ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ГАГАРИНА Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра «Прикладные информационные технологии»

Направление 09.03.02 «Информационные системы и технологии»

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка программного обеспечения поиска решения проблемы в базе знаний на основе информации в исключении Java приложения»

Студент (ка) _____ Мигачев Антон Алексеевич_____

фамилия, имя, отчество

курс 4 группа б2-ИФСТ41

Руководитель

| доц. каф. «ПИТ», кандидат технических наук, доцент | 11.06.20 | А. В. Ермаков |
|---|---|---|
| должность, ученая степень, уч. звание | подпись, дата | Инициалы Фамилия |

Допущен к защите
Протокол № 12 от «11» июня 2020 года

Зав. кафедрой «Прикладные информационные технологии»

кандидат технических наук, О.А. Торопова

| | | |
|---|---|---|
| доцент | 11.06.20 | |
| ученая степень, уч. звание | подпись, дата | Инициалы Фамилия |

Саратов 2020г

## Содержание

# ВВЕДЕНИЕ

На сегодняшний день процесс автоматизации занимает важную роль в любой IT – компании, от процессов разработки и внедрения программного обеспечения, до взаимодействия между командами внутри компании, особенно с большим количеством проектов и штатом сотрудников. Одним из таких процессов взаимодействия является тестирование ПО, поиск и исправление найденных ошибок после проведения тестов.

С увеличением числа проектов и их размера, процесс тестирования и исправления багов превращается в времязатратную и довольно рутинную работу. Во многом это происходит из–за увеличения числа функций, который нужно протестировать, а также возросшей сложностью проекта. Каждый функциональный элемент системы необходимо тщательно протестировать и, в случае каких-либо ошибок, сообщить об этом разработчикам. Исходя из этого возникает сразу несколько проблем, во-первых – необходимость в учёте и хранении проектов и описания тестовых сценариев для каждой части его функционала. Во-вторых, необходимость запуска тестовых сценариев с возможностью сбора необходимой системной информации на протяжении всего процесса тестирования. В-третьих – возможность автоматического создания отчёта и его анализа по итогам проведённого тестирования, поскольку велика вероятность, что схожая проблема уже когда-либо могла решаться. Таким

образом решение этих проблем принесло бы пользу как тестировщикам, поскольку для них будет предоставлен удобная система для хранения и запуска тестовых сценариев, а также генерации отчёта, так и разработчикам, поскольку такой отчет будет содержать всю необходимую информацию для решения проблемы. Исходя из этих фактов, можно сделать вывод об необходимости и актуальности разработки программного обеспечения, которое могло бы реализовать вышеописанные требования.

Целью выпускной квалификационной работы является разработка программного обеспечения автоматизации проведения тестовых сценариев, создания отчетов об ошибках и выполнения первичного анализа исходя из исключений, которые найдены в логах приложения.

Для достижения цели необходимо решить следующие задачи:

- Разработать программное обеспечение, которое позволит хранить и запускать тестовые сценарии для проектов IT-компании;
- Собирать информацию о работе приложения (логи) во время проведения тестирования;
- Генерировать отчёты об ошибках, с информацией о тестируемом ПО и проводимом тестовом сценарии;

- Анализировать отчёты об ошибках по найденным исключениям в логах и осуществлять поиск похожих проблем в корпоративной багтрекинг системе;
- Отправлять отчёты об ошибках в багтрекинг систему.

Это позволит значительно уменьшить время и трудозатраты как команды тестирования, так и команды разработки.

# 1.ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ

## 1.1 Введение в предметную область

Основными участниками процесса разработки и поддержки программного обеспечения являются команды тестирования и разработки. Как правило, эти команды работают совместно и процесс их взаимодействия выглядит следующим образом: в процессе проверки какого-либо тестового сценария, если результат проверки отличается от ожидаемого, то тестировщик сообщает об этом команде разработчиков, передавая информацию о найденной ошибке в виде специализированного отчёта в корпоративную систему отслеживания ошибок [1].

Система отслеживания ошибок (англ. bug tracking system) — программное обеспечение, главной целью которого является помощь участникам процесса разработки программного обеспечения (программистам, тестировщикам, специалистам поддержки и др.) в учёте и контроле ошибок (багов), найденных в разрабатываемых и эксплуатируемых программах, а также для мониторинга процессов устранения этих ошибок [2]. Также подобные программные продукты могут использоваться в качестве своеобразной

накопительной базы знаний о проекте, стадиях его разработки, поддержки и тестирования.

С развитием проекта и увеличением кодовой базы, растет и количество отчетов о найденных багах, как правило на основе таких отчетов составляется некий список типовых ошибок, которые в том или ином виде встречались на какой-либо стадии проекта. Для каждой ошибки разработчику необходимо выполнить первичный анализ, то есть выяснить не решалась ли такая задача ранее и есть ли для неё готовый ответ, найти возможные варианты решения проблемы, которые могут лежать на поверхности, а также собрать и предоставить всю необходимую информацию о проблеме для её дальнейшего исправления. Анализ производится исходя из данных, которые тестировщик передаёт разработчику в отчёте об ошибке, из них можно выделить следующие:

- кто сообщил о проблеме;
- дата и время, когда была обнаружена проблема;
- критичность проблемы;
- пользователь, для которого актуальна проблема;
- трассировка стека;
- скриншоты и др. вложения;
- конфигурация системы;

- настройки окружения, на котором работает программа;
- описание корректного и некорректного поведения программы.

Это лишь неполный список данных которые может хранить в себе типичная багтрекинг система, предоставляя при этом возможность отслеживания и мониторинга ошибок. На самом деле многие багтрэкинг системы позволяют вести намного более подробный учет проектной активности. В чем то, они напоминают системы управления проектами. А многие из них интегрированы с такими системами.

### 1.2 Проблемы анализа отчетов об ошибках в текущем процессе

Развитие компаний и программного обеспечения, которые они разрабатывают несут с собой дополнительные сложности по его поддержке. Результатами этого роста служит также увеличение и усложнение связей внутри компонентов ПО. Это приводит к возрастанию количества находимых багов и различных дефектов в системе. Таким образом, каждое такое отклонение от нормального поведения системы воспринимается её пользователями или тестировщиками как ошибка, о чём они и сообщают разработчикам в виде тикета в корпоративную багтрекинг систему. В результате этого со временем количество таких отчётов разрастается до значительных размеров и развитие программного продукта замедляется, поскольку основное

время, которым располагают разработчики должно быть направлено на устранение этих неисправностей и дефектов, чтобы удовлетворить требованиям качества, предъявляемым к программному обеспечению.

Процесс исправления найденной ошибки проходит следующим образом: разработчик, которому поручено заниматься проблемой, в первую очередь обращает внимание на тот объём информации, который был предоставлен ему для её анализа. С точки зрения разработчика, это довольно рутинный и трудоёмкий процесс. В каждом отчёте об ошибке содержится некоторый набор информации, который помогает разработчику как можно быстрее понять в чём заключается проблема и лишь после этого перейти к её устранению. Как правило этот процесс может занимать довольно большой промежуток времени. Разработчику необходимо учесть все факторы, которые каким-либо образом связаны с проблемой. Наиболее полезными из них являются записи в логах на момент воспроизведения ошибки, трассировка стека, версия приложения, программный модуль, в котором найдена ошибка в, а также скриншоты работы приложения. Все эти данные разработчику необходимо обработать и проанализировать вручную.

Если развитие проекта длится многие годы, то за такой длительный промежуток времени количество отчетов о багах достигает нескольких или даже десятков тысяч штук. Конечно, не каждый такой отчет действительно является

багом, как показывает личный опыт, лишь небольшой процент от общего числа тикетов (около 20%) требуют какого-либо отдельного разбирательства и дальнейшего внесения исправлений в программном коде. Ещё около 20% тикетов — различные проблемы с конфигурацией системы или настройками окружения, на котором запущено программное обеспечение. Таким образом более половины из всех поступающих тикетов к разработчикам с типом «Bug» зачастую являются либо проблемами, которые уже когда-либо встречались на одной из стадий жизненного цикла программного продукта, либо проблемами, которые так или иначе похожи на предыдущие или имеют какие-то общие черты.

Таким образом, большая часть времени разработчика уходит на исправление проблем, для которых есть уже готовое решение. В многом это происходит потому, что все вышеописанные шаги разработчик вынужден проходить вручную: поднимать всю статистику встречаемых багов, сортировать их по версиям приложения и программным модулям, выбирать из них только подходящие тестовые сценарии, сравнивать полученные результаты, анализировать логи, искать какие-либо закономерности в трассировке стека.

## 1.3 Обзор функциональности существующих систем

Поскольку процесс анализа довольно рутинный, трудоемкий и времязатратный, то встаёт вопрос о возможности его автоматизации. Для того, чтобы выработать единый подход к реализации процесса автоматического анализа тикетов рассмотрим, как подобная проблема решается в текущих реалиях. Какие программные средства используются для взаимодействия команд разработки и тестирования, их плюсы, минусы и отличительные особенности, а также возможности по интеграции с внешними системами через программный интерфейс.

Работу любой IT-компании, занимающейся разработкой программного обеспечения, трудно представить без использования багтрекинговой системы – программного обеспечения для отслеживания ошибок. Рынок систем отслеживания ошибок довольно велик и может удовлетворить требованиям практически любой компании: от стартапов с небольшим количеством сотрудников, до больших компаний со штатом в десятки тысяч человек [3]. Именно багтрекинг системы зачастую являются основными рабочими инструментами тестировщиков и разработчиков. Рассмотрим наиболее известные системы и их функции на примере трех самых популярных багтрекинг систем — Jira от компании Atlassian, YouTrack, разработанная компанией Jetbrains и BugZilla, поддерживаемая Mozilla Foundation.

### 1.3.1 Система отслеживания ошибок Jira

«Jira» – коммерческая система отслеживания ошибок, предназначена для организации взаимодействия участников процесса разработки ПО, также может быть использована в качестве системы управления проектами. Разработана компанией «Atlassian»[4].

Основным элементом учета в системе является задача, которая содержит название проекта, тему тип, приоритет, компоненты, содержание и может расширяться дополнительными пользовательскими полями. Задача может редактироваться или изменять статус. «Jira» обладает возможностями конфигурации, например, определение собственных прав доступа для проекта, видимость и поведение полей и т. д. «Jira» поддерживает интерфейсы SOAP, XML-RPC и REST для интеграции с внешними системами, а также библиотеки для популярных языков программирования.

Основными преимуществами Jira являются:

- Управление ошибками, задачами, усовершенствованиями или любой проблемой;
- Дружелюбный пользовательский интерфейс;
- Отслеживание изменений, компонентов и версий;
- Настраиваемые панели управления и статистика в реальном времени;

- Управление правами доступа и широкие возможности по управлению пользователями, упор на безопасность;
- Легкость в эксплуатации и расширении и интеграции с другими системами;
- Возможность запуска на любых аппаратных средствах, операционных системах и платформах баз данных.

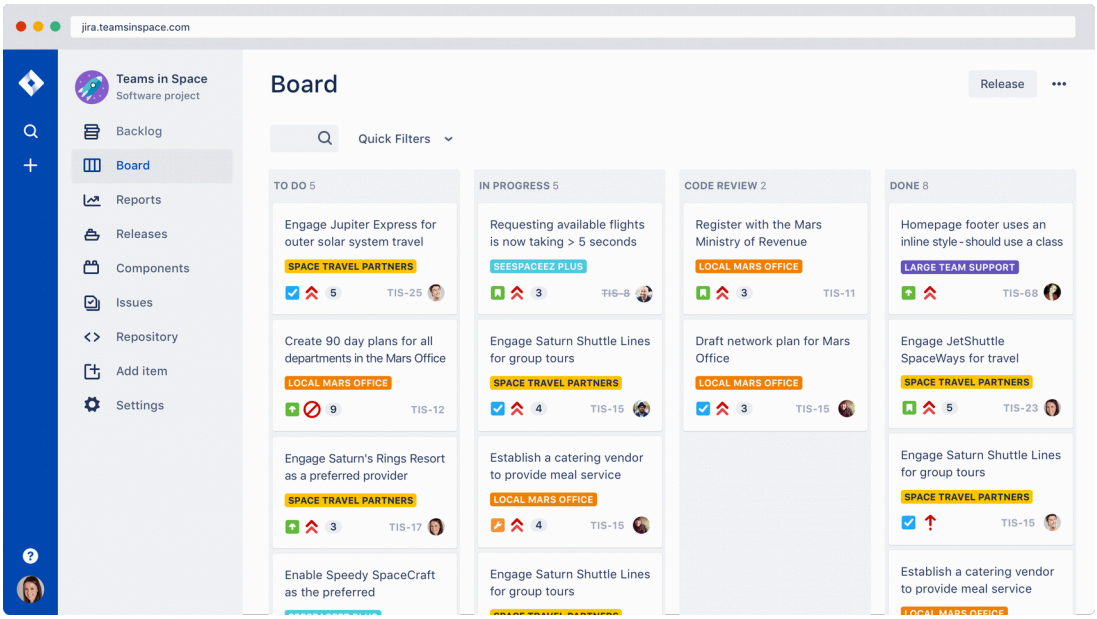


Рисунок 1.1 – Интерфейс Jira

Jira хранит в себе всю историю тикетов, тем самым выступая своеобразной базой знаний компании о её программном продукте, о том, как решались те или иные проблемы. Это является как большим плюсом — практически на любой вопрос можно найти ответ, так и минусом, поскольку значительно затрудняется навигация и ручной поиск в таком огромном объёме данных. Хотя в Jira и реализован довольно расширенный поиск, который позволяет произвести фильтрацию тикетов по типу, названию,

приоритету, а также проекту, но его часто бывает недостаточно или же поиск занимает много времени. Для того, чтобы исправить эту ситуацию разработчики из Atlassian предлагают использовать JQL — Jira Query Language. По сути, это отдельный языка запросов внутри Jira, который предлагает более гибкую возможность поиска тикетов в системе. JQL имеет большое количество операторов, ключевых слов и т.д., его можно сравнить с привычными структурированными языками запросов, а чём-то даже JQL превосходит их. Весь этот инструментарий безусловно идёт на пользу разработчику и немного облегчает процесс, но главной проблемой остаётся то, что этот процесс всё ещё выполняется вручную.  На рисунке 1.2 изображен интерфейс поиска Jira.

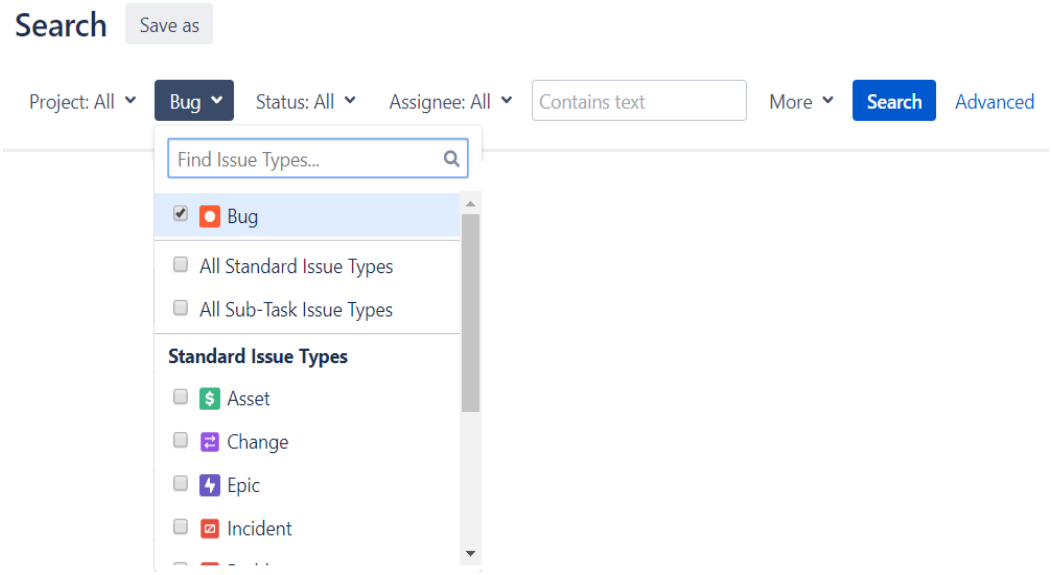


Рисунок 1.2. Диалоговое окно поиска

### 1.3.2 Система отслеживания ошибок и управления проектами YouTrack

YouTrack — программный продукт, представляющий собой систему отслеживания ошибок, разработанный компанией Jetbrains преимущественно для компаний, занимающихся разработкой и поддержкой программного обеспечения. Система переведена на множество языков, в том числе и на русский, предоставляет множество отчетов и осуществляет учет времени [5].

