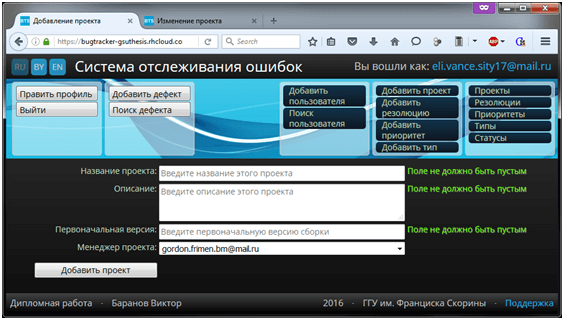
На странице со списком проектов отображены в виде таблицы все проекты системы (Рисунок 5.8).

Таблица содержит следующие колонки: Проект, Менеджер и Описание. Значение в колонке Проект представляет собой ссылку на страницу редактирования проекта.

На одной странице в таблице отображается только 10 проектов. Для перехода между страницами под таблицей есть соответствующие элементы в виде номеров страниц для пагинации.



**Рисунок 5.8 – Страница со списком проектов**



**Рисунок 5.9 – Страница добавления проекта**

Страница добавления проектов (Рисунок 5.9) содержит поля:

– Название проекта – текстовое поле;

– Описание проекта – текстовое поле;

– Версия сборки – текстовое поле;

– Менеджер проекта – выпадающий список.

Поле версии сборки позволяет указать начальную сборку проекта.

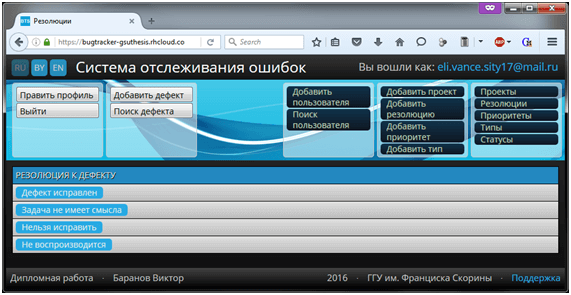
При редактировании проекта у пользователя имеется возможность добавлять другие версии сборки.

## 5.7 Работа с атрибутами дефектов

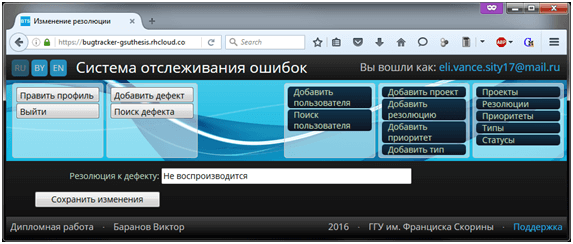
В системе поддерживаются различные атрибуты дефектов: резолюции, приоритеты, типы и статусы. Все, кроме последнего, поддерживают добавление новых атрибутов для расширения возможностей.

### 5.7.1 Атрибут резолюции к дефектам

На странице со списком резолюций, в табличном виде, перечислены все имеющиеся в приложении резолюции к дефектам (Рисунок 5.10).



**Рисунок 5.10 – Страница списка всех резолюций к дефектам в системе**



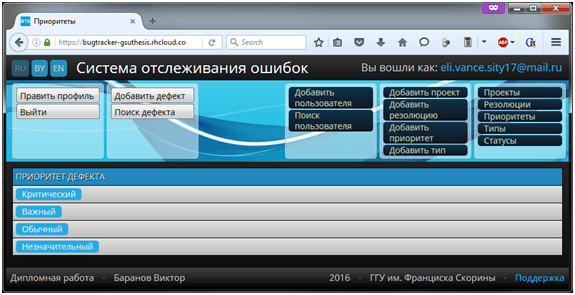
**Рисунок 5.11 – Страница редактирования резолюции к дефекту**

Таблица содержит лишь одну колонку – Резолюция к дефекту. Значение в колонке представляет собой ссылку на страницу редактирования резолюции. Список резолюций, так же как и страница добавления и редактирования резолюции, доступны лишь пользователям с ролью Администратора.

Страница с добавлением и редактированием резолюций (Рисунок 5.11) содержит лишь одно поле обязательное для заполнения.

