СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2

1 ОБЗОР ЛИТЕРАТУРЫ 4

1.1 Обзор аналогов 4

1.2 Взаимодействие c Telegram 5

1.3 Взаимодействие с JIRA 8

1.4 Взаимодействие с TeamCity 11

1.5 Выбор СУБД 14

1.6 Выводы 16

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 17

2.1 Структура программного средства 17

2.2 Развертывание программного средства 21

2.3 Моделирование данных программного средства 21

2.4 Выводы 21

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 21

3.1 Описание работы фреймворка 21

3.3 Описание структуры и взаимодействия между классами 21

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 22

4.1 Алгоритм 22

4.2 Алгоритм 22

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 23

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 25

6.1 Установка системы на удаленном сервере 25

6.2 Описание основных функций программного средства 25

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА 26

7.1 Характеристика программного продукта 26

7.2 Расчет сметы затрат, цены и прибыли 26

7.3 Расчет экономического эффекта у пользователя 26

ЗАКЛЮЧЕНИЕ 28

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29

ПРИЛОЖЕНИЕ А 30

ПРИЛОЖЕНИЕ Б 31

ПРИЛОЖЕНИЕ В 32

ПРИЛОЖЕНИЕ Г 33

# ВВЕДЕНИЕ

С постоянным развитием информационных технологий, возрастают как требования к производимому продукту, так и его сложность. С увеличением сложности продукта, растет и ответственность за организацию работы. Крупный проект требует вовлечения множества разнообразных специалистов, что требует налаживания эффективного взаимодействия в команде. Помимо процессов взаимодействия в команде неотъемлемой частью разработки являются немаловажные процессы сборки и развертывания программного продукта, его тестирование, управление задачами и отслеживание ошибок. Для решения этих задач разработано множество разнообразных программных средств, которыми интенсивно пользуются сотрудники предприятий.

Данный дипломный проект представляет собой программное средство, основанное на системе мгновенного обмена сообщениями Telegram, которое облегчает использование системы непрерывной интеграции TeamCity и системы отслеживания задач Atlassian JIRA с помощью непосредственного взаимодействия с ними через чат, а также неявного их взаимодействия между собой.

Перед каждой ИТ-компанией, в ходе её функционирования, становится вопрос распределения задач между работниками, а также отслеживание выполнения этих самых задач, потраченного на них времени, отслеживание ошибок и статуса их исправления.

Управление задачами – это процесс постановки задач исполнителям, промежуточный и итоговый контроль их выполнения. Каждая задача – это шаг для достижения целей компании. Поэтому важно, чтобы все задачи выполнялись качественно и в срок.

Многое усложняет работу руководителя. Могут изменяться сроки, приоритеты и важность задач, а также сами задачи, требуется координировать работы команды, а также отчитываться перед заказчиками о проделанной за некоторый интервал времени работе. Это отнимает много времени и сил. Если отсутствует система управления задачами, работа многократно усложняется. Однако и при использовании такой системы существует необходимость следить за тем, чтобы работники не забывали взаимодействовать с системой: отмечать завершенность задач, указывать затраченное время, давать задачам статус и другое.

Система управления задачами – это набор инструментов для постановки задач и контроля над их выполнением. В данном дипломном проекте будет использоваться в качестве такой системы Atlassian JIRA Software. JIRA Software — это мощная система для отслеживания задач, обладающая невероятно широкими возможностями персонализации. Каждая команда разработчиков по-своему подходит к организации рабочего процесса, поэтому в JIRA Software есть множество готовых шаблонов. Система масштабируема и подходит как для организаций с небольшим количеством сотрудников (менее 10 человек), так и для крупных предприятий (до 10000 человек).

Еще одним немаловажным инструментом команды разработчиков является система непрерывной интеграции. Непрерывная интеграция — это практика выполнении частых автоматизированных сборок проекта. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий.

Система отслеживания задач и система непрерывной интеграции являются важными инструментами команды разработчиков. Но каждая система требует отдельного взаимодействия. Сборка крупного проекта на сервере непрерывной интеграции также часто является длительным процессом. Чтобы объединить взаимодействие с данными системами, а также реализовать уведомления разработчиков об окончании сборки, было решено разработать программное средство на основе средства мгновенного обмена сообщениями.

Средство мгновенного обмена сообщениями, мессенджер — программное средство и протокол передачи данных, позволяющие интернет-пользователям общаться в реальном времени. Обычно мессенджеры включают в себя программы-клиенты, устанавливаемые на локальных компьютерах. Популярными мессенджерами являются: ICQ, Skype, Telegram. Telegram является одним из самых безопасных и популярных сервисов мгновенного обмена сообщениями. Сквозное шифрование трафика здесь является опциональным и включается только если пользователь сам этого пожелает, выбрав функцию «Создать секретный чат» (что, вполне оправдано, учитывая, что порядка 90% трафика не содержит конфиденциальной информации, и шифровать её по умолчанию не требуется). Кроме того, в секретных чатах блокируется снятие скриншотов с экрана, а на сообщения можно установить таймер самоуничтожения. Также программа имеет открытый исходный код, что позволяет убедиться в отсутствии у нее каких-либо недокументированных функций.

В ходе дипломного проектирования планируется разработать программное средство для удаленного управления сборкой приложений и взаимодействия с системой отслеживания задач со следующими функциями:

— запуск сборки проекта из чата;

— отправка уведомлений о результатах сборки проекта в групповой чат (например, успешно или неуспешно завершилась сборка и список изменений, авторы изменений);

— закрытие задач в JIRA, которые указаны в сообщениях коммитов;

— подсвечивание ссылками номеров задач JIRA в текстах новостей о билдах;

— уведомления об изменениях статусов задач в чате;

— назначение и изменение параметров задачи в чате (выставление затраченного времени и изменение статуса).

# **1** ОБЗОР ЛИТЕРАТУРЫ

На сегодняшний день без систем отслеживания задач и систем непрерывной интеграции разработка немыслима, а системы мгновенного обмена сообщениями прочно вошли в повседневную жизнь людей во всем мире. Непосредственно перед проектированием средства по обеспечению взаимодействия вышеупомянутых систем имеет смысл оценить текущее состояние предметной области, выяснить положительные и отрицательные стороны для каждой отдельной системы и проанализировать способы взаимодействия с ними.

## **1.1** Обзор аналогов

Программ-аналогов для этой области нет, многие процессы только начинают автоматизировать посредством чат-ботов, поэтому ниже будут рассмотрены различные боты, сферы их применения, а также последние тенденции развития в данном направлении.