YouTrack предоставляет большое количество полезных функций, которые помогают выполнять различные действия программным путем через свой RESTful API [6]. Например, импортировать задачи из другой системы отслеживания ошибок – для более плавной миграции на YouTrack. программно создавать, изменять и выполнять другие операции – чтобы можно было легко интегрировать YouTrack в свою среду.

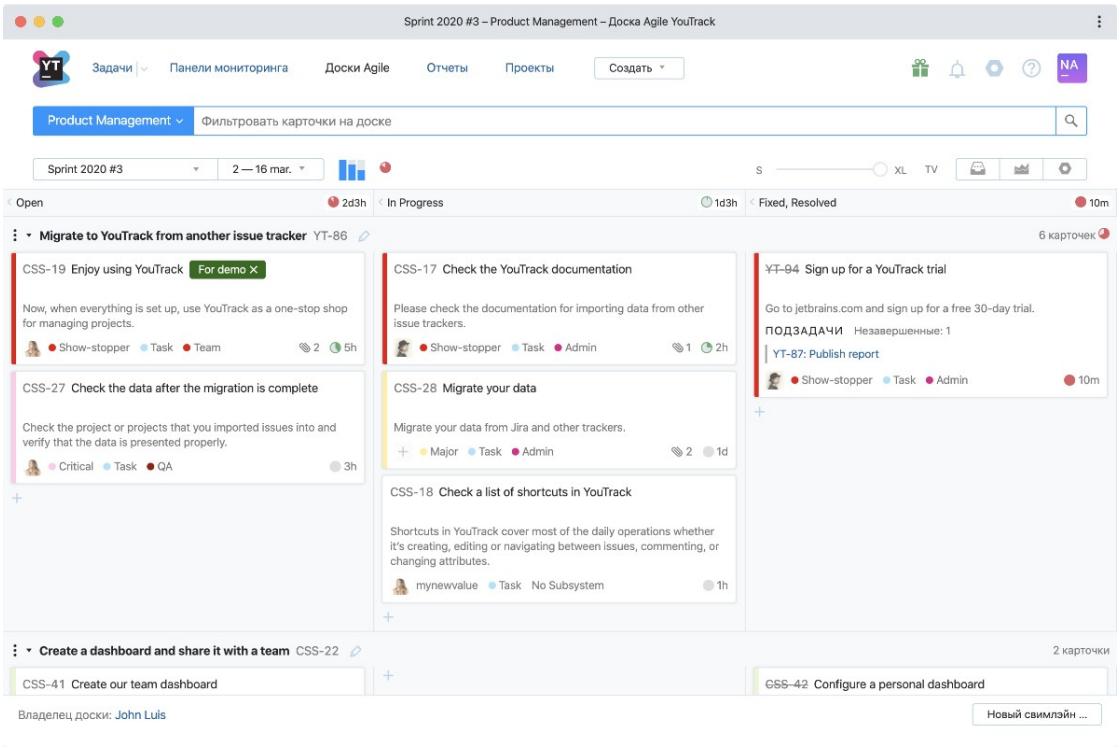


Рисунок 1.3. Интерфейс YouTrack

Через автоматическое предоставление задач от сторонних приложений. управлять проектами, пользователями, группами, ролями, типом ссылок и пользовательскими атрибутами. Стандартные интеграции YouTrack включают импорт из Jira, интеграции с электронными почтовыми ящиками, а также со многими системами контроля версий и хранения кода, такими как Github, Gitlab и BitBucket [7].

YouTrack имеет функцию интеллектуального поиска, которая позволяет пользователям быстро находить нужную информацию благодаря автодополнению. Он также предлагает короткие сокращения, которые значительно ускоряют рутинные процессы, что значительно повышает

эффективность и производительность. Также поддерживается возможность сортировки и фильтрации. На рисунке 1.4 изображен интерфейс поиска.

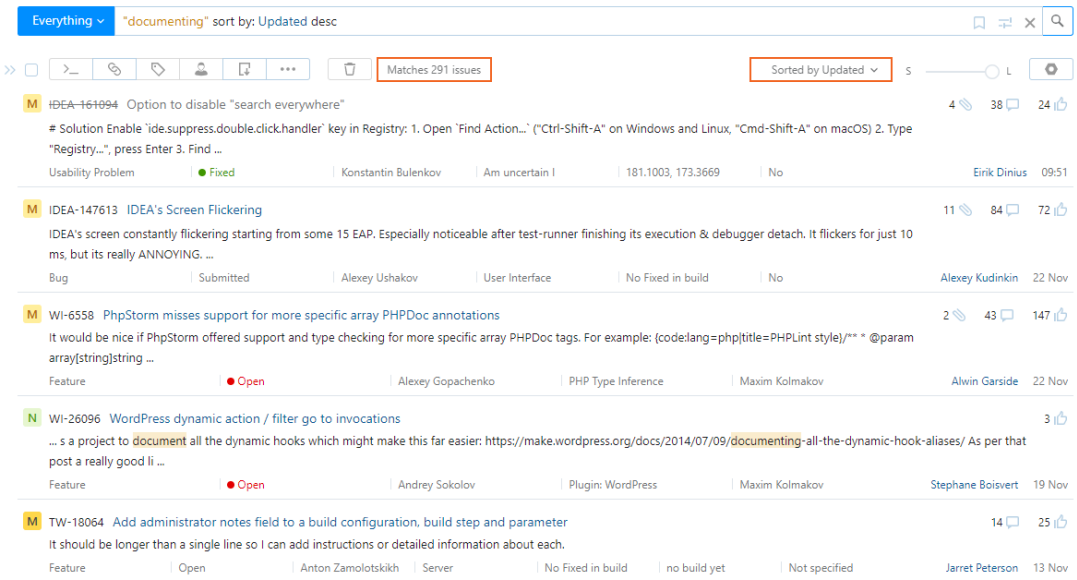


Рисунок 1.4. Диалоговое окно поиска

### 1.3.3 Свободная система отслеживания ошибок BugZilla

Ещё одной из наиболее популярных систем отслеживания ошибок является Bugzilla, выпущенная компанией Netscape. На данный момент разработкой и поддержкой занимается Mozilla Foundation [8]. На рынке систем отслеживания ошибок Bugzilla была одной из первых, поэтому многие наработки, которые были впервые реализованы в Bugzilla в дальнейшем использовались и

брались за основу в других системах багтрекинга, таких как Jira и YouTrack.

Сегодня BugZilla полностью бесплатна, а её исходный код доступен в открытом виде. Нередки случаи, когда какие-либо исправления или доработки осуществляются силами сообщества и пользователей этой системы.


Рисунок 1.5. Интерфейс BugZilla

В целом, функционала Bugzilla достаточно для покрытия большего количества сценариев использования багтрекинг системы, но всё же она значительно уступает вышеописанным системам, как и по удобству использования и наличию поддержки интеграций с различными сервисами.

Поиск по системе также значительно более простой и не поддерживает сложных поисковых запросов и различных условий для более релевантной поисковой выдачи.

Интерфейс BugZilla (рисунок 1.5) также заметно уступает конкурентам по своему удобству и внешнему виду, но в его плюсы можно отнести то, что он достаточно прост и не перегружен, а также легковесен и поддерживается во всех браузерах ранних версий.

### 1.4 Итоги сравнительного обзора

Если сравнивать вышеупомянутые системы отслеживания ошибок по удобству выполнения в них первичного анализа, то, в целом, все системы неплохо справляются с этой задачей, но Jira имеет одно очень важное конкурентное преимущество перед YouTrack и Bugzilla— собственный язык запросов JQL, который позволяет более гибко строить поисковые запросы, а также охватывать большее количество данных для их анализа. Также Jira обладает намного более широким списком API, используя который можно настроить интеграцию с внешними системами. Следует добавить, что обе этих функции прекрасно работают в паре, позволяя совмещать удобство интеграции с Jira, например, по REST и гибкость использования запросов через JQL [9].

Принимая во внимание эти факторы, можно сделать вывод о том, что на данный момент Jira — наиболее

подходящая система для реализации автоматического первичного анализа отчётов об ошибках, поскольку она совмещает в себе все необходимые функции, которые могут понадобиться для разработки такого решения.

# 2.ПРОЕКТИРОВАНИЕ СИСТЕМЫ ПЕРВИЧНОГО АНАЛИЗА ОШИБОК

Исходя из вышеописанного анализа предметной области, а также текущих процессов в организации взаимодействия между командами тестирования и разработки, создания и анализа отчётов об ошибках, сделан вывод о необходимости разработки программного обеспечения, позволяющего создавать и хранить в системе проекты и тестовые сценарии, а также осуществлять поиск похожих ошибок в базе знаний багтрекинг системы, формировать отчёт об ошибке и отправлять его в Jira.

Таким образом, автоматизация вышеописанных процессов позволит ускорить работу, а также увеличить продуктивность команд разработки и тестирования, поскольку основными элементами автоматизации со стороны тестировщика является удобный интерфейс создания, хранения и запуска тестовых сценариев, а также формирование отчета об ошибке. Со стороны разработчика автоматизируется и значительно ускоряется процесс анализа приходящих к нему отчетов об ошибках, поскольку на этапе создания отчета он сопровождается всей необходимой информацией для его анализа, а также осуществляется автоматический поиск схожих проблем по ошибкам, которые прикрепляются к отчету с которыми может ознакомиться разработчик.

## 2.1 Основные функциональные возможности предлагаемого ПО

- Управление и хранение проектов и тестовых сценариев в системе
- Синхронизация проектов с Jira
- Запуск и остановка тестовых сценариев
- Формирование отчёта о неудачном прохождении тестового сценария
- Поиск похожих проблем в системе Jira на основании ошибок, которые были обнаружены в логах работы приложения, а также ключевых слов
- Хранение отчёта в системе
- Отправка отчёта в Jira

## 2.2 Роли пользователей ПО и их разрешения на действия в системе

В разработанном ПО обеспечено разделение пользователей по системным группам и ограничение доступа к различным функциональным возможностям системы на основании принадлежности пользователей к той или иной группе в соответствии с предполагаемой моделью безопасности, которая будет использоваться при эксплуатации ПО. Система предусматривает принадлежность пользователей к одной из трёх системных групп: гость (guest), пользователь (user), администратор (admin). Для каждого пользователя из вышеуказанных групп действуют

соответствующие его группе разрешающие права на действия в системе.

Пользователи из группы Гость обладают разрешениями на следующие действия:

- Авторизация в системе
- Просмотр проектов
- Просмотр сценариев
- Просмотр тесткейсов
- Просмотр отчётов об ошибках
- Просмотр своего профиля

Группа Пользователи в дополнение к разрешениям, которыми обладают пользователи из группы Гость имеют доступ к следующим действия:

- Запуск тестового сценария
- Завершение тестового сценария
- Создание отчёта об ошибке
- Отправка отчёта об ошибке в Jira

Самым широким списком прав на действия, совершаемых в системе обладают пользователи из группы Администраторов. В дополнение к правам на действия, которыми обладает гость и обычный пользователь, администраторы обладают правами на управление проектами, сценариями и тесткейсами, изменения группы у пользователя, а также выполнению настройки системы посредством указания адреса Jira, а также учетных данных

для доступа к её API – логина и API Key. С полным списком можно ознакомиться ниже:

- Синхронизация проектов с Jira
- Создание проектов
- Удаление проектов
- Создание сценариев
- Удаление сценариев
- Создание тесткейсов
- Редактирование тесткейсов
- Удаление тесткейсов
- Изменение роли у пользователя
- Настройка интеграции с Jira

### 2.3 Входные и выходные данные в ПО

Программное обеспечение позволяет создавать и манипулировать следующими типами сущностей:

- Проект
- Сценарий
- Тесткейс
- Отчёт об ошибке
- Пользователь

Входными данными для сущности Проект являются:

- Название проекта
- Описание проекта
- Ключ проекта на Jira

Входными данными для сущности Сценарий являются:

- Название сценария
- Описание сценария

Входными данными для сущности Тесткейс являются:

- Название тесткейса
- Описание тесткейса
- Предпосылки, подготовительные шаги, настройки
- Тестовые шаги
- Ожидаемый результат

Входными данными для сущности Отчёт об ошибке являются:

- Название отчёта, тема
- Описание
- Предпосылки, подготовительные шаги, настройки
- Шаги по воспроизведению
- Ожидаемый результат
- Действительный результат
- Время начала теста
- Время окончания теста
- Трассировка стека, фрагмент лога
- Ключевые слова

Входными данными для сущности Пользователь являются:

- Логин, имя пользователя

- Имя
- Фамилия
- Пароль
- Адрес электронной почты

По окончанию анализа отчета об ошибке, система генерирует тикет, который будет в дальнейшем отправлен в Jira со следующими выходными данными:

- Ключ проекта на Jira
- Имя тикета
- Описание
- Трассировка стека

### 2.4 Схема базы данных и модель организации данных в системе

Для хранения данных в приложении используется реляционная модель данных. Данные хранятся в 10 таблицах, которые отвечают за системные сущности и связи между ними [10].

Для хранения сущности Пользователь (User) в базе предусмотрена таблица usr со следующими полями:

- User_id
- Account
- Email
- First_name
- Last_name
- Password

- Role_id

Для хранения сущности Роль (Role) в базе предусмотрена таблица role со следующими полями:

- Role_id
- Name

Для хранения сущности Проект (Project) в базе предусмотрена таблица project со следующими полями:

- Project_id
- Name
- Description
- Jira_project_key

Для хранения сущности Сценарий (Scenario) в базе предусмотрена таблица scenario со следующими полями:

- Scenario_id
- Name
- Description
- Project_id

Для хранения сущности Тесткейс (Testcase) в базе предусмотрена таблица testcase со следующими полями:

- Testcase_id
- Name
- Description
- Prerequisites
- Testcase_steps

- Expected_result
- Scenario_id

Для хранения сущности Отчёт об ошибке (Issue) в базе предусмотрена таблица issue со следующими полями:

- Issue_id
- Actual_result
- Expected_result
- Description
- Jira_issue_id
- Keywords
- Passed_to_jira
- Prerequisites
- Steps_to_reproduce
- Stacktrace
- Subject
- User_id
- Testcase_id

Для хранения сущности отвечающий за хранения состояния запущенного тестового сценария предусмотрена таблица testcase_process с полями:

- Testcase_process_id
- Start_date
- End_date
- User_id

Для хранения найденных исключений в логах предусмотрена таблица issue_exceptions со следующими полями:

- Issue_id
- Exception

После того как система осуществит поиск тикетов в Jira на основании исключений и ключевых слов, указанных в отчёте об ошибке, они будут сохранены в таблицах similar_issues_by_exceptions и similar_issues_by_keywords соответственно, для них определены следующие поля для таблицы similar_issue_by_keywords:

- Issue_id
- Similar_issue_by_keywords

И для таблицы similar_ issue_by_exceptions:

- Issue_id
- Similar_issue_by_exceptions

Связи между сущностями:

ManyToOne – между таблицей usr и role, между таблицами scenario и project, testcase и scenario, issue и user

OneToOne – между таблицами testcase_process и usr, testcase_process и testcase, issue и testcase

На рисунке 2.1 представлена ER – диаграмма базы данных.