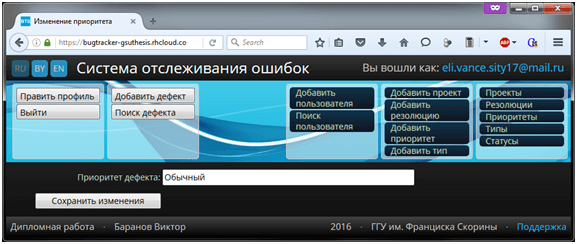
### 5.7.2 Атрибут приоритета дефектов

На странице со списком приоритетов в табличном виде перечислены все имеющиеся в приложении приоритеты дефектов (Рисунок 5.12).



**Рисунок 5.12 – Страница списка всех приоритетов дефектов в системе**

Таблица содержит лишь одну колонку – Приоритет дефекта. Значение в колонке представляет собой ссылку на страницу редактирования приоритета. Список приоритетов, так же как и страница добавления и редактирования приоритетов, доступны лишь пользователям с ролью Администратора.



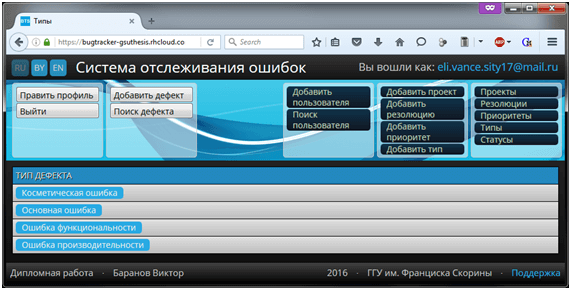
**Рисунок 5.13 – Страница редактирования приоритетов дефектов**

Страница с добавлением и редактированием приоритетов (Рисунок 5.13) содержит лишь одно поле обязательное для заполнения.

### 5.7.3 Атрибут типа дефектов

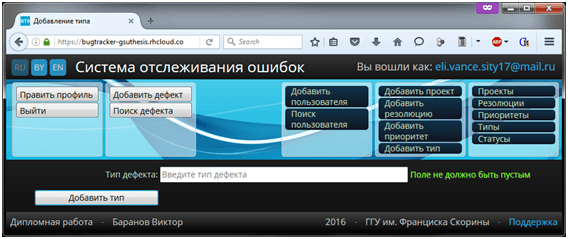
На странице со списком типов в табличном виде перечислены все имеющиеся в приложении типы дефектов (Рисунок 5.14).

Таблица содержит лишь одну колонку – Тип дефекта. Значение в колонке представляет собой ссылку на страницу редактирования типа. Список типов, так же как и страница добавления и редактирования типов, доступны лишь пользователям с ролью Администратора.



**Рисунок 5.14 – Страница списка всех типов дефектов в системе**

Страница с добавлением (Рисунок 5.15) и редактированием типов содержит лишь одно поле обязательное для заполнения.



**Рисунок 5.15 – Страница добавления типов дефектов**

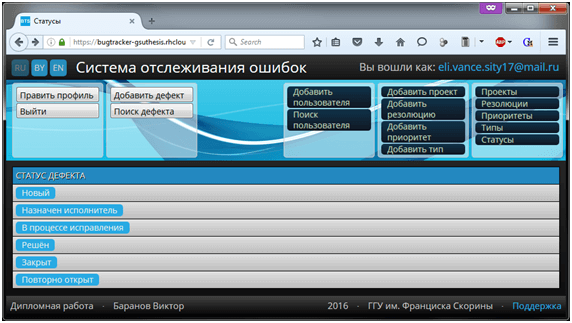
### 5.7.4 Атрибут статуса дефектов

На странице со списком статусов в табличном виде перечислены все имеющиеся в приложении статусы дефектов (Рисунок 5.16).

Таблица содержит лишь одну колонку – Статус дефекта. Значение в колонке представляет собой ссылку на страницу редактирования статуса. Список статусов, так же как и страница редактирования, доступны лишь пользователям с ролью Администратора.

В отличие от предыдущих атрибутов, статус дефекта не поддерживает добавления новых значений, так как он напрямую связан с жизненным циклом дефекта и добавление новых состояний затронет внутреннюю логику работы системы.