Чат-бот — это виртуальный собеседник, который общается с человеком на его языке и выполняет различные команды. Сегодня посредством ботов в мессенджере пользователь может получить информацию по прогнозу погоды, приобрести билет на самолет, вызвать такси, записаться на прием к врачу, сделать заказ в ресторане и даже откликнуться на заинтересовавшую вакансию — список предоставляемых услуг постоянно дополняется новыми сервисами. Также существуют множество ботов, предоставляющих развлекательные сервисы, будь то юмористические рассказы или полноценные игры.

Чат-боты экономят время, не требуют трафика для скачивания и установки, не занимают место в памяти смартфона. Для того чтобы совершить операцию в каком-либо приложении, часто требуется сделать с десяток кликов, в то время как в мессенджере достаточно написать одно сообщение.

Все необходимое можно получить в одном месте — именно так, по прогнозам, и будут выглядеть платежные сервисы в скором будущем. Эксперты считают, что золотой век мобильных приложений уходит в прошлое. Например, сегодня американцы в среднем за месяц не загружают ни одного нового приложения. К 2019 году 20% компаний откажутся от своих мобильных приложений [1].

Рассмотрим некоторых чат-ботов поподробнее.

BYNgo. Компания «Системные технологии», которая занимается разработкой программных продуктов для банков и предприятий, выпустила свой чат-бот в Telegram. Имя бота — BYNgo — именно так его можно найти в мессенджере. Пользоваться им можно независимо от того, клиентом какого банка является пользователь.

Бот позволяет оплатить мобильный телефон, ТВ, услуги интернет-провайдера и коммунальные услуги, оплатить штрафы ГАИ, пополнить баланс на банковских картах и электронных кошельках.

BYNgo с заданной периодичностью опрашивает сервис ЕРИП и выявляет, есть ли задолженности у пользователя, подписавшегося на рассылку. При наличии задолженности пользователю отправляется соответствующее push-уведомление со ссылкой на оплату.

Альта. Бот-помощник, который помогает контролировать расходы и подбирать кредит. Бот «Альта» в Telegram выполняет функции «личного финансового помощника». Он разработан группой студентов. «Альта» не рассказывает об услугах и продуктах конкретного банка, а оказывает финансовую помощь в целом [1]. Например, помогает следить за расходами и доходами, прогнозировать будущие траты или сбережения, дает советы и рекомендации о том, как лучше распорядиться деньгами, конвертирует валюту, рассчитывает суммы кредитов и депозитов.

Главное, что отличает «Альту» от банковских ботов, — с ней можно взаимодействовать, отправляя сообщения, а не путем выбора специальных команд. Отвечает бот соответствующе. Например, если попросить рассчитать кредит, «Альта» ответит: «Деньги всем нужны, с этим не поспоришь…». В случае же недостаточного заработка пользователя, «Альта» даст дружеский совет: «Я не советую тебе брать кредит, у тебя слишком маленькие доходы для этого».

Кроме того, «Альта» может провести ликбез по финансовым вопросам. Если спросить, что значит тот или иной экономический термин, бот пояснит его максимально простым языком.

Чат-бот БТА Банка в Telegram работает как «круглосуточный виртуальный помощник». Здесь пользователь может узнать текущие курсы валют, адреса отделений и банкоматов, уточнить информацию по продуктам и услугам банка. Например, пользователь может отправить боту свое местоположение, а в ответ он выдаст адреса ближайших отделений с актуальными курсами валют в них.

Также существуют множество других чат-ботов, целью которых является предоставление информации о погоде, расположения ближайших кафе или ресторанов, информации о текущей киноафише, грамматической проверке отосланного боту сообщения, конвертации документов и др. Многие новостные порталы также имеют своих чат-ботов.

Несмотря на популярность, у чат-ботов существует ряд серьезных недостатков. Например, проблемы с обработкой естественного языка пользователей — один из главных факторов медленного развития популярности ботов. Ведь качественный бот должен вести себя как «живой» собеседник.

## **1.2** Взаимодействие c Telegram

Telegram — бесплатный [кроссплатформенный](https://ru.wikipedia.org/wiki/%252525252525252525252525D0%2525252525252525252525259A%252525252525252525252525D1%25252525252525252525252580%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D0%252525252525252525252525BF%252525252525252525252525D0%252525252525252525252525BB%252525252525252525252525D0%252525252525252525252525B0%252525252525252525252525D1%25252525252525252525252582%252525252525252525252525D1%25252525252525252525252584%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D1%25252525252525252525252580%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B5_%252525252525252525252525D0%252525252525252525252525BF%252525252525252525252525D1%25252525252525252525252580%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B3%252525252525252525252525D1%25252525252525252525252580%252525252525252525252525D0%252525252525252525252525B0%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B5_%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B1%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D0%252525252525252525252525BF%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D1%25252525252525252525252587%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525B8%252525252525252525252525D0%252525252525252525252525B5) [мессенджер](https://ru.wikipedia.org/wiki/%252525252525252525252525D0%252525252525252525252525A1%252525252525252525252525D0%252525252525252525252525B8%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D1%25252525252525252525252582%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B0_%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B3%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B2%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B3%252525252525252525252525D0%252525252525252525252525BE_%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B1%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525B0_%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525BE%252525252525252525252525D0%252525252525252525252525B1%252525252525252525252525D1%25252525252525252525252589%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525B8%252525252525252525252525D1%2525252525252525252525258F%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B8) для смартфонов и других устройств, позволяющий обмениваться текстовыми сообщениями и медиафайлами различных форматов. Количество активных пользователей сервиса на февраль 2016 года составляло более 100 млн человек, а количество ежедневно пересылаемых сообщений достигло 10 миллиардов на август 2015 [2].

При помощи специального API сторонние разработчики могут создавать «ботов», специальные аккаунты, управляемые программами, созданные для того, чтобы автоматически обрабатывать и отправлять сообщения. По сути, эти аккаунты играют роль интерфейса к сервису, который работает на удалённом сервере. Боты отвечают на специальные команды в персональных и групповых чатах, также они могут осуществлять поиск в интернете или выполнять иные задачи, применяются в развлекательных целях и в бизнесе.

Для создания бота следует написать сообщение пользователю @BotFather. BotFather — это бот, который занимается регистрацией других ботов. При помощи него также меняются настройки у существующих ботов. Чтобы создать нового робота следует написать команду **/**newbot. Далее BotFather спросит имя нового бота и имя пользователя. Имя будет отображаться в контактах и чатах. Имя пользователя — короткое имя на латинице, которое используется для упоминаний бота. Имя пользователя должно состоять из букв латинского алфавита, подчёркиваний и цифр и быть длиной от 5 до 32 символов. Также имя пользователя обязательно должно заканчиваться на «bot», например: «tetris\_bot» или «TetrisBot». После этого BotFather предоставит ключ авторизации (токен). Ключ авторизации — это набор символов вида 110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw, который нужен, чтобы получать и отправлять сообщения с помощью Bot API [3].