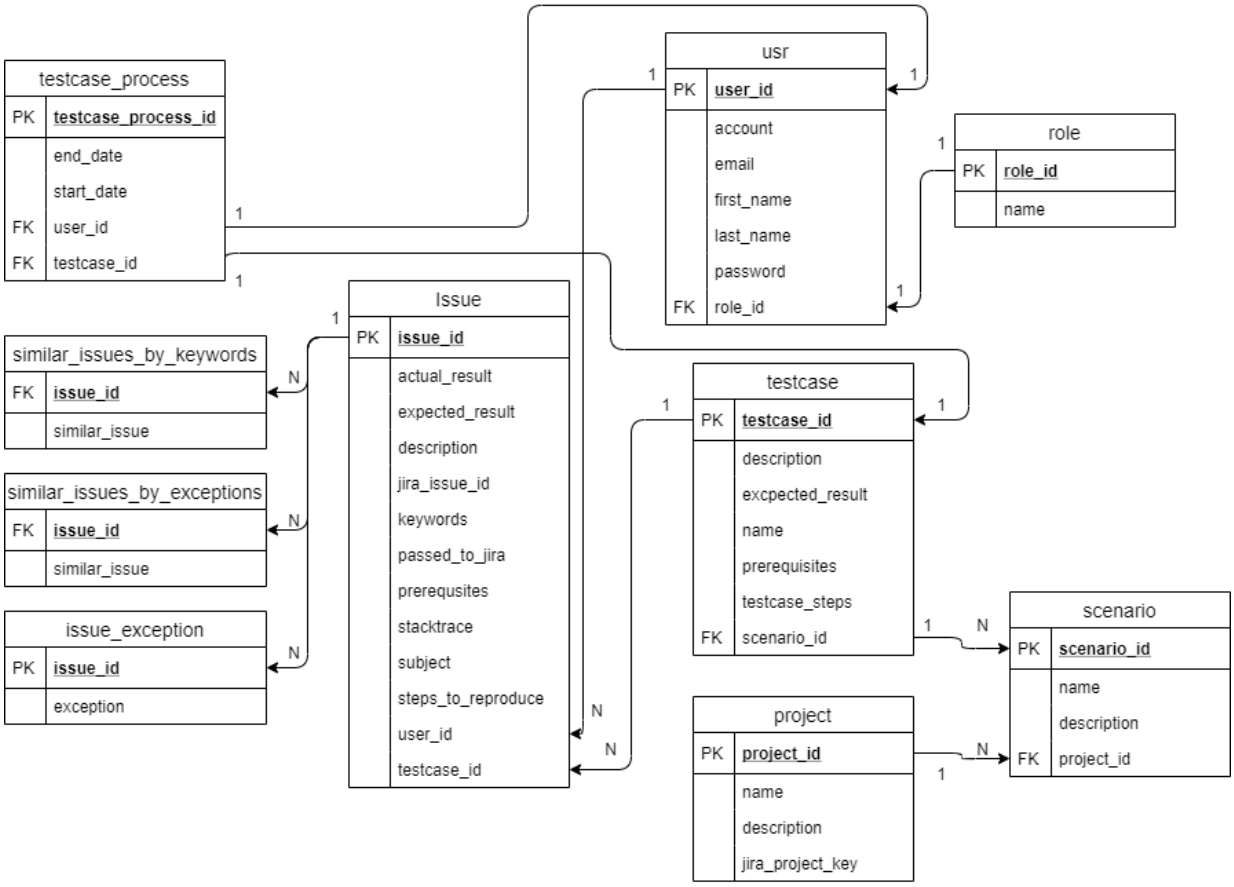


Рисунок 2.1 – ER диаграмма базы данных

## 2.5 Описание процесса ассесмента тикета в Jira

В процессе эксплуатации программного обеспечения пользователь взаимодействует с двумя основными системными сущностями – Тесткейс и Отчёт об ошибке.

Тесткейсы являются частью какого-либо сценария, т.е. по сути можно назвать сценарий контейнером для тесткейсов, который объединяет их по какому-либо признаку. Сценарий же в свою очередь имеет принадлежность к

какому-либо конкретному проекту. Таким образом связь выглядит так, как показано на рисунке 2.2.

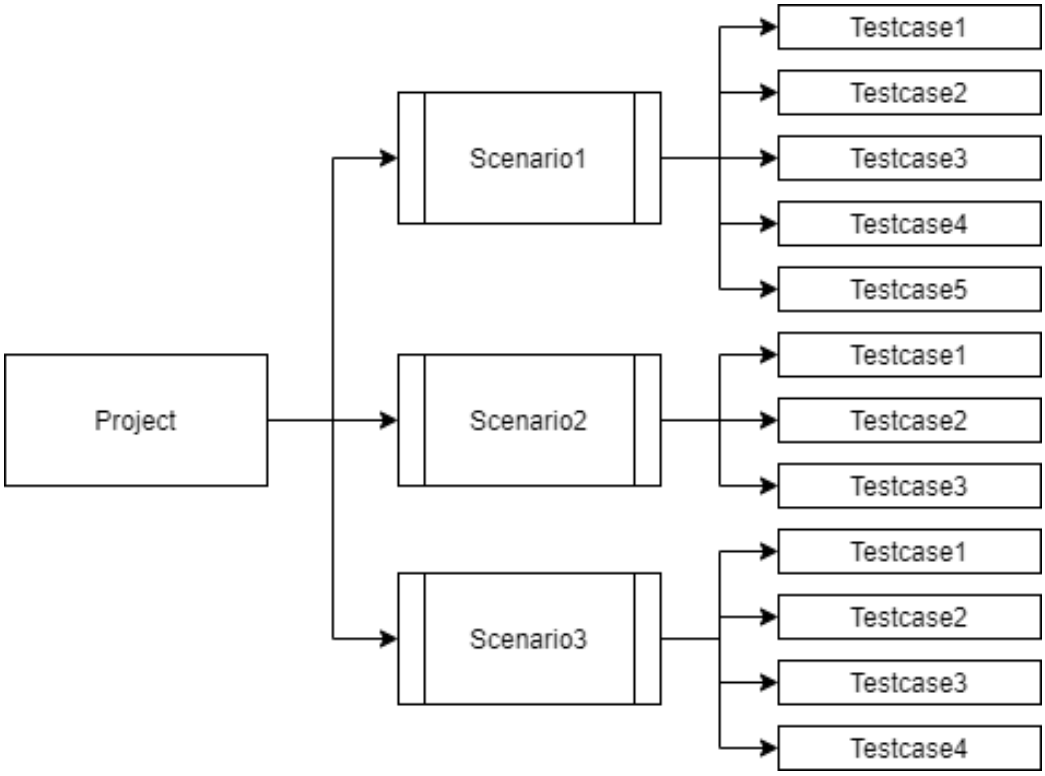


Рисунок 2.2 – Схема принадлежности сущностей

Каждый Тесткейс может быть запущен или остановлен. После запуска тесткейса система отслеживает его состояние через сущность TestcaseProcess, которая хранит в себе ссылку на тесткейс, пользователя, который его запустил, а также дату начала и дату окончания тестирования.

TestcaseProcess может быть остановлен двух случаях:

1. Успешное выполнение тесткейса (По кнопке Успешно пройден)
2. Неуспешное выполнение тесткейса (По кнопке Произошла ошибка)

В случае, когда тесткейс успешно завершён, система останавливает TestcaseProcess на этом заканчивается жизненный цикл этой сущности. Тесткейс меняет свой статус с Запущен на Не запущен. Такой тесткейс может быть запущен повторно при необходимости ещё раз провести тестирование.

В противоположном случае, по нажатию на кнопку Произошла ошибка, запускается процесс сбора информации для создания отчёта об ошибке. Пользователь перенаправляется на страницу создания отчёта. На странице создания отчёта будет набор обязательных для заполнения полей, часть из которых уже будет заполнена. Также у пользователя будет возможность добавить фрагмент лога, который будет получен из журналов тестируемого приложения за период между началом и стартом тестового сценария.

Таким образом, автоматически будут заполнены следующие поля отчёта об ошибке:

- Описание, с информацией об авторе отчёта, приветственным словом, указанием сценария и тесткейса, а также с временем начала и старта тестирования.
- Настройки, которые были сделаны перед началом тестирования, импортруются из тесткейса.
- Шаги для воспроизведения проблемы, также импортируются из тесткейса

- Ожидаемый результат тесткейса
- Время начала и время окончания

Все вышеописанные поля, кроме ожидаемого результата, вермени начала и времени окончания, пользователь может отредактировать на своё усмотрение.

Поля, которые необходимо заполнить пользователю вручную:

- Название отчёта
- Результат тестирования
- Фрагмент лога, стектрейс
- Ключевые слова

После нажатия кнопки создания отчёта, запускается сразу несколько процессов. В первую очередь происходит анализ предоставленного фрагмента логов и стектрейса на наличие в нём программных исключений (Exception), каждое найденное исключение записывается в отдельную таблицу в БД и связывается с текущим отчётом об ошибке.

Далее стартует процесс поиска тикетов на Jira, исходя из предоставленных для этого ключевых слов и найденных исключений. Для каждого такого поискового объекта выполняется запрос во внешнюю систему Jira посредством REST-запроса на API, которые предоставили разработчики этой багтрекинг системы.

В случае поиска похожих тикетов на Jira по ключевым словам, система совершает запросы на следующий REST API:

GET: /rest/api/3/issue/picker?query={query}&currentProjectId={projectId}

В запросе указываются несколько параметров, query и currentProjectId, значением параметра query является ключевое слово, по которому будет осуществляться поиск, а параметр currentProjectId указывает на идентификатор проекта на Jira, в котором будет произведен поиск. На этот запрос система получает ответ в формате json, из которого можно получить список найденных тикетов. Этот список прикрепляется к отчёту об ошибке и сохраняется в БД.

Для поиска тикетов на Jira по исключениям используется запрос, который включает выражение на JQL (Jira Query Language) необходимые поля из поисковой выдачи:

GET: /rest/api/3/search?jql=text~"{exception}"&fields=summary

В параметре jql передаётся JQL-выражение, в параметре fields – поле summary, которое в ответе на этот запрос содержит название и идентификатор тикета на Jira.

Запрос на JQL выглядит следующим образом: text ~ «exception», где на месте exception, будет отправлено найденное в логе исключение.

Ответом на этот запрос также служит список тикетов, который также добавляется в БД и свзявается с текущим отчётом об ошибке.

После завершения этих процессов отчёт об ошибке сохранятся в системе, а также к нему добавляется дополнительная информация в его описание. В ней указан результат поиска тикетов в Jira по исключениям и ключевым словам.

Далее пользователь имеет возможность отправить этот отчёт разработчикам в багтрекинг систему Jira, со всей необходимой информацией, которая была добавлена к отчету в процессе его создания и анализа. После отпраки отчёта образуется свзязь между отчётом между системами, поскольку теперь в отчёте об ошибке появляется ссылка на тикет в Jira, чтобы пользователь мог легко перейти на него и отслеживать его статус, а также прогресс работы над ним со стороны команды разработки.

# 3.ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 3.1 Обзор использованных технологий и инструментов разработки

Основным инструментом, который был использован при разработке программного обеспечения, является программная платформа Java, которая представляет собой среду для создания и исполнения кроссплатформенного прикладного программного обеспечения.

На сегодняшний день основными платформами для которых разрабатывается программное обеспечение с использованием Java являются мобильные приложения и клиент-серверные веб-приложения корпоративного сегмента. Популярность Java на рынке крупных корпоративных систем обусловлена тем, что разработка приложений ведётся сравнительно быстро, такие приложения являются довольно производительными и масштабируемыми, а также легко поддерживаемыми. Всё это достигается благодаря спецификации API, используемого для разработки таких приложений. Такой стандарт получил название Java Enterprise Edition (Java EE).

Java EE является промышленной технологией и в основном используется в высокопроизводительных проектах, в которых необходима надежность, масштабируемость, гибкость [18].

Спецификации максимально детализированы, чтобы обеспечить переносимость приложений между платформами. Основная цель спецификаций — обеспечение масштабируемости приложений и целостность данных во время работы системы.

Одним из наиболее популярных фреймворков для разработки корпоративных приложений на Java является Spring Framework [19], который используется в качестве контейнера внедрения зависимостей. Благодаря подходам Dependency Injection (DI) и Inversion of Control (IoC), которые лежат в его основе, создание и управление зависимостями между компонентами приложения становятся внешними задачами. А огромный выбор дополнительных компонентов фреймворка позволяют упростить и ускорить разработку, особенно в таких областях, как создание веб-приложений.

Поскольку для полноценной работы веб приложения на Spring Framework требуется достаточно громоздкая и сложная конфигурация зависимостей и компонентов, этот процесс становится весьма сложным и утомительным, а также подверженным различным ошибкам. Особенно, когда это крупное приложение с большим количеством сторонних библиотек и зависимостей. Так, например, необходимо вручную настраивать:

- В зависимости от типа создаваемого приложения (Spring MVC, Spring JDBC, Spring ORM и т.д.) импортировать необходимые Spring-модули
- Импортировать библиотеку web-контейнеров (в случае web-приложений)
- Импортировать необходимые сторонние библиотеки (например, Hibernate, Jackson), при этом вы должны искать версии, совместимые с указанной версией Spring
- Конфигурировать компоненты DAO [20], такие, как: источники данных, управление транзакциями и т.д.
- Конфигурировать компоненты web-слоя, такие, как: диспетчер ресурсов, view resolver
- Определить класс, который загрузит все необходимые конфигурации

Для того, чтобы ускорить и значительно упросить конфигурацию был использован Spring Boot.

Spring Boot — это проект, который является частью инфраструктуры Spring, преследующий цели по упрощению создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода. Это достигается за счёт упаковки всех необходимых сторонних зависимостей для каждого компонента Spring в так называемые starter-пакеты. По сути такой пакет представляет собой описание зависимостей,

которые необходимы Spring для работы того или иного модуля.

В приложении были использованы следующие starter-пакеты:

- spring-boot-starter-web
- spring-boot-starter-logging
- spring-boot-starter-thymeleaf
- spring-boot-starter-data-jpa
- spring-boot-starter-security

В качестве хранилища данных для приложения была выбрана база данных PostgreSQL [21], которая является свободной объектно-реляционной СУБД. Основными преимуществами следует отметить высокопроизводительные и надёжные механизмы транзакций и репликации, а также возможность работы с различными языками программирования и фреймворками, полную поддержку ACID и соответствие стандартам ANSI SQL-92, SQL-99, SQL:2003, SQL:2011.

Для осуществления доступа к базе данных PostgreSQL из приложения, был использован Spring Data JPA, который обеспечивает работу с данными, как с обычными Java объектами и коллекциями, совершая над ними привычные действия и операции, сохраняя, получая и удаляя их из базы данных. Такая простота и удобство также достигается

благодаря строгой спецификации Java Persistence API (JPA), которая описывает методы работы с объектами и данными.

Для создания веб-приложения был использован Spring Web MVC, который реализует паттерн Model — View — Controller, где:

Model (Модель) инкапсулирует (объединяет) данные приложения, для дальнейшей передачи и отображения их пользователю.

View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML страницу, отображаемую пользователю.

Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Для обеспечения безопасности приложения был использован Spring Security, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Spring Security позволяет за короткий срок, с минимальными усилиями и небольшим объёмом конфигурации внедрить в приложение необходимый функционал безопасности. Фреймворк Spring Security [22] представляет собой всеобъемлющее решение по обеспечению безопасности, реализующее возможность аутентификации и

авторизации как на уровне веб-запросов, так и на уровне вызовов методов. В приложении используется аутентификация, которая основана на использовании cookie и HTTP-сессии.

Для отображения данных на пользовательском интерфейсе используется шаблонизатор Thymeleaf. Он обладает широкими возможностями по интеграции со Spring и как следствие с Java, что позволяет не только использовать модели, которые были переданы контроллером для отображения, но и вызывать методы, а также проводить различные манипуляции с данными. А при помощи библиотеки thymeleaf-extras-springsecurity5 появляется дополнительная возможность по проверке соответствующих прав доступа у пользователя на какие-либо элементы на странице.

В качестве сервера приложений используется встроенный в Spring Boot контейнер сервлетов Apache Tomcat, который реализует спецификацию сервлетов. Tomcat работает в паре с HTTP сервером Apache HTTP Server и используется в качестве самостоятельного веб сервера для приложений написанных на Java и использующих спецификацию Servlet [23].

Интеграция с внешней системой JIRA осуществляется посредством REST API, а также библиотеки для Java - jira-rest-java-client-core. Библиотека предоставляет возможности по

работе с Jira из Java, путём предоставления соответствующей реализации необходимого функционала по работе с багтрекинг системой, например, операции над задачами, проектами, пользователями и т.д.

Для работы запросов из приложения, библиотеке jira-rest-java-client-core необходимо предоставить URL, по которому можно получить доступ к Jira, а также логин и API key пользователя с необходимыми разрешениями для работы с Jira. Для этого был использован пользователь с правами администратора на Jira.