Страница с редактированием статусов содержит лишь одно поле обязательное для заполнения.



**Рисунок 5.16 – Страница списка всех статусов дефектов в системе**

# Заключение

В ходе выполнения дипломной работы была спроектирована архитектура приложения и её базе разработана система отслеживания ошибок. Также изучены технологический стек платформы Java EE, библиотека Hibernate и фреймворк автоматизации сборки Maven.

Задача выполнена полностью, система может использоваться небольшими командами разработчиков программного обеспечения.

# Список используемой литературы

1 Пакуров, К.Ф. Программирование на языке Java / К.Ф. Пакуров. – СПб.: Питер, 2013. – 480 с.

2 Вольская, А.М. Архитектура корпоративных приложений / А.М. Вольская. – М.: Вильямс. – 2014. – 565 с.

3 Капер, А. Шаблоны проектирования: Справ. Пособие / А. Капер, М. Струф. – Мн.: Выш. шк., 2010. – 243 с.: ил.

4 Стипер, В. Введение в Spring / В. Стипер. – М.: Вильямс. – 2015. – 432 с.

5 Брин, В. Эффективная работа с Maven / В. Брин. – М.: Вильямс. – 2014. – 130 с.

6 Моварт, С. Современные инструменты разработки. Моварт. – Спб.: БХВ-Питербург. – 2014. – 445 с.

# Приложение A

**Описание сборки проекта в Maven**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>BugTracker</groupId>

<artifactId>BugTracker</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>/BugTracker</name>

<url>http://maven.apache.org</url>

<properties>

<java-version>1.7</java-version>

<org.springframework-version>3.2.8.RELEASE</org.springframework-version>

<org.springframework-security-version>3.2.4.RELEASE</org.springframework-security-version>

<org.hibernate-version>4.2.7.Final</org.hibernate-version>

<org.aspectj-version>1.7.4</org.aspectj-version>

<org.slf4j-version>1.6.6</org.slf4j-version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${org.springframework-version}</version>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${org.springframework-security-version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${org.springframework-security-version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>${org.springframework-security-version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-taglibs</artifactId>

<version>${org.springframework-security-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjrt</artifactId>

<version>${org.aspectj-version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${org.slf4j-version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>${org.slf4j-version}</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>${org.slf4j-version}</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>javax.inject</groupId>

<artifactId>javax.inject</artifactId>

<version>1</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.0.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<scope>runtime</scope>

<version>1.1.1</version>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>c</artifactId>

<version>1.1.1</version>

<scope>runtime</scope>

<type>tld</type>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>fmt</artifactId>

<version>1.1.1</version>

<scope>runtime</scope>

<type>tld</type>

</dependency>

<dependency>

<groupId>javax.mail</groupId>

<artifactId>mail</artifactId>

<version>1.4.7</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

</dependency>

<dependency>

<groupId>net.sf.json-lib</groupId>

<artifactId>json-lib</artifactId>

<version>2.4</version>

<classifier>jdk15</classifier>

</dependency>

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-core-asl</artifactId>

<version>1.9.13</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.2.5-b04</version>

</dependency>

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>org.hibernate.javax.persistence</groupId>

<artifactId>hibernate-jpa-2.0-api</artifactId>

<version>1.0.1.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${org.hibernate-version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>${org.hibernate-version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.1.1.Final</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.1.0.Final</version>

</dependency>

<dependency>

<groupId>net.wimpi</groupId>

<artifactId>telnetd-x</artifactId>

<version>2.1.1</version>

</dependency>

</dependencies>

<repositories>

<repository>

<id>java.net</id>

<url>http://download.java.net/maven/2/</url>

</repository>

</repositories>

<build>

<finalName>BugTracker</finalName>

<plugins>

<plugin>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<additionalProjectnatures>

<projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>

</additionalProjectnatures>

<additionalBuildcommands>

<buildcommand>org.springframework.ide.eclipse.core.springbuilder</buildcommand>

</additionalBuildcommands>

<downloadSources>true</downloadSources>

<downloadJavadocs>true</downloadJavadocs>

</configuration>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

# Приложение Б

**Конфигурационные файлы Spring**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:jee="http://www.springframework.org/schema/jee" xmlns:lang="http://www.springframework.org/schema/lang"

xmlns:p="http://www.springframework.org/schema/p" xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