Bot API представляет из себя HTTP-интерфейс для работы с ботами в Telegram. Все запросы к Telegram Bot API должны осуществляться через HTTPS в следующем виде:

https://api.telegram.org/bot<token>/НАЗВАНИЕ\_МЕТОДА.

Допускаются GET и POST запросы. Для передачи параметров в Bot API доступны 4 способа:

— GET-запрос;

— application/x-www-form-urlencoded;

— application/json (не подходит для загрузки файлов);

— multipart/form-data (для загрузки файлов).

Ответ придёт в виде JSON-объекта, в котором всегда будет булево поле ok и опциональное строковое поле description, содержащее описание результата. Если поле ok истинно, запрос прошёл успешно и результат его выполнения будет находиться в поле result. В случае ошибки поле ok будет равно false, а причины ошибки будут описаны в поле description. Кроме того, в ответе будет присутствовать целочисленное поле error\_code, но коды ошибок будут изменены в будущем.

Существует два диаметрально противоположных по логике способа получать обновления от бота: длинные опросы и веб-хуки. Независимо от способа получения обновлений, в ответ будет получен объект Update, сериализованный в JSON. Этот объект представляет из себя входящее обновление. Под обновлением подразумевается действие, совершённое с ботом — например, получение сообщения от пользователя. Принципиальное отличие: при длинных опросах приложению самому нужно запрашивать обновления у API, а используя веб-хуки — сервера Telegram будут отправлять на сервер каждое обновление с помощью HTTPS POST-запроса [4].

Длинные опросы, другое название способа — «очередь ожидающих запросов». Способ заключается в периодическом опросе серверов Telegram на предмет наличия новой информации. Схема работы такова (см. рис. 1.1) [5]:

1) Клиент отсылает запрос на сервер.

2) Сервер, вместо того, чтобы быстро обработать этот запрос и отправить ответ клиенту, запускает цикл, в каждой итерации которого следит за возникновением событий (другой клиент добавил запись или удалил).

3) При возникновении события сервер генерирует ответ и отсылает его клиенту, таким образом завершая запрос.

4) Клиент, получив ответ от сервера, запускает обработчик события и параллельно отправляет очередной «длинный» запрос серверу.



Рисунок 1.1 — Схема коммуникации сервера и клиента

способом длинных опросов

Веб-хуки — механизм оповещения пользователей системы о событиях. Используя веб-хуки — сервера Telegram будут отправлять на сервер каждое обновление с помощью HTTPS POST-запроса. Отпадает необходимость периодически опрашивать серверы. Однако появляется необходимость установки полноценного веб-сервера на тот сервер, на котором планируется запускать ботов. Также необходимо иметь собственный SSL-сертификат, т.к. веб-хуки в Telegram работают только по HTTPS.

Исходя из всего вышеперечисленного, было принято решение использовать способ длинных опросов.

Для взаимодействия с Telegram будет использоваться библиотека с открытым исходным кодом, реализованная на языке программирования Java, TelegramBots, основанная на Telegram Bot API [6].

Для использования библиотеки следует с помощью фреймворка Apache Maven добавить следующую зависимость:

<dependency>

<groupId>org.telegram</groupId>

<artifactId>telegrambots</artifactId>

<version>2.4.4</version>

</dependency>

Для создания бота, основанного на длинных опросах, следует наследовать собственный класс от класса TelegramLongPollingBot. В свою очередь, для создания бота, использующего веб-хуки, следует наследовать от TelegramWebhookBot. Класс, наследующий TelegramLongPollingBot, должен реализовывать 3 метода:

1) public void onUpdateReceived(Update update). Этот метод будет вызываться всякий раз при получении обновления Update (например, в случае получения сообщения, адресованного боту).

2) public String getBotUsername(). Этот метод должен всегда возвращать имя бота.

3) public String getBotToken(). Этот метод должен всегда возвращать токен бота, полученный у BotFather.

После этого остается проинициализировать контекст API при помощи ApiContextInitializer.init(), создать TelegramBotsApi и зарегистрировать своего бота с помощью метода registerBot() объекта TelegramBotsApi.

## **1.3** Взаимодействие с JIRA

Atlassian JIRA — коммерческая веб-система отслеживания ошибок, предназначенная для отслеживания хода исполнения задач по исправлению ошибок ПО, управлению проектами, управлению ходом исполнения отдельных задач и рабочими процессами. Позволяет отслеживать актуальный статус решения задач в режиме реального времени. Разработана австралийской компанией Atlassian Software Systems. Работа в Atlassian JIRA происходит через web-браузер без установки клиентских приложений на рабочих местах.

Популярными системами такого же рода являются Trello, Asana, Bugzilla, Redmine, Github Issues. Преимуществами Atlassian JIRA перед остальными являются [7]:

1) Универсальность. JIRA задумывалась не для использования тем или иным человеком или ролью, а как единый инструмент, которым сможет пользоваться каждый член команды. С помощью этого инструмента можно взаимодействовать, отслеживать ход работы и быстрее поставлять качественное ПО.

2) Гибкость и возможность расширения. В системе есть множество готовых шаблонов, отличающихся невероятно широкими возможностями персонализации, поэтому она подойдет для любой команды и любого процесса.

3) Масштабирование. В JIRA существует возможность расширения по мере вовлечения новых специалистов в команду.

Atlassian JIRA предоставляет функционал для создания и управления проектами и позволяет разбивать их на этапы, настраивать типы задач, связывать задачи между собой, настраивать возможные переходы между этапами решения задачи, назначать ответственных по различным направлениям. Система позволяет с помощью JIRA Query Language (JQL) искать задания в проекте по целому набору критериев и создавать фильтры, которые можно сохранить и использовать постоянно.

Для организации работы с пользователями Atlassian JIRA позволяет образовывать группы пользователей и назначать им роли. Платформа обладает гибкой системой разграничения и контроля доступа к проектам, задачам и функциям, основанной на членстве пользователей в группах и их ролях.

Для аналитических целей JIRA создает карту проекта, позволяет просматривать загрузку каждого пользователя и позволяет создавать следующие стандартные отчеты для эффективного управления проектами:

— нерешенные высокоприоритетные задачи;

— количество задач, созданных одним пользователем;

— среднее время решения задачи;

— задачи, имеющие определенные статус;

— задачи, имеющие определенный приоритет;

— отчет о нагрузке на разработчиков и другие.