Логин и API key используется для того, чтобы получить доступ к REST эндпоинтам Jira, поскольку они защищены при помощи Basic Authentication, которая предоставляет простой механизм для аутентификации при взаимодействии с REST API. Однако, поскольку базовая аутентификация постоянно отправляет имя пользователя и пароль (в кодировке base64) при каждом запросе, который может кэшироваться в веб-браузере, это не самый безопасный метод аутентификации, который существует на данный момент, но его вполне достаточно для настройки интеграции. Таким образом, для того чтобы получить доступ к защищенному ресурсу, каждый HTTP запрос должен содержать заголовок вида Authorization : Basic base64(login:apiKey).

Такой механизм необходимо использовать как при использовании REST API напрямую, так и при использовании

библиотеки, поскольку она лишь удобная обёртка для запросов к REST API, которая позволяет работать с Jira используя привычные конструкции языка Java, такие как: объекты, методы и т.д. В случае же использования REST API, где всё взаимодействия осуществляется через передачу данных в формате json, необходимо самостоятельно проводить разбиение строкового ответа от Jira на компоненты, создавать из них объекты и следить за целостностью данных. Такой подход является более сложным и требующим написания большего количества кода, но зато предоставляет большую гибкость и возможность использовать API так, как этого требует решаемая задача. Таким образом, используя GET, POST, DELETE запросы на соответствующие REST эндпоинты появляется возможность интеграции с функционалом Jira из любого приложения, разработанного практически на любом языке программирования, используя сетевое взаимодействие. Благодаря этому запросы получаются довольно гибкими и лаконичными, а также позволяют получать и отправлять необходимые поля и объекты, взаимодействуя с ними из Java. Например, чтобы удалить какую-либо задачу на Jira, необходимо отправить DELETE запрос на url – https:// <jiraAddress> /rest/api/3/issue/, указать необходимый авторизационный заголовок – Authorization : Basic base64(login:apiKey), а в качестве параметра – ключ задачи на Jira. Таким образом в общем виде, используя утилиту curl,

запрос будет выглядеть следующим образом: curl -XGET -H 'Authorization: login:password' 'https://jira4cloud.atlassian.net/rest/api/3/issue/XX-11', в результате будет возвращён HTTP код ответа, который сообщит о том, успешно ли была совершена операция или же произошла какая-либо ошибка.

### 3.2 Описание реализованных программных модулей

Поскольку программное обеспечение разработано с использованием паттерна [11] Model-View-Controller (MVC) [12], организация проекта и структура его модулей подчиняется правилам организации пакетов, классов, ресурсов и конфигурационных файлов, использующих для работы фреймворк Spring Web MVC.

Основными конфигурационными файлами приложения являются application.properties и log4j2-spring.xml

В файле log4j2-spring.xml описаны настройки логгирования приложения, конфигурация формата вывода логов, директория и файл, в который будет вестись их запись, а также уровень логгирования и настройки ротации файлов с логами.

Файл application.properties представляет собой конфигурационный файл с настройками для Spring приложения. В нем содержатся настройки для подключения к базе данных:

- spring.datasource.url=jdbc:postgresql://localhost/issue-analysis-db
- spring.datasource.username=postgres
- spring.datasource.password=root

Настройками Java Persistence API и Hibernate [13]:

- spring.jpa.generate-ddl=true
- spring.jpa.show-sql=false
- spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
- spring.jpa.hibernate.ddl-auto=validate

### 3.3 Описание архитектуры приложения

Пакет models содержит классы сущностей системы и которые также используются для генерации таблиц в БД посредством Spring Data. Для этого каждый класс должен быть аннотирован специальной аннотацией, указывающей на то, что по данным из этого класса будут созданы таблицы в БД — @Entity и @Table. Для указания поля, которое будет использовать в качестве первичного ключа используется аннотация @Id и @GeneratedValue для указания на то, что значение этого поля должно генерироваться автоматически или по какой-либо выбранной стратегии. Для определения связей между сущностями используются аннотации @OneToOne, @OneToMany, @ManyToOne, @ManyToMany и

@ElementCollection. На рисунке 3.1 приведён пример использования Spring Data для генерации в базе данных таблицы для сущности Пользователь (User) с использованием вышеописанных аннотаций.

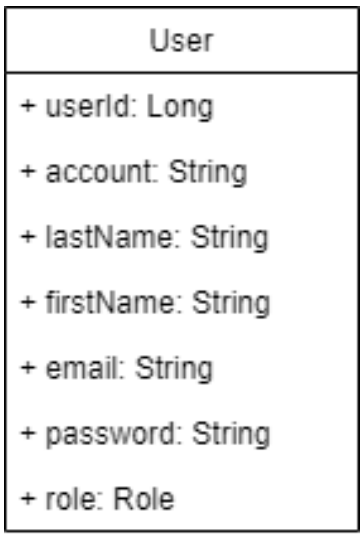


Рисунок 3.1 – Класс модели User.

Для отображения данных на пользовательском интерфейсе использован шаблонизатор Thymeleaf [14], который расширят возможности HTML по обработке тэгов, используя специальные атрибуты, позволяющие передавать какие-либо значения с сервера на страницу пользователя. В архитектуре MVC он выступает на стороне отображения (View).

Существенно позволило увеличить скорость разработки пользовательского интерфейса использование так называемых макетов (layouts), которые позволяют переиспользовать фрагменты HTML страниц по всему веб-приложению. Так, например, в проекте использован макет,

который содержит в себе навигационную панель и основной блок с контентом. Это позволило значительно уменьшить объем кода и создать наиболее универсальный пользовательский интерфейс, который поддерживает отображение контента в зависимости от разрешений пользователя на его просмотр.

Все используемые в проекте элементы пользовательского интерфейса, согласно правилам построения Spring Web MVC [15] приложений, располагаются в пакете resources. В нём, по отдельным пакетам, согласно функциональной принадлежности расположены HTML файлы, на который происходит предварительный рендер и отображение данных с сервера.

### 3.4 Контроллеры приложения

Пожалуй, основным элементом в архитектуре MVC является контроллер, который обеспечивает маршрутизацию пользовательских запросов, связывает между собой Вид и Модель, а также определяет поведение приложения, реагируя на различные действия пользователя.

Список контроллеров приложения:

- ApplicationController – основной контролер приложения, отвечает за запросы к главной странице, а также к странице входа и регистрации

- IssueController – контроллер для работы с отчетами об ошибках
- LogController – контроллер для получения логов с сервера
- ProjectController – контроллер для работы с проектами
- ScenarioController – контроллер для работы со сценариями
- TestcaseController – контроллер для работы с тесткейсами
- UserAdminController – контроллер, отвечающий за функции администратора
- UserProfileController – контроллер, отвечающий за функции редактирования профиля пользователя

Основной задачей контроллеров в приложении является обработка запроса от пользователя на получение и отправку каких-либо данных, проверки прав на осуществление операции, вызов необходимой бизнес-логики из сервисного слоя приложения, получение из него каких-либо данных, объектов, моделей и их передача на отображение пользователю. В проекте все контроллеры расположены в пакете controllers и отвечают за работу с той или иной сущностью. Так, например, различные методы, отвечающие за работу с отчетами об ошибках (Issue) располагаются в контроллере IssueController и доступны по URL, начинающемуся на /issues/issue.

Рассмотрим некоторые методы и их URL, которые отвечают за работу с отчётами об ошибках, в формате HTTP-метод [16] – URL:

GET : /issues – Получение страницы со всеми отчетами об ошибках.

GET : /issues/issue/{id} – Получение страницы с отчётом об ошибке по его идентификатору.

POST : /issue/newJiraIssueInternal – Создание отчёта об ошибке, сохранение его в системе.

GET : /issue/sendToJira/{id} – Отправка отчета об ошибке в Jira по его идентификатору.

Запрос на получение логов сервера доступен через REST API и представляет собой REST эндпоинт с адресом /public/stacktrace, передав в который в качестве параметра запроса перемененную startDate, можно получить фрагмент лога, за указанный период времени, таким образом, запрос может выглядеть следующим образом:

GET : /public/stacktrace?startDate=2020-05-30T14:55:38

На рисунке 3.2 приведён пример кода класса LogController, отвечающий за логику обработки запросов на получение логов.

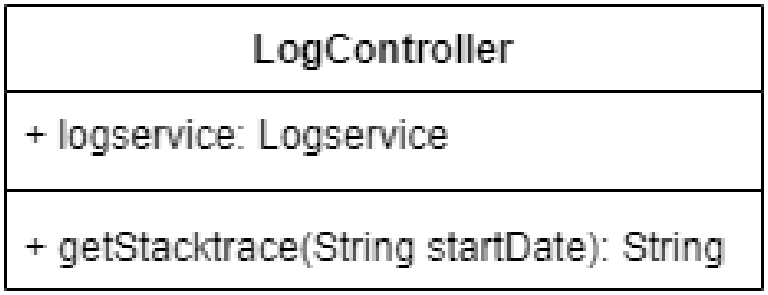


Рисунок 3.2 – Класс LogController

### 3.5 Слой бизнес-логики приложения

После обработки запроса соответствующим контроллером, управление переходит в слой бизнес-логики (сервисный слой). В нём осуществляется валидация данных, манипуляция над ними, получение данных из БД, а также другая безнес-логика приложения.

Краткое описание классов слоя бизнес-логики приложения и их предназначение:

- UserService – CRUD операции над пользователями
- RoleService – Операции поиска и связывания ролей пользователя
- ProjectService – CRUD операции над Проектами
- ScenarioService – CRUD операции над Сценариями
- TestcaseService – CRUD операции над Тесткейсами
- TestcaseProcessService – управление запуском, остановкой, получением и работой с запущенными Тесткейсами.

- IssueService – Операции создания Отчётов об ошибках, их поиска, получения, отправки в Jira, обновления описания после поиска похожих тикетов.
- FindJiraIssueService – Отвечает за поиск похожих тикетов на Jira по исключениям из логов, а также по ключевым словам. Выполняет запросы на API Jira, обрабатывает ответы, сериализует и десериализует объекты запросов и ответов. Предоставляет интерфейс, по которому можно получить списки найденных тикетов.

### 3.6 Слой доступа к данным

Для доступа к данным в БД из сервисного слоя используется механизм JpaRepository, который предоставляется вместе со Spring Data [17], который является ещё одним компонентом в инфраструктуре Spring, который предоставляет удобный интерфейс для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение их полей, а также поиска и удаления. Для этого необходимо лишь расширить интерфейс JpaRepository<T, ID> и объявить методы, которые будут отвечать за взаимодействие с той или иной сущностью, указанной при расширении этого интерфейса.

Список репозиториев приложения:

- IssueJpaRepository

- ProjectJpaRepository
- RoleJpaRepository
- ScenarioJpaRepository
- TestcaseJpaRepository
- TestcaseProcessJpaRepository
- UserJpaRepository

Пример использования JpaRepository на примере TestcaseProcessJpaRepository, для получения объекта TestcaseProcess для конкретного пользователя и тесткейса изображен на рисунке 3.3.

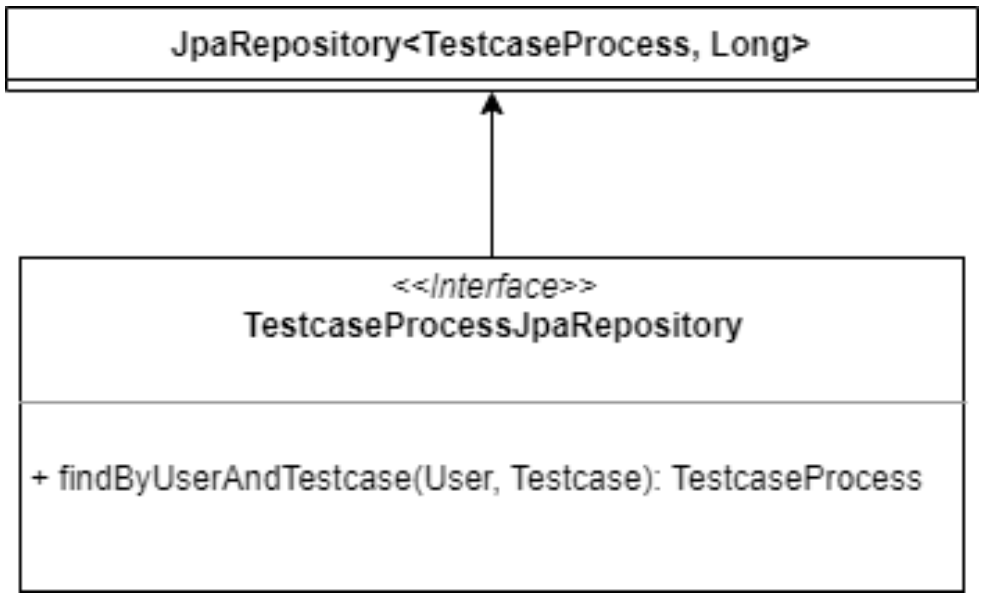


Рисунок 3.3 – Интерфейс TestcaseProcessJpaRepository

Таким образом, использование Spring Data Jpa и механизма JpaRepository значительно упрощает доступ к данными в БД и операции над ними из приложения, а также предоставляет удобный механизм по расширению, если

разработчику не хватает функционала, который доступен по умолчанию.

### 3.7 Описание работы программного обеспечения.

Использование ПО подразумевается только аутентифицированными пользователям, поэтому при открытии любой страницы происходит на страницу с формой логина с предложением ввести свои учетные данные или создать аккаунт, если пользователь ранее не пользовался системой и не имеет в ней своего профиля. На рисунке 3.4 изображена форма входа, в которой необходимо ввести имя пользователя и пароль. После успешного входа пользователь будет перенаправлен на главную страницу (рисунок 3.6), откуда пользователю доступно меню навигации по системе.

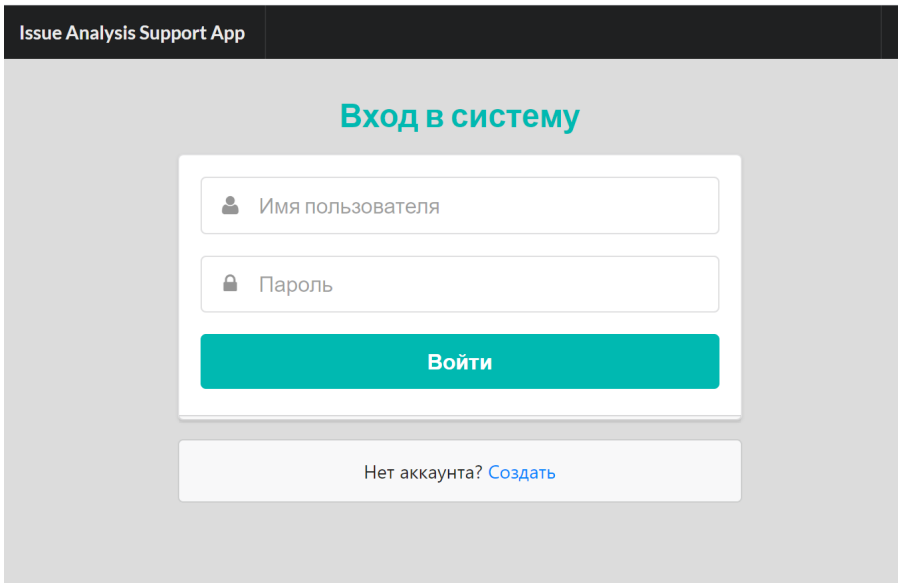


Рисунок 3.4 – Страница входа в систему

Страница регистрации (рисунок 3.5) представляет собой форму, где пользователю необходимо ввести свой логин, имя и фамилию, а также придумать пароль и ввести свой адрес электронной почты. После ввода учетных данных произойдет создание аккаунта, и пользователь автоматически войдет в систему, после этого будет совершено перенаправление на главную страницу.