<tx:annotation-driven/>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

</bean>

<bean id="messageSource"

class="org.springframework.context.support.ReloadableResourceBundleMessageSource">

<property name="basename" value="classpath:messages" />

<property name="defaultEncoding" value="UTF-8" />

</bean>

<bean id="propertyConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"

p:location="/WEB-INF/jdbc.properties" />

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource"

p:driverClassName="${jdbc.driverClassName}" p:url="${jdbc.databaseurl}"

p:username="${jdbc.username}" p:password="${jdbc.password}" />

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="packagesToScan" value="by.gsu.bugtracker.domain" />

<property name="jpaVendorAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />

</property>

<property name="jpaProperties">

<props>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<prop key="hibernate.show\_sql">false</prop>

<prop key="hibernate.dialect">${jdbc.dialect}</prop>

</props>

</property>

</bean>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver">

<property name="cookieName" value="bugtrackerPreferredLanguage"/>

<property name="defaultLocale" value="en\_US" />

<!-- 60 second \* 60 \* 24 \* 7 (7 days) -->

<property name="cookieMaxAge" value="604800"/>

</bean>

</beans>

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:jee="http://www.springframework.org/schema/jee" xmlns:lang="http://www.springframework.org/schema/lang"

xmlns:p="http://www.springframework.org/schema/p" xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

<context:annotation-config />

<context:component-scan base-package="by.gsu.bugtracker.dao.persistence" />

<context:component-scan base-package="by.gsu.bugtracker.service.impl" />

<import resource="data.xml" />

<import resource="security.xml" />

<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">

<property name="host" value="smtp.gmail.com"/>

<property name="port" value="587"/>

<property name="username" value="graduate.work.bugtrackerXXX@gmail.com" />

<property name="password" value="■■■■■■■■■■" />

<property name="javaMailProperties">

<props>

<prop key="mail.transport.protocol">smtp</prop>

<prop key="mail.smtp.auth">true</prop>

<prop key="mail.smtp.starttls.enable">true</prop>

<prop key="mail.debug">false</prop>

</props>

</property>

</bean>

<bean id="templateMessage" class="org.springframework.mail.SimpleMailMessage">

<property name="from" value="graduate.work.bugtracker@gmail.com" />

<property name="subject" value="Notification BugTrackerSystem" />

</bean>

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver"

p:maxUploadSize="50000000" />

</beans>

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.2.xsd">

<http access-denied-page="/error403.jsp" use-expressions="true">

<intercept-url pattern="/Admin/\*\*"

access="hasAnyAuthority('ADMINISTRATOR')" />

<intercept-url pattern="/Issues/\*"

access="hasAnyAuthority('ADMINISTRATOR,USER')" />

<intercept-url pattern="/Users/\*"

access="hasAnyAuthority('ADMINISTRATOR,USER')" />

<intercept-url pattern="/Issues\*"

access="hasAnyAuthority('USER,GUEST,ADMINISTRATOR')" />

<form-login login-page="/login.jsp"

authentication-failure-url="/login.jsp?error=true" />

<logout logout-url="/logout" logout-success-url="/" />

<anonymous username="guest" granted-authority="GUEST" />

<remember-me />

</http>

<authentication-manager>

<authentication-provider>

<jdbc-user-service id="userService"

data-source-ref="dataSource"

users-by-username-query="select emailaddress, password, true from users where emailaddress=?"

authorities-by-username-query="SELECT emailaddress, USERROLE FROM users WHERE emailaddress=?" />

</authentication-provider>

</authentication-manager>

</beans:beans>

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="by.gsu.bugtracker.web" />

<mvc:interceptors>

<bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">

<property name="paramName" value="lang" />

</bean>

</mvc:interceptors>

</beans>

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/mvc"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.springframework.org/schema/beans"

xsi:schemaLocation="

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<annotation-driven/>

<resources mapping="/resources/\*\*" location="/resources/" />

<beans:bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<beans:property name="prefix" value="/WEB-INF/views/" />

<beans:property name="suffix" value=".jsp" />

</beans:bean>

<beans:import resource="controllers.xml" />

</beans:beans>