Для взаимодействия с внешними программами Atlassian JIRA предоставляет REST API. REST API предоставляет доступ к ресурсам (сущностям данных) с помощью URI. Чтобы взаимодействовать с REST API приложение должно осуществить HTTP запрос и разобрать полученный ответ. Используются стандартные HTTP методы, такие как GET и POST. Форматом обмена данными с JIRA REST API является формат JSON. URI имеет следующую структуру [8]:

http://host:port/context/rest/api-name/api-version/resource-name

Доступны API для аутентификации auth и для всего остального api. Текущая API-версия (api-version) для auth 1, а для api — 2.

Выделяют следующие способы аутентификации:

1) Basic HTTP. Пользователь аутентифицируется с помощью логина и пароля, что небезопасно, если не настроен доступ к JIRA посредством HTTPS, так как пользовательские данные (имя и пароль) передаются в открытом виде (закодированы в формате Base64).

2) OAuth. Протокол авторизации, который позволяет предоставить третьей стороне ограниченный доступ к защищённым ресурсам пользователя без необходимости передавать ей (третьей стороне) логин и пароль.

Существуют библиотеки для облегчения работы с JIRA REST API, устраняющие необходимость напрямую пользоваться URI, подготавливать запросы и анализировать полученные ответы. Среди таких библиотек можно выделить следующие: JIRA REST Java Client (JRJC), Jira-client от rcarz и Jira-rest-client от lesstif [9-11]. Рассмотрим наиболее популярные из них.

JRJC. Изначально разработана компанией Atlassian, и ныне поддерживается сообществом разработчиков. Имеет открытый исходный код. Данная библиотека поддерживает только Basic HTTP метод аутентификации.

Для использования библиотеки следует с помощью фреймворка Apache Maven добавить следующую зависимость:

<dependency>

<groupId>com.atlassian.jira</groupId>

<artifactId>jira-rest-java-client-core</artifactId>

<version>4.0.0</version>

</dependency>

Для аутентификации следует использовать следующие строки, где URL - это URI системы, USERNAME — имя пользователя, а PASSWORD - его пароль:

JiraRestClientFactory factory = new AsynchronousJiraRestClientFactory();

URI uri = new URI(URL);

JiraRestClient client = factory.createWithBasicHttpAuthentication(uri, USERNAME, PASSWORD);

К недостаткам JRJC следует отнести то, что она не поддерживает переназначение задачи другому исполнителю, изменение/редактирование задачи, кроме изменения статуса [9].

Rcarz/jira-client. Разработал данную библиотеку Bob Carroll. Библиотека имеет открытый исходный код. Поддерживается только Basic HTTP метод аутентификации. Для подключения библиотеки к проекту следует добавить следующую зависимость:

<dependency>

<groupId>net.rcarz</groupId>

<artifactId>jira-client</artifactId>

<version>0.5</version>

<scope>compile</scope>

</dependency>

Для аутентификации следует использовать следующие строки, где URL - это URI JIRA сервера, USERNAME - имя пользователя, а PASSWORD - его пароль:

BasicCredentials creds = new BasicCredentials(USERNAME, PASSWORD);

JiraClient jira = new JiraClient(URL, creds);

Функционал данной библиотеки включает следующие возможности по работе с задачами: получение конкретной задачи, расширенный поиск с помощью JQL, создание и редактирование (включая системные и созданные пользователем поля), изменение статуса, добавление комментариев и файлов, возможность голосовать, начинать или прекращать наблюдение, добавлять или удалять ссылки на другие задачи, создавать подзадачи. Также реализована поддержка GreenHooper и Agile API [10].

Lesstif/jira-rest-client. Разрабатывает данную библиотеку KwangSeob Jeong. На данный момент поддерживается только Basic HTTP метод аутентификации. Для подключения библиотеки к проекту следует с помощью фреймворка Apache Maven добавить следующую зависимость:

<dependency>

<groupId>com.lesstif</groupId>

<artifactId>jira-rest-api</artifactId>

<version>0.8.0</version>

</dependency>

Для аутентификации следует создать файл jira-rest-client с расширением properties в директории CLASS\_PATH, и занести в него следующую информацию:

jira.server.url="URL"

jira.user.id="USERNAME"

jira.user.pwd="PASSWORD"

где URL - это URI JIRA сервера, USERNAME - имя пользователя, а PASSWORD - его пароль.

Библиотека на данный момент является незавершенной. Среди реализованного функционала можно выделить получение информации об определенном проекте, получения списка всех проектов, получение информации о конкретной задаче, создание новой задачи, прикрепление к задаче файла, получение всех типов задач, получение всех возможных для задачи приоритетов, получение всех полей задачи, созданных пользователем.

К нереализованному функционалу можно отнести обновление полей задачи, изменение статуса задачи, расширенный поиск с помощью JQL, а также указания затраченного на задачу времени [11].

Исходя из всего вышеописанного можно сделать вывод, что самой развитой в плане функционала библиотекой является rcarz/jira-client. Именно она будет использовать для написания приложения. Все три библиотеки поддерживают только Basic HTTP метод аутентификации, что в рамках данного дипломного проекта является не критичным, поскольку в корпоративной среде доступ к системе JIRA как правило происходит исключительно по HTTPS протоколу.

## **1.4** Взаимодействие с TeamCity

TeamCity — серверное программное обеспечение от компании JetBrains, написанное на языке Java, билд-сервер для обеспечения непрерывной интеграции. Непрерывная интеграция — это практика разработки программного обеспечения, в которой разработчики фиксируют изменения кода в общем репозитории несколько раз в день. За каждой фиксацией следует автоматизированная сборка, которая гарантирует, что новые изменения хорошо впишутся в существующую базу кода. TeamCity поддерживает множество систем управления версиями, среди которых Subversion, CVS, Git. Среди возможностей TeamCity стоит отметить следующие [12]:

1) Предварительное тестирование кода перед коммитом. Предотвращает возможность коммита программного кода, содержащего ошибки, нарушающего нормальную сборку проекта, путём удалённой сборки изменений перед коммитом.

2) Грид-сборка проекта. Предоставляет возможность производить несколько сборок проекта одновременно, производя тестирование на разных платформах и в различном программном окружении.

3) Интеграция с системами оценки покрытия кода, инспекции кода и поиска дублирования кода.

4) Интеграция с различными средами разработки: Eclipse, IntelliJ IDEA, Visual Studio.

5) Поддержка различных платформ: Java, PHP, .NET и Ruby.

Сторонние приложения могут взаимодействовать с TeamCity посредством REST API. Приложение отправляет HTTP запрос к TeamCity серверу и получает результаты выполнения запроса в ответ.

Поддерживаются следующие HTTP методы:

— GET (получение данные);

— POST (создание данных);

— PUT (обновление данных);

— DELETE (удаление данных).