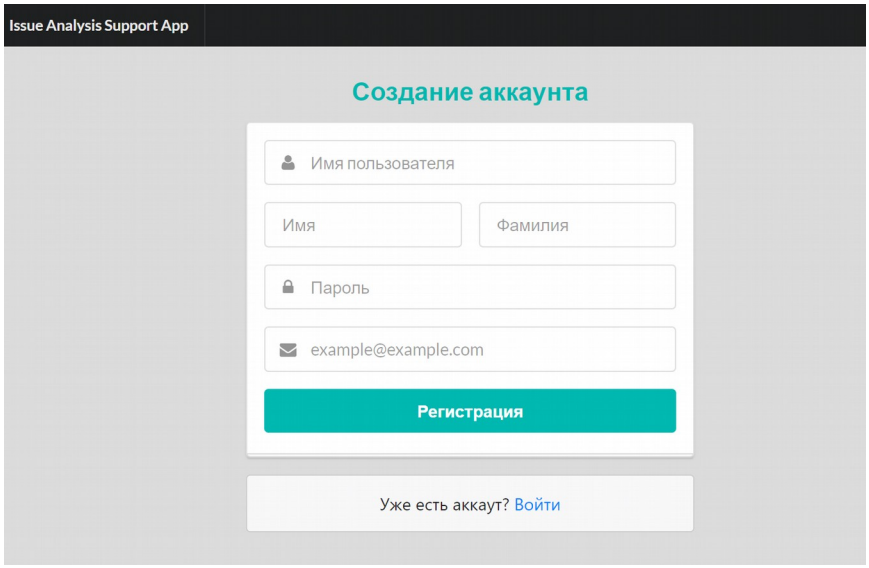


Рисунок 3.5 – Страница создания аккаунта в системе

На главной странице приложения (рисунок 3.7) располагается основная информация о системе, её предназначении и действиях, которые можно выполнять с помощью неё. На рисунке 3.6 изображена страница профиля.

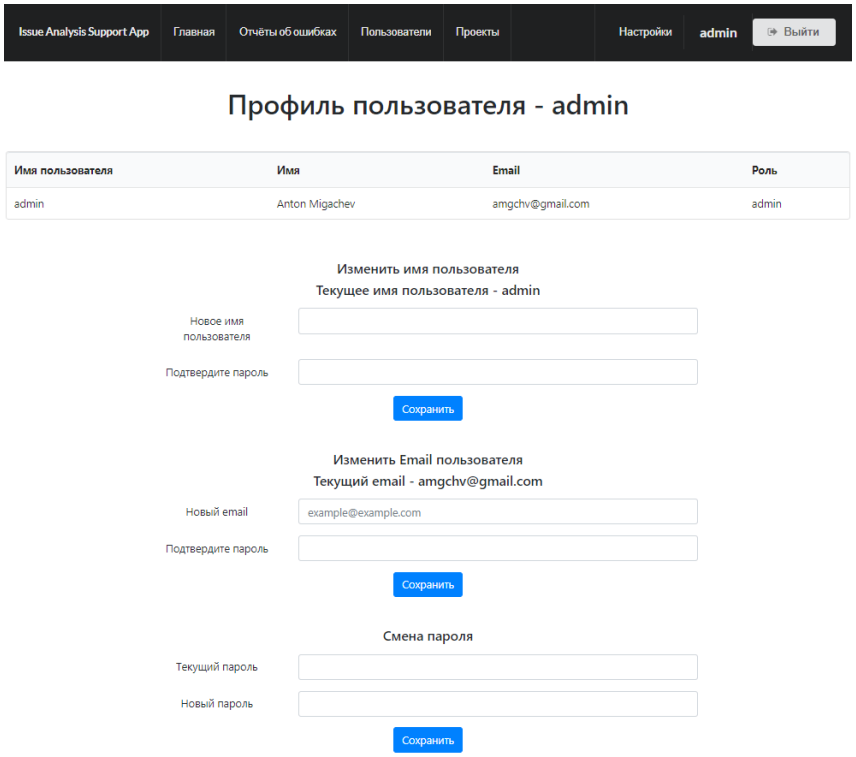


Рисунок 3.6 – Страница редактирования профиля

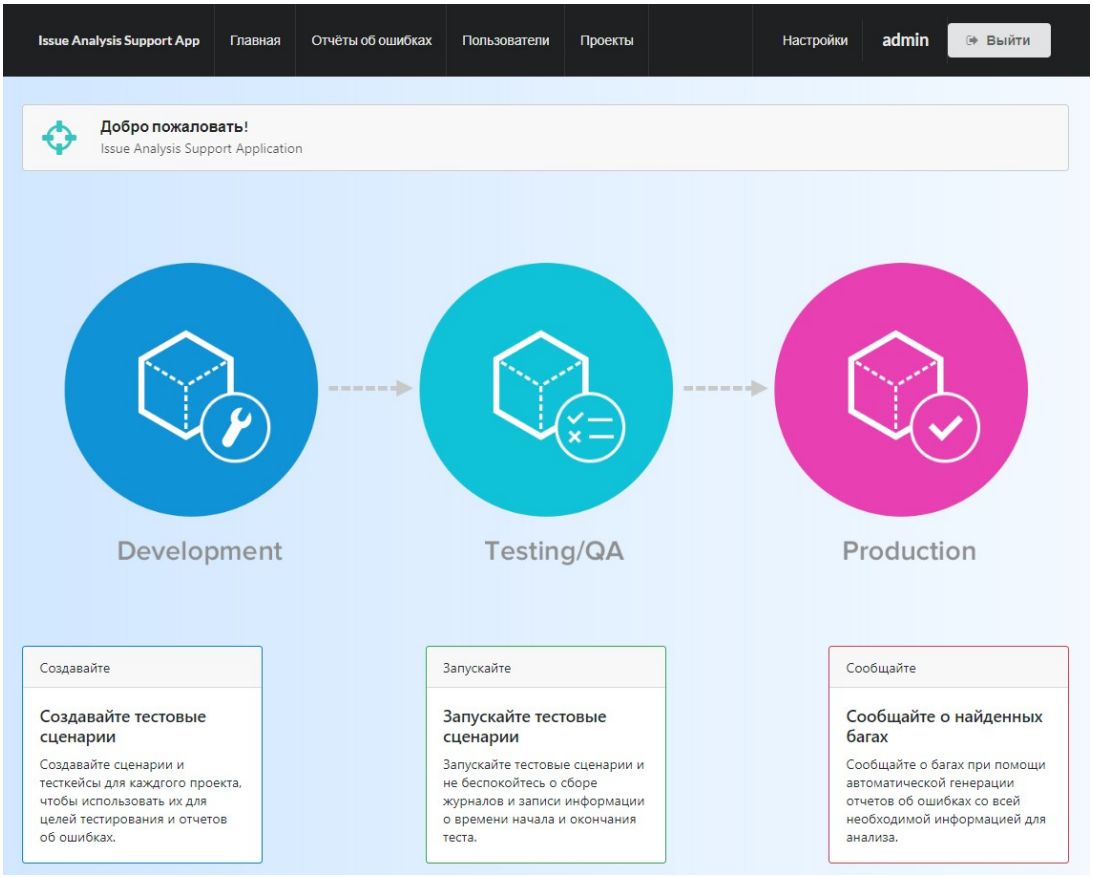


Рисунок 3.7 – Главная страница

На странице со всеми пользователями у администратора есть возможность сменить роль у выбранного пользователя, а также удалить пользователя из системы, как это показано на рисунке 3.8.

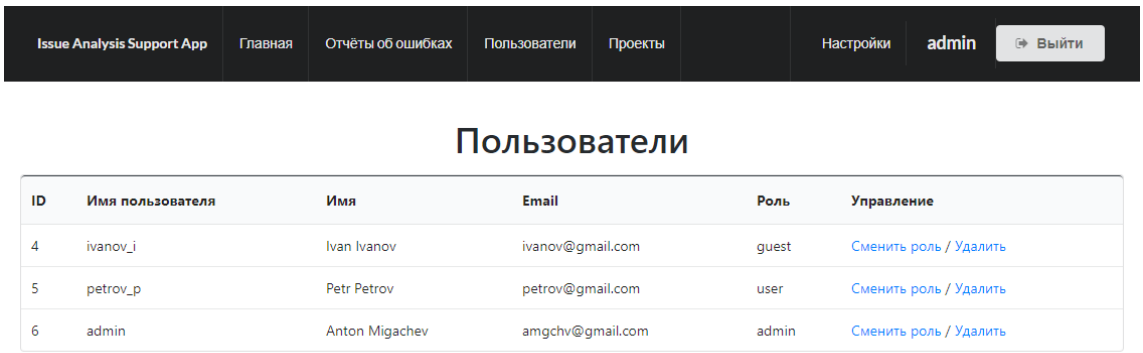

Рисунок 3.8 – Страница со списком пользователей системы

Страница изменения роли в системе у пользователя (рисунок 3.9). На ней отображается имя пользователя, у которого будет изменена роль, а также список ролей из которых нужно выбрать ту, которую необходимо присвоить пользователю.

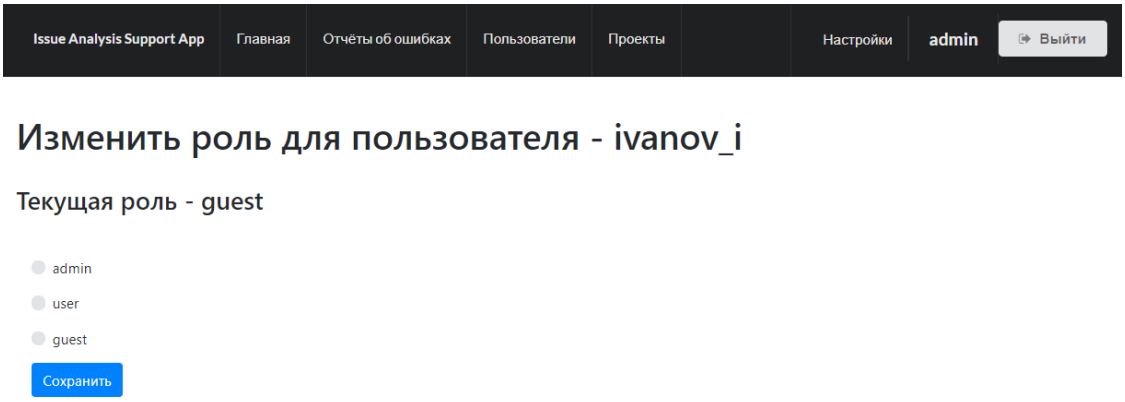


Рисунок 3.9 – Страница изменения роли

Началом работы с системой можно считать запуск синхронизации проектов из Jira. В результате этого действия выполняется запрос на получение проектов, которые уже созданы в Jira и их импорт в систему. Изначально страница с проектами пуста (рисунок 3.10), но после выполнения этой операции на ней будут отображаться созданные в результате синхронизации проекты.

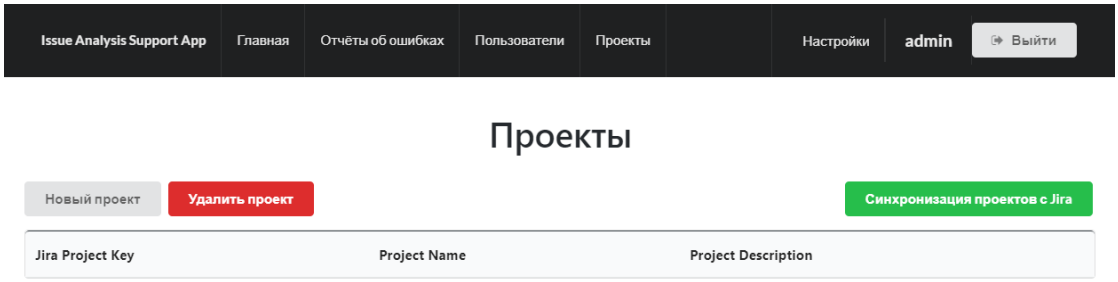


Рисунок 3.10 – Страница со списком проектов

Страница с проектами в багтрекинг системе Jira (рисунок 3.11) содержит в себе список проектов, с некоторой дополнительной информацией о них, такой как тип проекта, его ключ, а также название и руководителя.

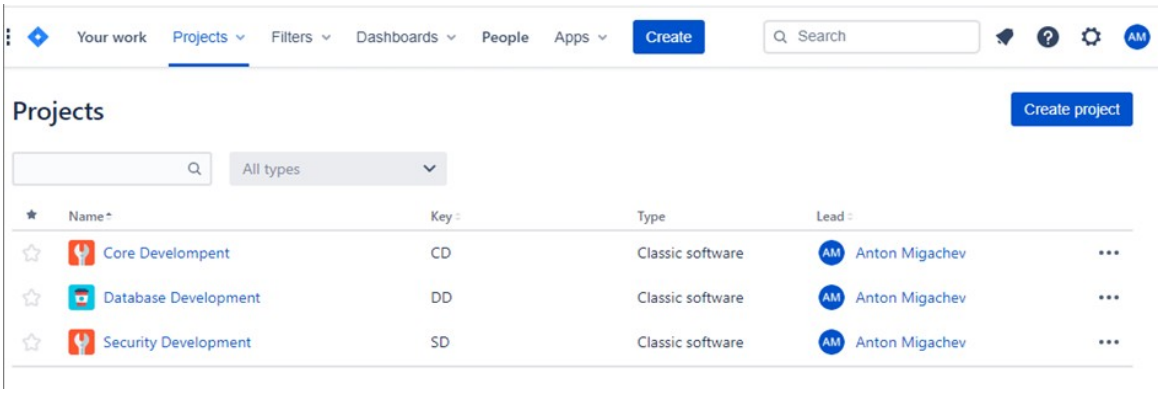


Рисунок 3.11 – Страница со списком проектов на Jira

После нажатия на кнопку Synchronize Projects with Jira на произойдёт синхронизация проектов между системами и после этого будут созданы проекты, если их ещё нет в системе. Добавленные в результате синхронизации проекты изображены на рисунке 3.12.

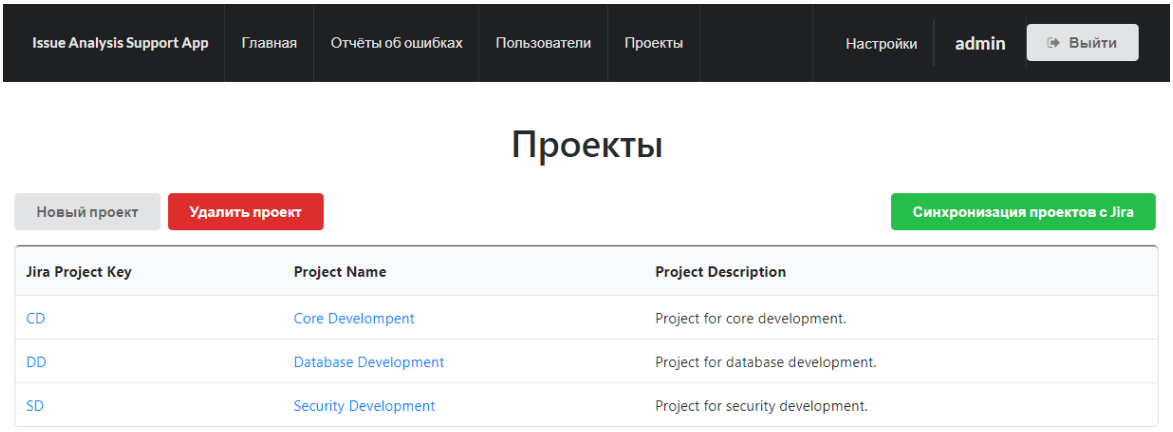


Рисунок 3.12 – Страница со списком проектов после синхронизации

Также имеется возможность добавления проекта в систему без выполнения синхронизации, для этого нужно ввести его название, описание и ключ проекта на Jira, как это показано на рисунке 3.13.

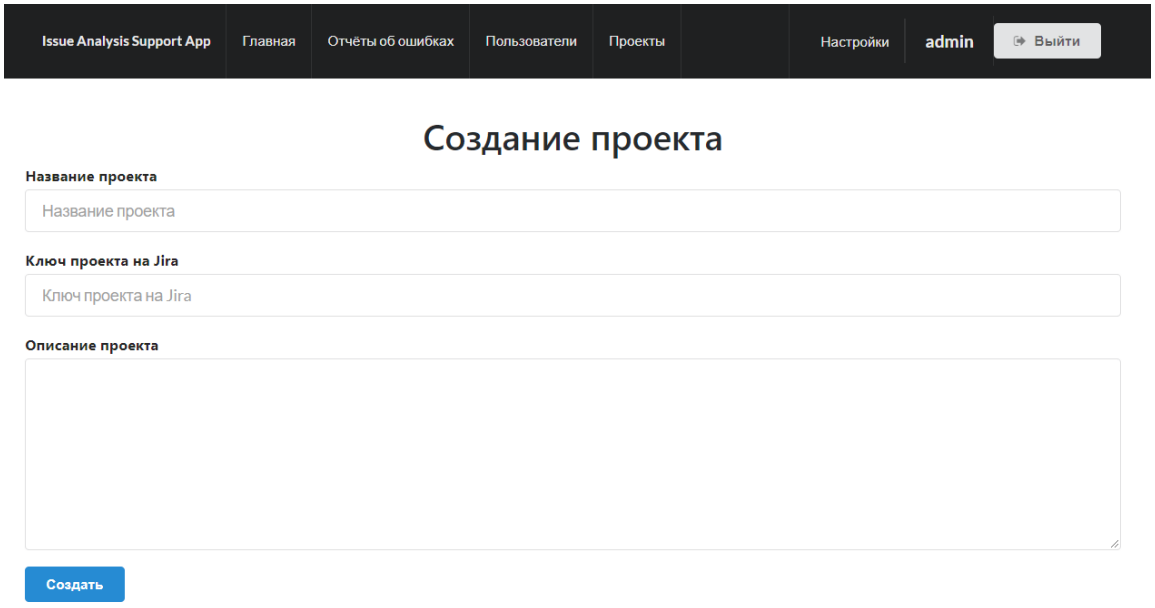


Рисунок 3.13 – Страница создания проекта

Для удаления проекта необходимо нажать на соответствующую кнопку, тогда произойдёт переход на страницу удаления, где нужно будет вести название проекта, который будет удалён и подтвердить действие нажатием на кнопку Удалить (рисунок 3.14).

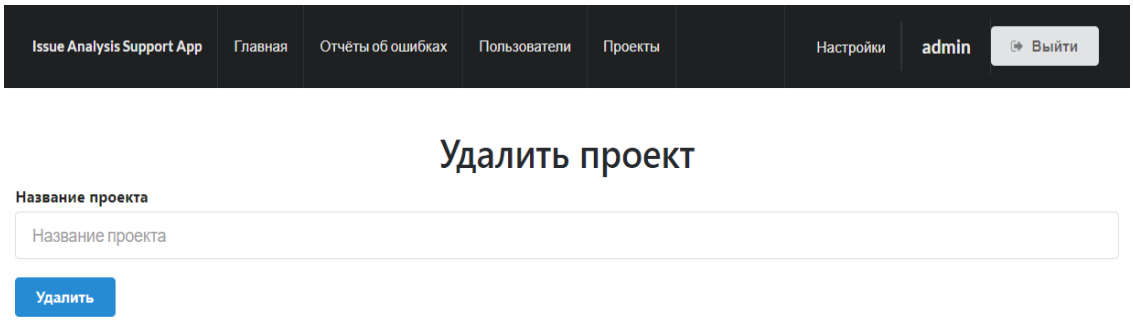


Рисунок 3.14 – Страница удаления проекта

На странице проекта расположен список сценариев для этого проекта, а также кнопка, которая вызывает интерфейс создания нового сценария. По нажатию на название сценария, открывается выпадающий список со всеми тесткейсами, которые относятся к этому сценарию. Также можно открыть каждый сценарий на отдельной странице или удалить его. Страница просмотра проекта, сценариев и вложенных в него тесткейсов изображена на рисунке 3.15.

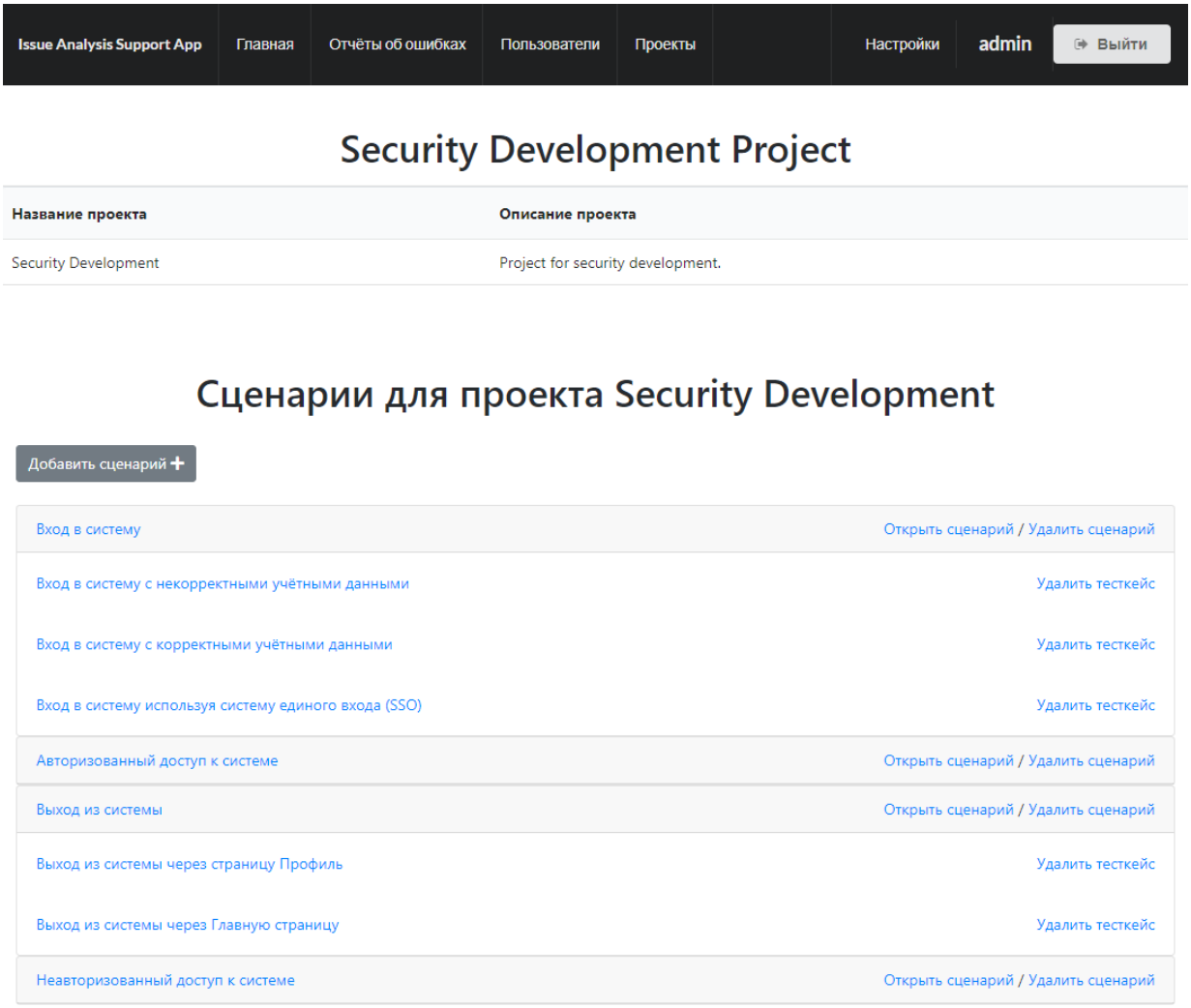


Рисунок 3.15 – Страница проекта с развёрнутым списком сценариев и вложенных в них тесткейсов

На странице просмотра сценария пользователю доступен список тесткейсов для этого сценария, а также кнопка, которая вызывает интерфейс создания нового тесткейса (рисунок 3.16). В выпадающей форме пользователю необходимо ввести название тесткейса, описание тесткейса, пререквизиты, шаги тестового сценария, а также ожидаемый результат после успешного прохождения тесткейса, после этого нажать кнопку Добавить.

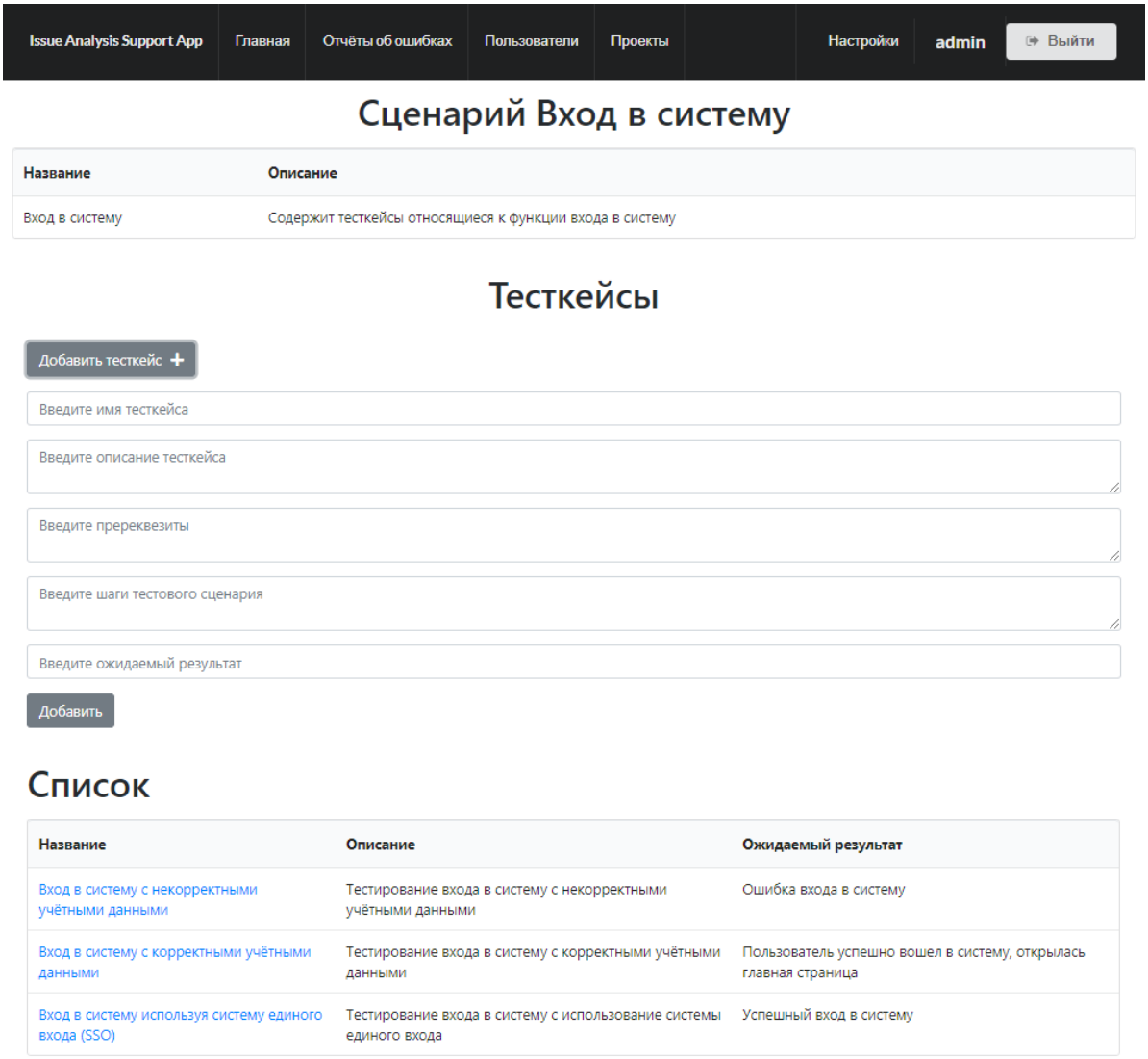


Рисунок 3.16 – Страница сценария с открытым окном создания тесткейса

По нажатию на название тесткейса открывается страница его просмотра (рисунок 3.17). На ней можно увидеть его статус (Запущен или Не запущен), а также кнопку которая запустит процесс тестирования, текущего тесткейса. В результате этого тесткейс перейдёт в статус Запущен, а на пользовательском интерфейсе будут доступны

кнопки завершения тестового сценария, которые подразумевают его успешность или провал.

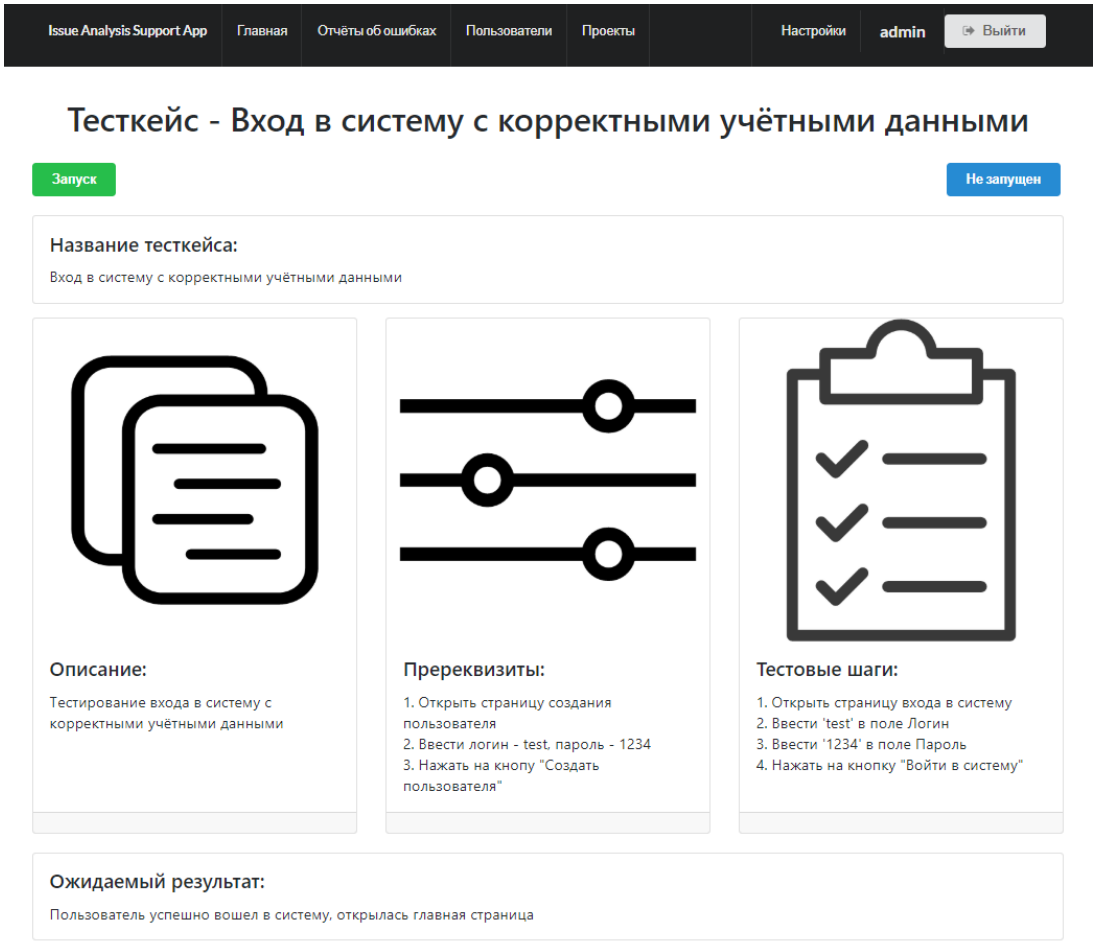


Рисунок 3.17 – Страница просмотра тесткейса

После прохождения всех необходимых предварительных настроек и выполнения пунктов из тесткейса в случае, когда он прошёл успешно пользователь сообщает об этом системе путём нажатия на кнопку Успешно пройден (рисунок 3.18). В случае, когда что-то пошло не так, например, возникла какая-либо ошибка или действительный результат отличается от ожидаемого, пользователю необходимо нажать кнопку Произошла ошибка, тогда пользователь будет

перенаправлен на страницу создания отчета об ошибке (рисунок 3.19).