TeamCity REST API позволяет обмениваться данными в следующих форматах [13]:

1) Простой текст. В HTTP Accept заголовке указывается text/plain. Используется для получения единственного значения.

2) XML. В HTTP Accept заголовке указывается application/xml. Используется для получения множества значений.

3) JSON. В HTTP Accept заголовке указывается application/json. Используется для получения множества значений.

URL имеет следующую структуру [13]:

teamcityserver:port/<authType>/app/rest/<apiVersion>/<restApiPath>?<parameters>, где

— teamcityserver и port определяют имя сервера и порт, используемые TeamCity;

— <authType> (необязательный) определяет используемый тип аутентификации;

— app/rest. Корневой путь TeamCity REST API;

— <apiVersion> (необязательный) представляет собой ссылку на конкретную версию REST API;

— <restApiPath>?<parameters> является REST API частью URL.

Существуют следующие способы аутентификации:

1) Basic HTTP. Пользователь аутентифицируется с помощью логина и пароля. Для этого выбирается httpAuth тип аутентификации.

2) Гостевой пользователь. Данный способ должен быть предварительно разрешен администратором. Тип аутентификации — guestAuth. Гостевой пользователь имеет права только для просмотра информации о проектах (настраивается администратором).

Так как REST API постоянно разрабатывается и улучшается, в протоколе могут быть несовместимые с предыдущими версиями изменения. Политика TeamCity REST API обеспечивает поддержку минимум одной предыдущей версии.

Для доступа к последней версии URL имеет следующую структуру:

— teamcityserver:port/app/rest/;

— teamcityserver:port/app/rest/latest.

Предыдущие версии могут быть доступны через teamcityserver:port/app/rest/<apiVersion>.

Иногда требуется получить информацию о каком-то конкретном проекте или конкретной сборке. Для этого REST API предоставляет локатор — строку, которая содержит параметры, по которым будет отфильтрован ответ. Формат локатора может быть следующим:

— единичное значение, а именно строка без символов «,:-()»;

— перечисление значений, для формирования запроса по множеству критериев вида: <criteria1>:<value1>,<criteria2>:<value2> и т. д.

Для программной реализации взаимодействия с TeamCity REST API будет использован фреймворк Jersey. Jersey предоставляет свои собственные API, которые расширяют набор инструментальных средств JAX-RS дополнительными функциями и утилитами для упрощения разработки RESTful сервиса и клиента [14].

Для получения возможности использования данного фреймворка в проекте следует с помощью фреймворка для сборки проектов Apache Maven добавить следующую зависимость:

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-server</artifactId>

<version>${jersey.version}</version>

</dependency>

Для аутентификации следует использовать следующие строки, где USERNAME — имя пользователя, а PASSWORD — его пароль:

HttpAuthenticationFeature feature =

HttpAuthenticationFeature.basicBuilder()

.credentials(USERNAME, PASSWORD)

.build();

ClientConfig сonfig = new ClientConfig();

config.register(feature);

Client client = ClientBuilder.newClient(config);

Для получения информации о каком-то конкретном проекте по PROJECT\_ID, где URL — это URI системы:

WebTarget webTarget = client

.target("URL/httpAuth/app/rest/")

.path("projects")

.path("id:PROJECT\_ID");

Invocation.Builder invocationBuilder =

webTarget.request(MediaType.APPLICATION\_XML);

Response response = invocationBuilder.get();

Project project = response.readEntity(Project.class);

Для запуска сборки по BUILD\_ID, где URL — это URI системы:  
 WebTarget webTarget = client

.target("URL/httpAuth/action.html?add2Queue=BUILD\_ID");

Invocation.Builder invocationBuilder = webTarget.request();

Готовых библиотек для взаимодействия с TeamCity, устраняющих необходимость напрямую обращаться к REST API, подготавливать запросы и анализировать полученные ответы, нет. Поэтому для реализации взаимодействия ранее перечисленного в данном дипломном проекте будет использован фреймворк Jersey.

## **1.5** Выбор СУБД

Базы данных — это логически смоделированные хранилища любых типов данных. Реляционные системы реализуют реляционную модель работы с данными, которая определяет всю хранимую информацию как набор связанных записей и атрибутов в таблице. СУБД такого типа используют таблицы для хранения и работы с данными. Каждый столбец содержит свой тип информации. Каждая запись в базе данных, обладающая уникальным ключом, передаётся в строку таблицы, и её атрибуты отображаются в столбцах таблицы. Каждый элемент, формирующий запись, должен удовлетворять определённому типу данных (целое число, дата и т.д.). Различные РСУБД используют разные типы данные, которые не всегда взаимозаменяемы. Рассмотрим подробнее наиболее популярные реляционные СУБД [15].

SQLite — это библиотека, встраиваемая в приложение, которое её использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных. Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не интерфейса, что повышает скорость и производительность операций.

Поддерживаемые типы данных в SQLite: NULL-значение, INTEGER.(целое со знаком), REAL (число с плавающей запятой), TEXT (текстовая строка с кодировкой UTF-8, UTF-16BE или UTF-16LE), BLOB (тип данных, хранящийся точно в таком же виде, в каком и был получен).

Преимущества данной СУБД:

1) Вся база данных хранится в одном файле, что облегчает перемещение.

2) SQLite использует SQL; некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), однако, есть и некоторые новые.

Недостатки СУБД:

1) Одним из ограничений SQLite являются операции записи. Эта РСУБД допускает единовременное исполнение лишь одной операции записи.

2) Не подходит для многопользовательских приложений.

MySQL — это одна из самых популярных из всех крупных серверных БД. Является очень простой, также для неё существует большое количество документации. В MySQL SQL-стандарты реализованы не полностью. Приложения общаются с базой данных через процесс-демон.

Поддерживаемые типы данных в MySQL: TINYINT, SMALLINT, MEDIUMINT, INT или INTEGER, BIGINT, FLOAT (знаковое число с плавающей запятой одинарной точности), DOUBLE или REAL (знаковое число с плавающей запятой двойной точности), DECIMAL или NUMERIC (знаковое число с плавающей запятой), DATE, DATETIME (комбинация даты и времени), TIMESTAMP (отметка времени), TIME, YEAR (год в формате YY или YYYY), CHAR (строка фиксированного размера, дополняемая справа пробелами до максимальной длины), VARCHAR (строка переменной длины), TINYBLOB, TINYTEXT (BLOB- или TEXT-столбец длиной максимум 255 символов), BLOB, TEXT (длина максимум 65535 символов), MEDIUMBLOB, MEDIUMTEXT (длина максимум 16777215 символов), LONGBLOB, LONGTEXT (длина максимум 4294967295 символов), ENUM (перечисление), SET (множества).

Преимущества данной СУБД:

1) Простота. MySQL легко устанавливается. Существует много сторонних инструментов, включая визуальные, облегчающих начало работы с БД.

2) Много функций. MySQL поддерживает большую часть функционала SQL.

3) Безопасность. Встроено много функций безопасности.

4) Мощность и масштабируемость. MySQL может работать с действительно большими объёмами данных, и неплохо походит для масштабируемых приложений.

5) Скорость. Пренебрежение некоторыми стандартами позволяет MySQL работать производительнее.

Недостатки MySQL:

1) SQL-совместимость. Поскольку MySQL не пытается полностью реализовать стандарты SQL, она не является полностью совместимой с SQL. Из-за этого могут возникнуть проблемы при интеграции с другими РСУБД.

2) Конкурентность. Одновременные операции чтения-записи могут вызвать проблемы.

PostgreSQL (или Postgres) — это самая продвинутая РСУБД, ориентирующаяся в первую очередь на полное соответствие SQL-стандартам ANSI/ISO и расширяемость. PostgreSQL отличается от других РСУБД тем, что обладает объектно-ориентированным функционалом, в том числе полной поддержкой концепта ACID (Atomicity, Consistency, Isolation, Durability).

Будучи основанным на мощной технологии Postgres отлично справляется с одновременной обработкой нескольких запросов. Поддержка конкурентности реализована с использованием MVCC (Multiversion Concurrency Control), что также обеспечивает совместимость с ACID.

Хотя эта РСУБД не так популярна, как MySQL, существует много сторонних инструментов и библиотек для облегчения работы с PostgreSQL.

Поддерживаемые типы данных: bigint, bigserial (автоматически инкрементируемое 8-битное целое), bit (битовая строка фиксированной длины), bit varying (битовая строка переменной длины), boolean, box (прямоугольник на плоскости), bytea (бинарные данные), character varying (строка символов переменной длины), character, cidr (сетевой адрес IPv4 или IPv6), circle (круг на плоскости), date, double precision (число с плавающей запятой двойной точности), inet (адрес хоста IPv4 или IPv6), integer, line (бесконечная прямая на плоскости), lseg (отрезок на плоскости), macaddr (MAC-адрес), money, path (геометрический путь на плоскости), point (геометрическая точка на плоскости), polygon (многоугольник на плоскости), real, smallint, serial (автоматически инкрементируемое 4-битное целое), text, time, timestamp, tsquery (запрос текстового поиска), tsvector (документ текстового поиска), txid\_snapshot (снэпшот ID пользовательской транзакции), uuid (уникальный идентификатор), xml.

Преимущества:

1) Полная SQL-совместимость.

2) Сообщество. PostgreSQL поддерживается опытным сообществом 24/7.

3) Расширяемость. PostgreSQL можно программно расширить за счёт хранимых процедур.

4) Объектно-ориентированность. PostgreSQL — не только реляционная, но и объектно-ориентированная СУБД.

Недостатки PostgreSQL:

— производительность. В простых операциях чтения PostgreSQL может уступать своим соперникам.

Исходя из всего вышеописанного, для реализации данного дипломного проекта в качестве СУБД была выбрана PostgreSQL, из-за полной SQL-совместимости, надежности, развитой функциональности и, что немало важно, кроссплатформенности.

## **1.6** Выводы

После проведения анализа можно сделать вывод, что программ-аналогов в данной сфере нет, но решений на основе чат-ботов существует множество и их количество постоянно растет. В ходе дипломного проекта планируется разработать программный модуль, который позволит объединить взаимодействие с TeamCity и JIRA с помощью Telegram чат-бота. Для взаимодействия с Telegram будет использована библиотека TelegramBots, основанная на Telegram Bot API. Для реализации взаимодействия с JIRA REST API будет использована библиотека rcarz/jira-client, а для взаимодействия с TeamCity REST API — фреймворк Jersey. Хранение данных будет осуществляться с помощью СУБД PostgreSQL.

# **2** СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Программное средство во время проектирования необходимо разбить на отдельные логически взаимосвязанные модули, что является необходимым условием для обеспечения гибкости структуры программного средства. Таким образом становится возможной выборочная модернизация отдельных частей программного кода, с минимальным влиянием на остальные части проекта, либо вовсе без их изменения.

Для определения логических модулей необходимо определить основные возможности программного средства:

— взаимодействие с пользователем понятными ему способами;

— возможность аутентификации в TeamCity, JIRA;

— возможность взаимодействия с TeamCity, JIRA;

— обработка команд пользователя;

— возможность сохранения пользовательских данных;

— возможность доставить пользователю уведомление о каком-либо событии.

## **2.1** Структура программного средства

Структурная схема представлена в приложении Б. Как видно из структурной схемы, данный дипломный проект состоит из собственно программного средства и внешних систем, с которыми производится взаимодействие. Внешние системы представлены на схеме модулями Telegram, TeamCity и JIRA. Ниже будет рассмотрен каждый модуль подробнее.

1) Telegram. То, с чем непосредственно будет работать пользователь. Текст и команды, введенные пользователем, будут передаваться с помощью Telegram Bot API программному средству для дальнейшей обработки. Уведомления, поясняющие сообщения, сообщения об ошибках и др. также будут переданы пользователю с помощью Telegram.

2) Atlassian JIRA. Система остлеживания задач. Отвечает за учет задач, ошибок и связанной с ними дополнительной информации.

3) TeamCity. Система непрерывной интеграции. Отвечает за сборку проектов, тестирование, выявление ошибок.

На структурной схеме представлено непосредственно программное средство, в котором были выделены следующие модули:

— модуль взаимодействия с Telegram;

— модуль интеграции с TeamCity;

— модуль интеграции с JIRA;

— модуль взаимодействия с пользователем;

— модуль обработки команд из чата;

— модуль отправки уведомлений;

— модуль аутентификации;

— модуль доступа к данным;

— база данных.

Ниже будет подробнее рассмотрено за что каждый модуль будет отвечать и чем будет представлен.

Модуль взаимодействия с Telegram. Данный модуль реализует логику работы с Telegram. Отвечает за прием и отправку сообщений, а также за необходимый для корректной работы чат-бота функционал. Для взаимодействия с Telegram ботами используется Telegram Bot API. Как было определено в разделе 1.2, для программной реализации работы с мессенджером будет использоваться библиотека TelegramBots.

Модуль интеграции с TeamCity. Данный модуль реализует логику работы с TeamCity. В основные задачи модуля входит получение с сервера информации о проектах и сборках, запуск сборки проекта, получение информации о результатах сборки и об авторах внесенных изменений. TeamCity взаимодействует с внешними системами с помощью REST API. Как было определено в разделе 1.4, для программной реализации работы с TeamCity REST API будет использоваться фреймворк Jersey.