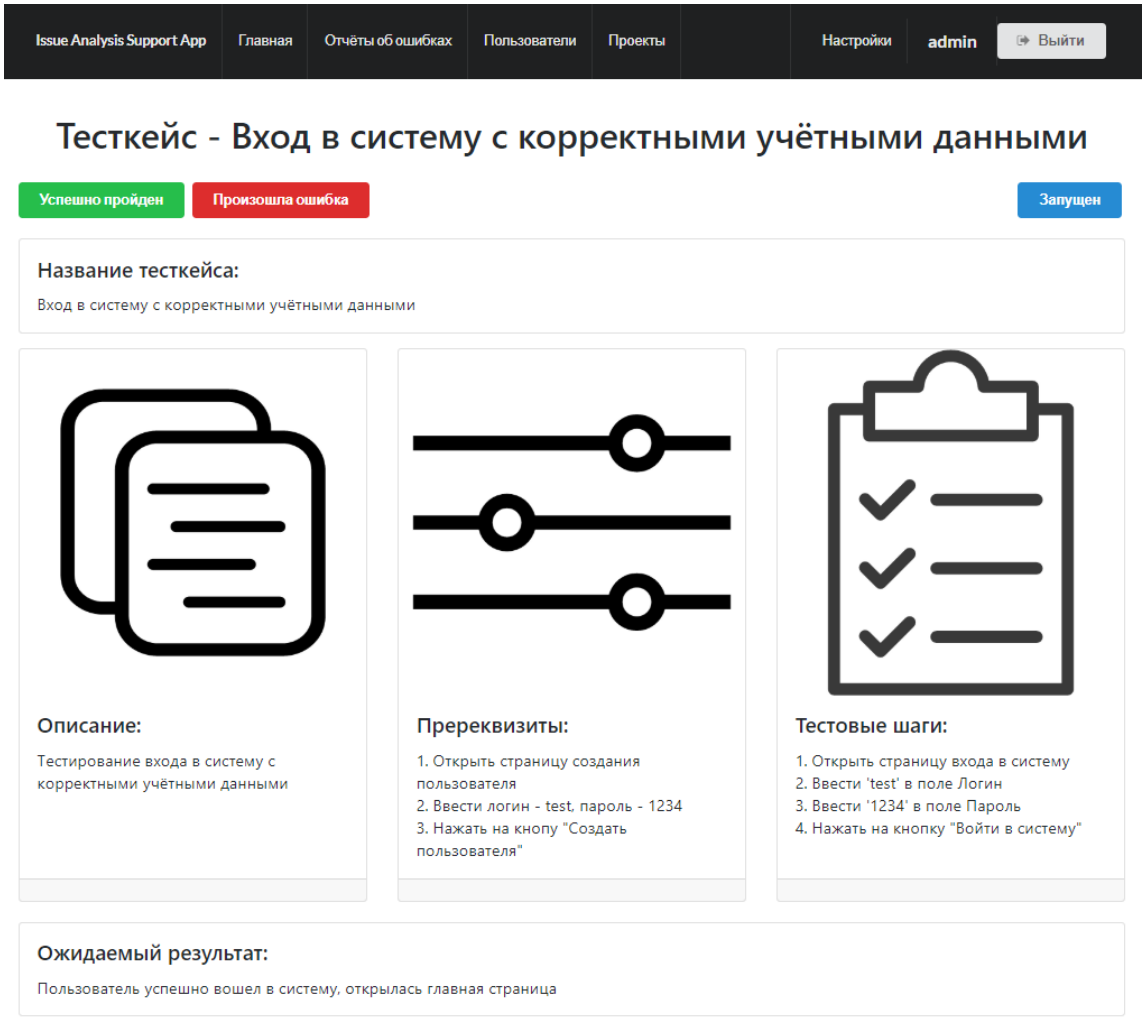


Рисунок 3.18 – Страница просмотра запущенного тесткейса

Страница создания отчета представляет собой форму, в которой пользователь в дополнение к уже имеющимся автоматически заполненным полям должен указать название отчёта, а также результат, который получился в результате тестирования, трассировку стека (часть лога) за время, в которое проводилось тестирование и ключевые слова,

которые кратко описывают суть проблемы и будут использованы для поиска похожих тикетов. Также при необходимости имеется возможность отредактировать поля с описанием проблемы, а также шагов, которые были совершены до начала тестирования, а также шаги, которые проделаны в процессе тестирования.

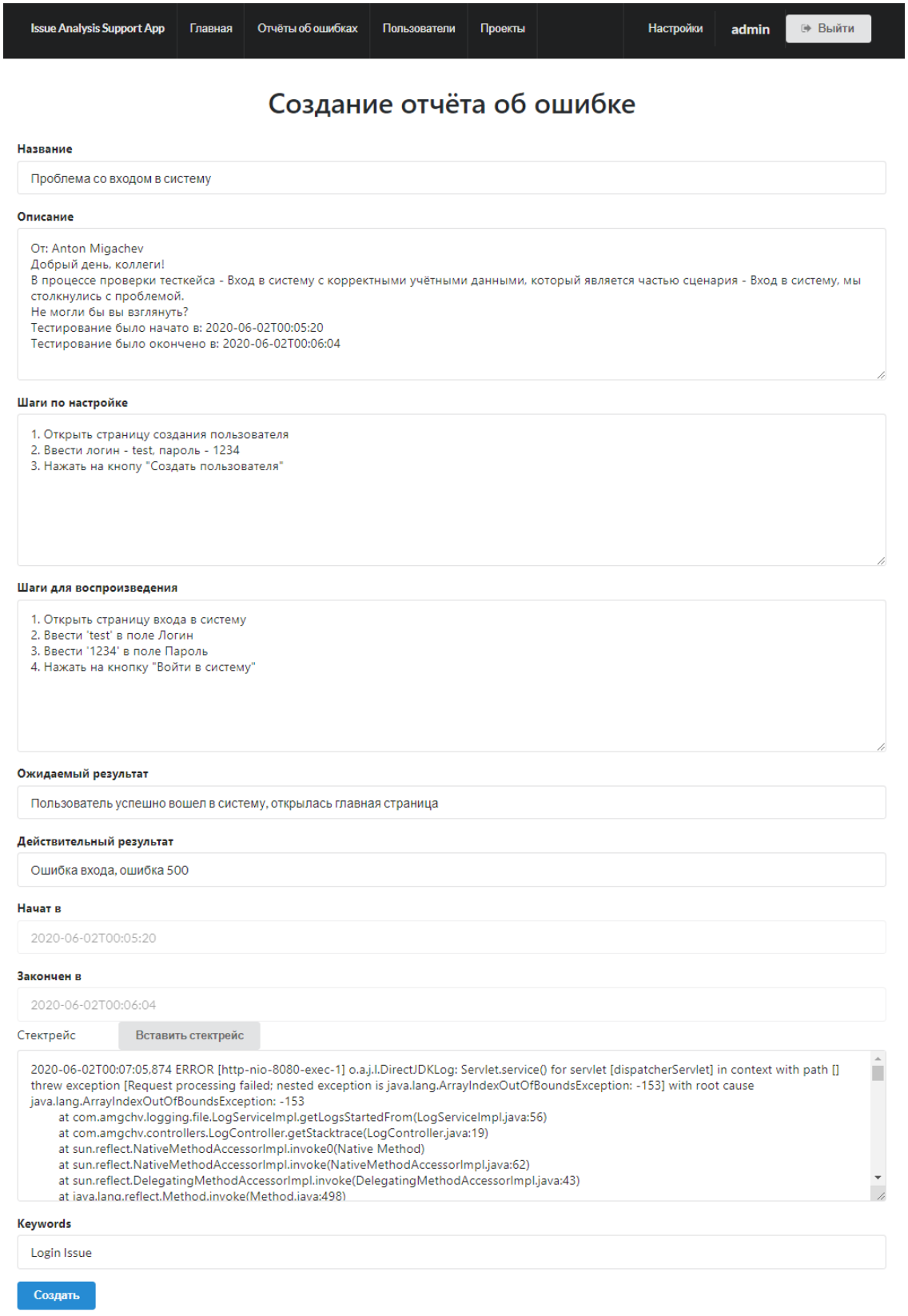


Рисунок 3.19 – Страница создания отчёта об ошибке

После нажатия на кнопку создания отчёта, пользователь перенаправляется на страницу со списком всех отчетов (рисунок 3.20). На ней отображается название отчета, автор, был ли отправлен этот отчёт в Jira, а также суммарное количество схожих тикетов, которые были найдены в Jira.

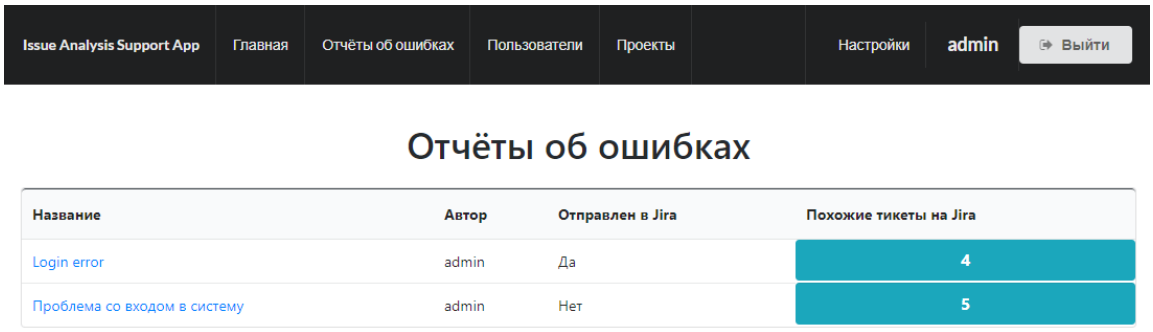


Рисунок 3.20 – Страница просмотра отчётов

На странице просмотра отчета (рисунок 3.21) пользователю доступно полное описание проблемы, шагов, которые были проделаны до начала тестирования и непосредственно во время тестирования, время старта и окончания тестирования, ожидаемы и действительный результат, а также результаты поиска похожих тикетов, которые подгружаются из Jira. После нажатия на кнопку Отправить отчет в Jira, отчёт будет отправлен в Jira, предварительно преобразовавшись в подходящий для этого формат.

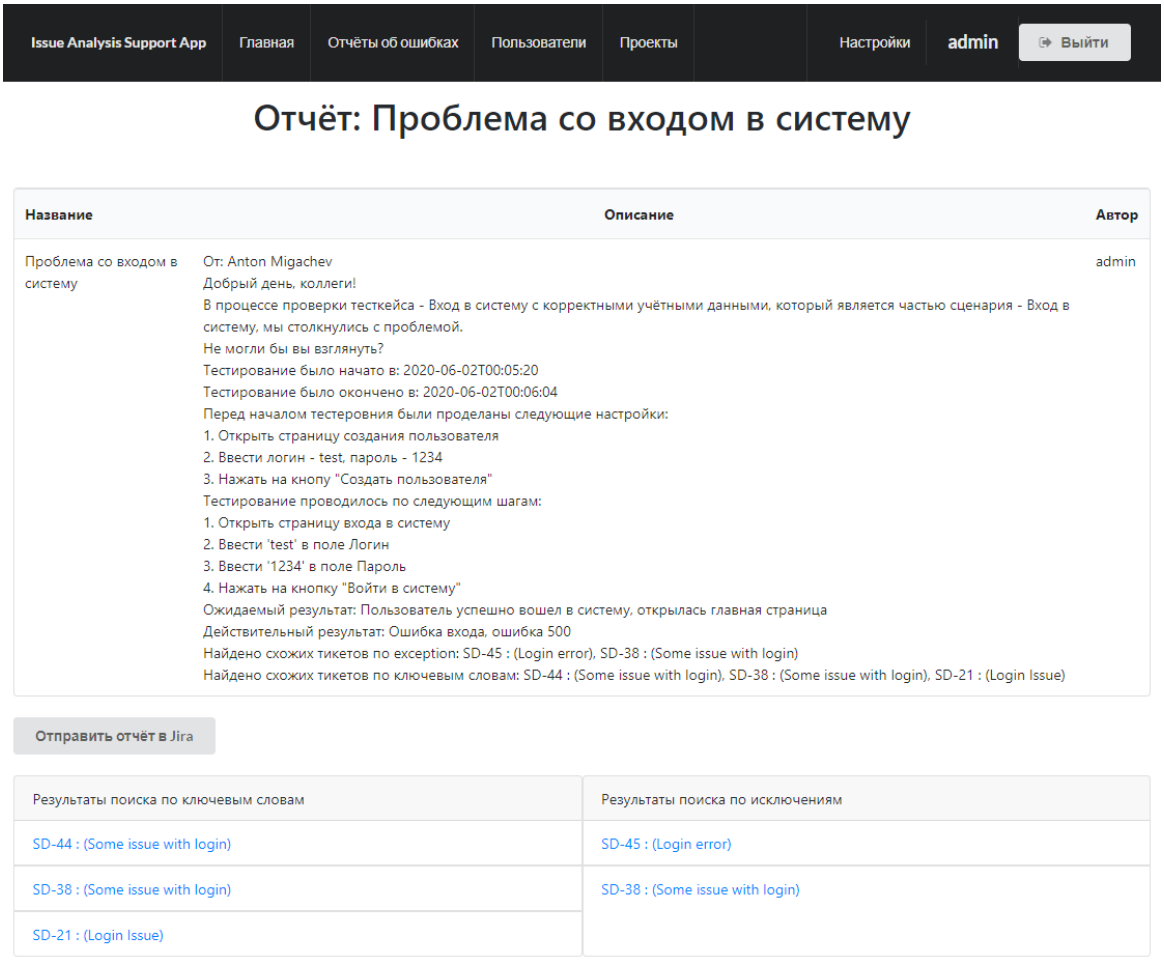


Рисунок 3.21 – Страница просмотра отчёта

После отправки отчёта в Jira происходит перенаправление на созданный тикет (рисунок 3.22), который будет содержать прежде заполненные поля, а также информация о похожих тикетах, которые были найдены на предыдущем этапе.

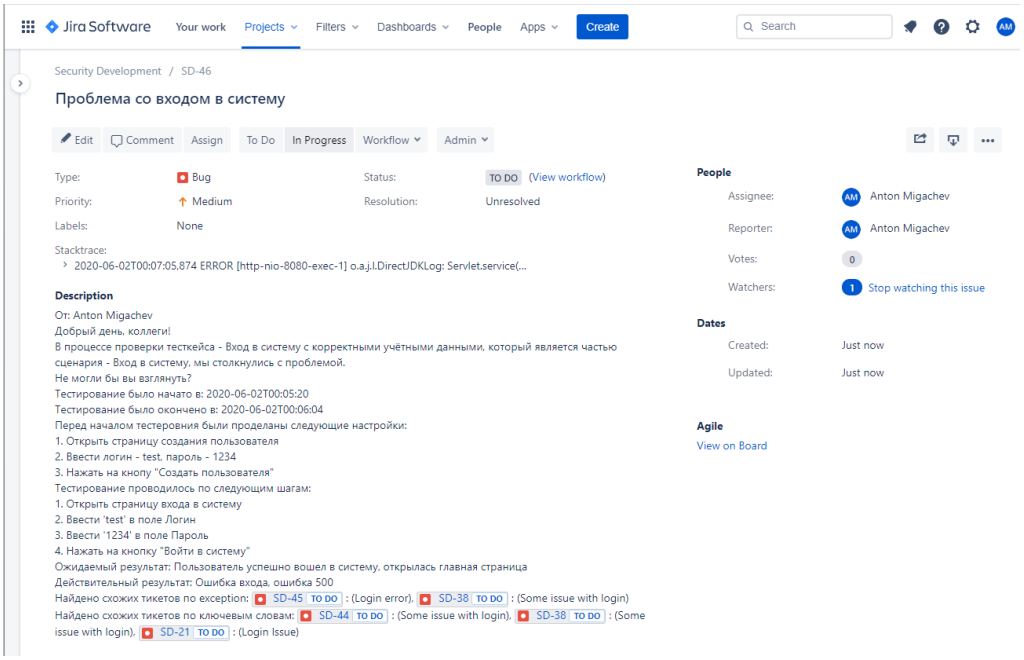


Рисунок 3.22 – Страница отчёта на Jira

После успешной отправки тикета в Jira, на странице отчета в системе будет доступна ссылка (рисунок 3.23), перейдя по которой можно открыть этот отчёт в багтрекинг системе, таким образом появляется связь между отчетом и тикетом между системами.

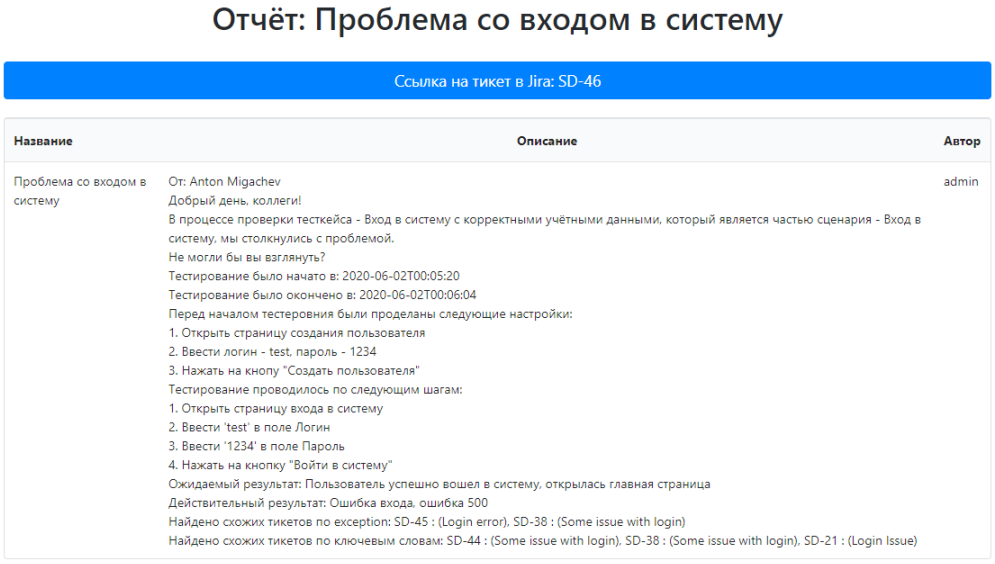

Рисунок 3.23 – Страница отчёта после отправки в Jira

## ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была спроектировано и разработано программное обеспечение, которое позволяет автоматизировать процесс анализа ошибок в приложении исходя из исключений в логах, которые возникают в процессе тестирования. Также оно позволяет осуществлять управление и учёт проектов и тестовых сценариев, предоставляет возможность их запуска и автоматической генерации отчёта по их завершению.

В ходе разработки программного обеспечения были осуществлены сбор и анализ предметной области в сфере анализа отчётов об ошибках, а также устройство и назначение багтрекинг систем, применены знания проектирования программного обеспечения и создания базы данных, а также разработки информационных систем.

Внедрение такого программного обеспечения позволит облегчить работу участникам процесса разработки и тестирования в IT-компании, экономить время на написание отчетов об ошибках, донесении необходимого количества информации до разработчиков, которые должны будут заниматься исправлением той или иной проблемы

Программное обеспечение разработано на платформе Java с применением технологии Spring Web MVC для создания клиент-серверного веб-приложения используя архитектурный шаблон MVC. Клиентская часть приложения

была реализована на языке разметки HTML с применением шаблонизатора Thymeleaf. В качестве СУБД была использована PostgreSQL. Также программное обеспечение поддерживает интеграцию с багтрекинг системой Jira используя REST API.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ


1. Тестирование. Фундаментальная теория [Электронный ресурс]: Режим доступа: https://habr.com/ru/post/279535/ (дата обращения: 14.04.2020).

2. Система отслеживания ошибок [Электронный ресурс]: Режим доступа: https://ru.bmstu.wiki/Система_отслеживания_ошибок (Дата обращения 22.03.2020)

3. Обзор популярных систем багтрекинга [Электронный ресурс]: Режим доступа: https://xakep.ru/2014/10/08/bug-tracking-systems/ (Дата обращения 22.03.2020)

4. Atlassian Jira [Электронный ресурс]: Режим доступа: https://www.atlassian.com/ru/software/jira (дата обращения: 15.03.2020).

5. YouTrack [Электронный ресурс]: Режим доступа: https://www.jetbrains.com/ru-ru/youtrack/ (дата обращения: 21.03.2020).

6. What is REST? [Электронный ресурс]: Режим доступа: https://restfulapi.net/ (Дата Обращения: 16.04.2020).

7. О системе контроля версий [Электронный ресурс]: Режим доступа: https://git-scm.com/book/ru/v2/Введение-О-системе-контроля-версий (Дата обращения 22.03.2020)

8. Bugzilla, Mozilla Россия [Электронный ресурс]: https://mozilla-russia.org/products/bugzilla/ (Дата обращения 22.03.2020)

9. Поиск задач в Jira (Простым языком). Продвинутый поиск [Электронный ресурс]: Режим доступа: https://habr.com/ru/company/raiffeisenbank/blog/449932/ (Дата обращения 22.03.2020)

10. Зиндер Е.З.: Проектирование баз данных: новые требования, новые подходы. - [Электронный ресурс]. – 2001 - 2017. URL: http://citforum.ru/database/kbd96/41.shtml (Дата обращения: 30.04.2020).

11. Йенер М. Паттерны проектирования для профессионалов / М. Йенер, А. Фидом. – СПб.: Питер, 2016. – 240с.

12. Модель-Представление-Контроллер (MVC) [Электронный ресурс]. — URL: http://rsdn.ru/article/patterns/generic-mvc.xml (Дата обращения: 25.05.2020)

13. Гэвин Кинг и Кристиан Бауэр, Java Persistence with Hibernate / Гэвин Кинг и Кристиан Бауэр, 2006. 876 с.

14. Michael Good. Thymeleaf with Spring Boot: An easy to follow guide / Michael Good. – 2019. -79с.

15. Web MVC framework. [Электронный ресурс] – URL https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html (Дата обращения: 20.04.2020)

16. Б. Кришнамурти, Дж. Рексфорд, Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика/ Б. Кришнамурти, Дж. Рексфорд, – Бином, 2002. -592с

17. Spring Data JPA - Reference Documentation [Электронный ресурс] – URL https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference (Дата обращения: 20.04.2020)

18. Герберт Шилдт Вильямс. Java. Полное руководство. 8-е издание/ Герберт Шилдт М.: Вильямс, 2017.- 1376с.

19. Уоллс Крейг. Spring в действии, 5-е издание/ Уоллс Крейг, 2015 ДМК Пресс.- 754с.

20. Объекты доступа к данным (DAO) [Электронный ресурс]. — URL: http://docs.spring.io/autorepo/docs/spring/4.2.x/spring-framework-reference/ html/dao.html. (Дата обращения 15.05.2020).

21. Е.П. Моргунов. PostgreSQL. Основы языка SQL / Уоллс Крейг, 2019 БХВ-Петербург.- 336с.

22. Роберт Уинч, Spring Security 3-е издание / Роберт Уинч, 2012 РАСКТ, -456с.

23. Брюс У. Перри, Java сервлеты и JSP. Сборник рецептов / Брюс У. Перри, 2006, КУДИЦ-Образ. -768с.

24. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж - СРб.: Питер, 2008 – 336 с.

25. Хеник, Б. HTML и CSS. Путь к совершенству / Б. Хеник. — СанктПетербург: Питер, 2011.

26. Bauer, C. Java Persistence with Hibernate, second edition / C. Bauer, G. King, G. Gregory. — Shelter Island: Manning, 2016.

27. Шеффер, К. Spring 4 для профессионалов / К. Шеффер, К. Хо, Р. Харроп. Киев: Вильямс, 2015.

28. Walls, C. Spring Boot In Action / C. Walls. — Shelter Island: Manning, 2016.

29. Пьюривал, С. Основы разработки веб-приложений / С. Пьюривал. — Санкт-Петербург: Питер, 2015

30. Spring Documentation [Электронный ресурс]. — URL: https://docs.spring.io/ (Дата обращения 10.05.2020).

# Настройки интеграции приложения с Jira

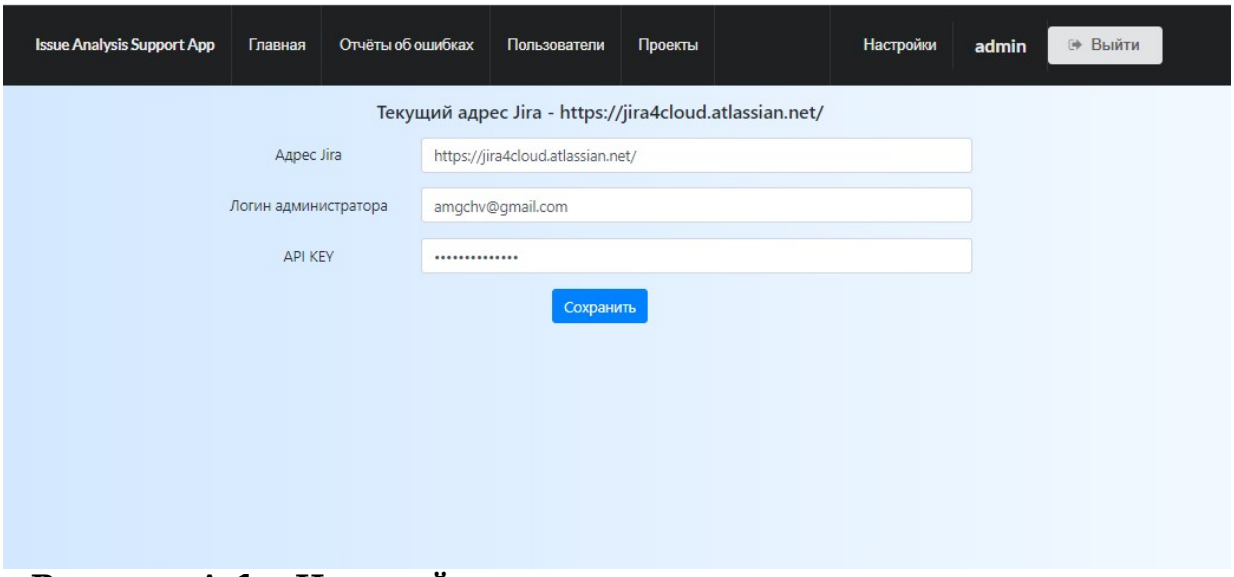


**Рисунок А.1 – Настройки интеграции на стороне приложения.**

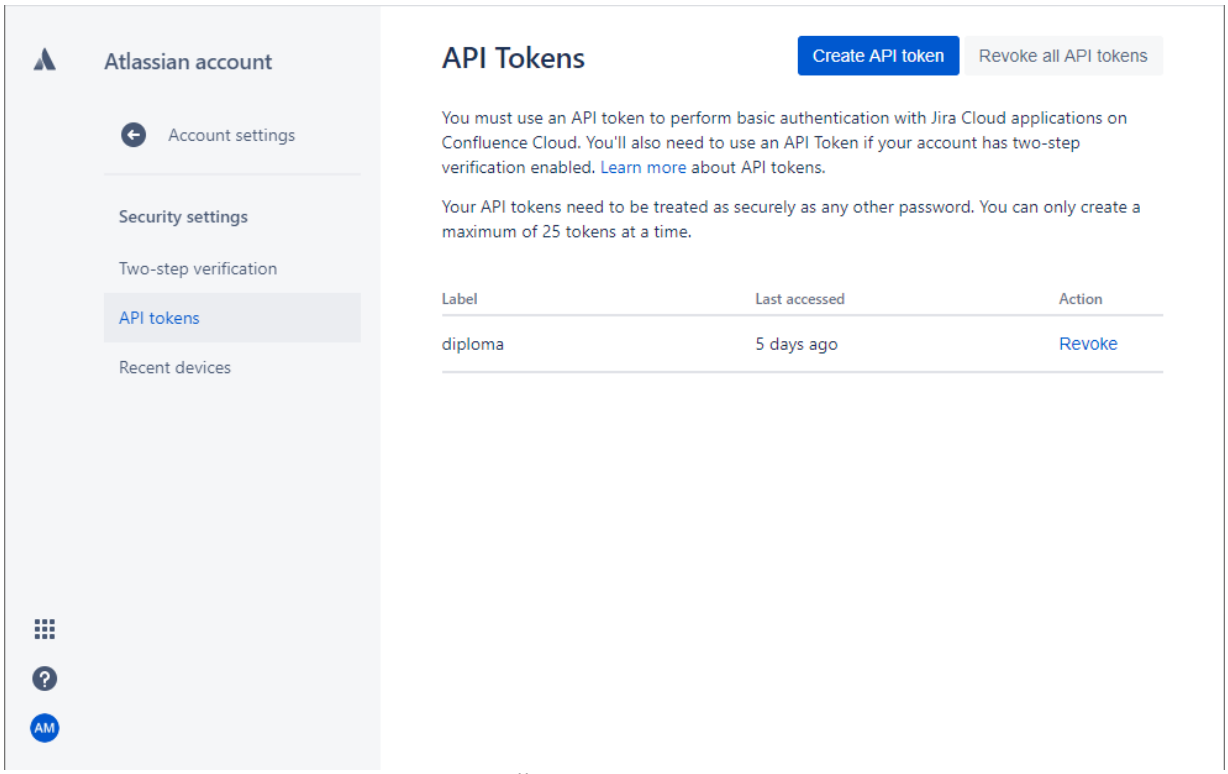


**Рисунок А.2 – Настройки интеграции на стороне Jira.**

## Исходный код класса FindJiraIssueServiceImpl

### Листинг Б.1 – Исходный код сервиса по поиску Issue в Jira

```
package com.amgchv.services.impl;

import
com.amgchv.models.jira.response.search.jql.IssueJqlSearchResponse
;
import
com.amgchv.models.jira.response.search.picker.SearchResponse;
import com.amgchv.services.FindJiraIssueService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestOperations;

import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
public class FindJiraIssueServiceImpl implements
FindJiraIssueService {

    private static final String SEARCH_ISSUE_ENDPOINT =
"/rest/api/3/issue/picker?
query={query}&currentProjectId={projectId}";
    private static final String SEARCH_BY_JQL_ISSUE_ENDPOINT =
"/rest/api/3/search?jql=text~\"{exception}\"&fields=summary";

    private final RestOperations restOperations;

    @Override
    public List<String> getIssuesFromJiraByExceptions(List<String>
exceptions, String projectId) {
        return getSearchByJqlResults(exceptions, projectId);
    }
    @Override
    public List<String> getIssuesFromJiraByKeywords(List<String>
keywords, String projectId) {
```

```
        return getSearchResults(keywords, projectId);
    }
```

**ПРИЛОЖЕНИЕ Б (стр. 2)**

```
    private List<String> getSearchResults(List<String>
searchElements, String projectId) {
        List<String> issuesFromJiraByKeywords = new ArrayList<>();

        for (String keyword : searchElements) {
            SearchResponse searchResponse =
restOperations.getForObject(SEARCH_ISSUE_ENDPOINT,
                    SearchResponse.class, keyword, projectId);
            if (searchResponse != null) {
                searchResponse.getSections()
                        .forEach(section ->
section.getIssues().stream().map(jiraIssue ->
                                jiraIssue.getKey() + " : (" +
jiraIssue.getSummaryText() + ")")
                                .forEach(issuesFromJiraByKeywords::add));
            }
        }

        return issuesFromJiraByKeywords;
    }

    private List<String> getSearchByJqlResults(List<String>
searchElements, String projectId) {
        List<String> issuesFromJiraByKeywords = new ArrayList<>();

        for (String element : searchElements) {
            IssueJqlSearchResponse searchResponse =
restOperations.getForObject(SEARCH_BY_JQL_ISSUE_ENDPOINT,
                    IssueJqlSearchResponse.class, element, projectId);
            if (searchResponse != null) {
                searchResponse.getIssues().stream().map(issue ->
                        issue.getKey() + " : (" +
issue.getFields().getSummary() + ")")
                        .forEach(issuesFromJiraByKeywords::add);
```

```
            }
        }

        return issuesFromJiraByKeywords;
    }
}
```

ПРИЛОЖЕНИЕ В

## Исходный код класса ApplicationController

### Листинг В.1 – Исходный код главного контроллера приложения

```
package com.amgchv.controllers;

import com.amgchv.models.User;
import com.amgchv.services.UserService;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.AuthenticationManager
;
import
org.springframework.security.authentication.UsernamePasswordAut
henticationToken;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.web.authentication.WebAuthentication
Details;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.HttpServletRequest;

@Controller
```

```
@RequiredArgsConstructor
public class ApplicationController {

    private final UserService userService;
    private final AuthenticationManager authenticationManager;

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/login")
    public String login() {
        return "login/index";    }
```

**ПРИЛОЖЕНИЕ В (стр. 2)**

```
    @GetMapping("/signup")
    public String signup() {
        return "signup/index";
    }

    @PostMapping("/signup")
    public String signup(User user, @RequestParam("password")
String password, HttpServletRequest request) {
        userService.register(user, "guest");
        authenticateUserAndSetSession(user, password, request);
        return "redirect:/";
    }

    private void authenticateUserAndSetSession(User user, String
password, HttpServletRequest request) {
        String username = user.getAccount();
        UsernamePasswordAuthenticationToken token = new
UsernamePasswordAuthenticationToken(username, password);

        request.getSession();

        token.setDetails(new WebAuthenticationDetails(request));
```

```
        Authentication authenticatedUser =
authenticationManager.authenticate(token);


SecurityContextHolder.getContext().setAuthentication(authenticated
User);
    }
}
```