Модуль интеграции с JIRA. Данный модуль реализует логику работы с JIRA. Основной функционал направлен на работу с задачами, а именно: завершение открытых и повторное открытие закрытых задач, создание новых задач, изменение параметров, ответственного за исполнение и затраченного времени. JIRA взаимодействует с внешними системами с помощью REST API. Как было определено в разделе 1.3, для программной реализации будет использоваться библиотека rcarz/jira-client.

Модуль аутентификации. Предназначен для аутентификации пользователей во внешних системах, а именно в TeamCity и JIRA, для того чтобы программное средство имело возможность получать от этих систем какую-либо информацию и проводить с ней взаимодействие. Для аутентификации пользователь должен предоставить URI сервера, на котором установлена система, логин и пароль. Данная информация будет передана пользователем через чат Telegram. Для аутентификации в TeamCity и JIRA будет использоваться REST API, метод Basic HTTP.

Модуль отправки уведомлений. Отвечает за получение от внешних систем информации об окончании какого-либо конкретного события, подготовки сообщения, и отправки пользователю уведомления о произошедшем конкретном событии. Для отправки уведомлений пользователю в чат будет использована библиотека TelegramBots, а для взаимодействия с Atlassian JIRA и TeamCity будут использоваться библиотека rcarz/jira-clien и фреймворк Jersey соответственно.

Модуль взаимодействия с пользователем. Реализован с помощью Telegram Bot API. Для взаимодействия пользователя с чат-ботом может использоваться простой текст. Дополнительно Telegram Bot API предоставляет более формальные инструменты, а именно:

1) Команды. Команды имеют следующий синтаксис:

/command [optional] [argument].

Все команды должны начинаться с «/» и иметь длину не более 32-х символов, могут использоваться заглавные буквы, цифры и символы нижнего подчеркивания. Сообщения, начинающиеся с «/», всегда передаются боту. Telegram также поддерживает выпадающие списки команд с описанием (см. рис. 2.1) и подсветку команд в тексте сообщений.



Рисунок 2.1 — Снимок экрана со списком

поддерживаемых ботом команд

2) Клавиатура. Каждый раз, когда бот отправляет сообщение в чат, он может прикрепить к сообщению специальный набор клавиш с предопределенными параметрами ответа. Приложения Telegram, на которых это сообщение будет получено, будут отображать этот набор клавиш пользователю как клавиатуру (см. рис. 2.2). Нажатие любой из клавиш сразу же отправит боту соответствующую команду. Таким образом значительно упрощается взаимодействие пользователя с ботом. Поддерживается простой текст и смайлы.



Рисунок 2.2 — Снимок экрана с набором клавиш

3) Встроенная клавиатура. Некоторые пользователи могут предпочитать делать что-нибудь не отправляя никаких сообщений в чат. В таких случаях могут использоваться встроенные клавиатуры, которые интегрированы непосредственно в сообщения, к которым они принадлежат. В отличие от клавиатур, описанных в предыдущем пункте, нажатие кнопок на встроенных клавиатурах не генерирует сообщения в чат.



Рисунок 2.3 — Снимок экрана со встроенной клавиатурой

Таким образом, данный модуль отвечает за формирование сообщений-ответов пользователю, а также за предоставление отличного от отправки простых текстовых сообщений метода взаимодействия.

Модуль обработки команд из чата. Данный модуль обрабатывает полученные от пользователя через модуль взаимодействия с Telegram сообщения. Сообщения разделяются на предназначенные для бота команды и сторонние сообщения. Сторонние сообщение игнорируются. Команда может быть передана простым текстом или же специальной кнопкой. В случае простого текста по ключевым словам определяется на какую внешнюю систему направлена команда. В случае кнопки и после обработки простого текста команда передается конкретному модулю, отвечающего за работу с определенной внешней системой. В случае неоднозначного трактования или некорректности полученной команды отправляется сообщение, уведомляющее пользователя об этом.

Модуль доступа к данным. Данный модуль реализует логику работы с базой данных, а именно: позволяет обращаться к таблицам в базе данных и выполнять с ними различные операции: чтения и поиск, вставка и удаление объектов, а также вызывать хранимые процедуры. Работа с базой данных производится с использованием технологии *Object-Relational Mapping (ORM)*, которая связывает реляционные базы данных с концепциями объектно-ориентированного программирования. Подход, используемый в приложении, позволяет построить базу данных на основании моделей, описанных в объектно-ориентированном стиле.

База данных. Представляет собой хранилище для данных, полученных от пользователя для аутентификации во внешних системах, а также для данных, полученных в результате работы программного средства. Как было определено в разделе 1.5 пояснительной записки, предпочтительным является использование реляционных баз данных, в частности, PostgreSQL.

## **2.2** Развертывание программного средства

## **2.3** Моделирование данных программного средства

## **2.4** Выводы

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробным образом функционирование разрабатываемого программного модуля.

## **3.1** Описание работы фреймворка

**3.2** Описание модели данных

**3.2.1** Таблица

**3.2.2** Таблица

**3.2.3** Таблица

## **3.3** Описание структуры и взаимодействия между классами

Изучение модели данных, используемых в программном продукте, является одним из основных этапов понимания процесса обработки программой данных что, как известно, является основной конечной целью любой разрабатываемой программной системы. Не менее значимым этапом с этой точки зрения является описание структуры классов и взаимодействия между ними. Для описания структуры классов перечислим основные классы программного модуля с указанием способов их взаимодействия между собой, методов и полей, используя диаграмму классов (см. чертеж ГУИР.400201.003 РР.1) в соответствии с модулями, выделенными на этапе системного проектирования (см. чертеж ГУИР.400201.003 С1).

**3.3.1** Класс

Методы класса:

**3.3.2** Класс

Поля класса:

Методы класса:

**3.3.3** Класс

# **4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программного модуля анализа данных профессиональных социальных сетей использовалось значительное количество сторонних библиотек, что объясняется необходимостью выполнения определенного перечня стандартных задач по работе с API сторонних сервисов, анализу данных и генерации отчетов. Следует, однако, заметить, что социальные сети и иные сервисы схожего назначения в открытый доступ посредством API выносят лишь небольшую часть данных, доступных при помощи веб-интерфейса. Данные особенности работы с данными оформлены в виде специальных соглашений об использовании данных и недвусмысленно упоминают о запрете на автоматизацию обращений к веб-версии сайта с целью получения недостающей части данных. Следует заметить, что данные, доступные публично через удобные для разработчика API, ожидаемо являются в некоторой степени полезными для задач анализа, однако значительно более интересные свойства и характеристики аккаунтов закрыты для использования. При создании настоящего программного модуля ставилась задача получения наиболее интересных видов данных о пользователях сервисов любыми средствами, в том числе и нарушающими пользовательские соглашения и соглашения об использовании данных. Рассмотрим некоторые алгоритмы, позволяющие организовать получение данных, их дальнейший анализ и отображение. Полный листинг наиболее специфических участков программного кода приводится в приложении к настоящей пояснительной записке (см. приложение А).

## **4.1** Алгоритм

## **4.2** Алгоритм

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Одной из наиболее значимых задач в ходе жизненного цикла разработки ПО является тестирование. Тестирование программного обеспечения — это процесс исследования ПО с целью выявления ошибок и определения соответствия между реальным и ожидаемым поведением ПО, осуществляемый на основе набора тестов, выбранных определённым образом. В более широком смысле, тестирование ПО — это техника контроля качества программного продукта, включающая в себя проектирование тестов, выполнение тестирования и анализ полученных результатов.

Каждый этап разработки системы сопровождался написанием большого количества разнообразных тестов. Каждая новая функциональность программы тщательно проверялась различными методами. В ходе разработки были применены такие виды тестирования, как:

* функциональное тестирование;
* тестирование производительности;
* тестирование стабильности;
* тестирование совместимости;
* тестирование интерфейса пользователя.

На сегодняшний день промышленные способы проверки качества программного обеспечения используют различные автоматизированные системы тестирования. В Ruby on Rails такой системой по умолчанию является TestUnit. Данная автоматизированный модуль позволяет проводить тестирование моделей, контроллеров и представлений. В среду Rails встроена на удивление надежная и готовая к использованию система тестирования. Затрачивая минимум усилий, можно задавать воспроизводимые настройки баз данных, отправлять Web-приложениям тестовые HTTP-сообщения и выполнять три вида тестирования: модульное, функциональное и комплексное.

При написании тестов для различных уровней приложения значительно снижаются риски нарушения работоспособности приложения при внесении различных изменений в существующий функционал. Полное покрытие тестами занимает порядка 38% от общего времени работы программиста над проектом, но временные затраты полностью окупаются эффектом надежности и стабильности программного обеспечения. По мере продвижения проекта стоимость устранения дефектов ПО может экспоненциально возрастать. Инструменты статического и динамического анализа помогают предотвратить эти затраты благодаря обнаружению программных ошибок на ранних этапах жизненного цикла ПО.

При написании данной дипломной работы разработка велась по принципу TDD (Test Driven Development), реализующему процесс написания приложения через тестирование. Основополагающий принцип TDD – это написание тестов до написания кода приложения.

Разработка через [тестирование](https://ru.wikipedia.org/wiki/%2525252525252525252525252525D0%2525252525252525252525252525A2%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252581%2525252525252525252525252525D1%252525252525252525252525252582%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B2%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D0%2525252525252525252525252525B5_%2525252525252525252525252525D0%2525252525252525252525252525BF%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B3%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525BC%2525252525252525252525252525D0%2525252525252525252525252525BC%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B3%2525252525252525252525252525D0%2525252525252525252525252525BE_%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B1%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252581%2525252525252525252525252525D0%2525252525252525252525252525BF%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252587%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D1%25252525252525252525252525258F)  — техника [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%2525252525252525252525252525D0%2525252525252525252525252525A0%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525B7%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525B1%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D1%252525252525252525252525252582%2525252525252525252525252525D0%2525252525252525252525252525BA%2525252525252525252525252525D0%2525252525252525252525252525B0_%2525252525252525252525252525D0%2525252525252525252525252525BF%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B3%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525BC%2525252525252525252525252525D0%2525252525252525252525252525BC%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B3%2525252525252525252525252525D0%2525252525252525252525252525BE_%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B1%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252581%2525252525252525252525252525D0%2525252525252525252525252525BF%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252587%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D1%25252525252525252525252525258F), которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.  Разработка через тестирование требует от разработчика создания автоматизированных [модульных тестов](https://ru.wikipedia.org/wiki/%2525252525252525252525252525D0%25252525252525252525252525259C%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B4%2525252525252525252525252525D1%252525252525252525252525252583%2525252525252525252525252525D0%2525252525252525252525252525BB%2525252525252525252525252525D1%25252525252525252525252525258C%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B5_%2525252525252525252525252525D1%252525252525252525252525252582%2525252525252525252525252525D0%2525252525252525252525252525B5%2525252525252525252525252525D1%252525252525252525252525252581%2525252525252525252525252525D1%252525252525252525252525252582%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D1%252525252525252525252525252580%2525252525252525252525252525D0%2525252525252525252525252525BE%2525252525252525252525252525D0%2525252525252525252525252525B2%2525252525252525252525252525D0%2525252525252525252525252525B0%2525252525252525252525252525D0%2525252525252525252525252525BD%2525252525252525252525252525D0%2525252525252525252525252525B8%2525252525252525252525252525D0%2525252525252525252525252525B5), определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, либо нет. Когда они выполняются, говорят, что тест пройден.

Тестирование в рамках TDD позволяет получить простой способ оценки полноты интерфейсов: необходимым и достаточным считается такой интерфейс, который позволяет выполнить все написанные тесты. Все, что находится за этими рамками, считается ненужным. Наконец, использование тестов в качестве инструмента дизайна заставляет программиста в первую очередь концентрироваться на интерфейсе, а уже во вторую - на имплементации, что также положительно влияет на результат.

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **6.1** Установка системы на удаленном сервере

## **6.2** Описание основных функций программного средства

# **7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

## **7.1** Характеристика программного продукта

## **7.2** Расчет сметы затрат, цены и прибыли

## **7.3** Расчет экономического эффекта у пользователя

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Прогнозы: когда мы полностью откажемся от наличных, а банковские карточки заменят смартфоны – [Электронный ресурс]. – Режим доступа: <https://finance.tut.by/news514441.html>
2. Telegram (мессенджер) [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Telegram\_(мессенджер)](https://ru.wikipedia.org/wiki/Telegram_(%252525252525252525252525D0%252525252525252525252525BC%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D1%25252525252525252525252581%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D0%252525252525252525252525BD%252525252525252525252525D0%252525252525252525252525B4%252525252525252525252525D0%252525252525252525252525B6%252525252525252525252525D0%252525252525252525252525B5%252525252525252525252525D1%25252525252525252525252580)).
3. Роботы [Электронный ресурс]. – Режим доступа: <https://tlgrm.ru/docs/bots>
4. Документация Telegram API [Электронный ресурс]. – Режим доступа: <https://tlgrm.ru/docs/bots/api>
5. Длинные опросы (long polling) [Электронный ресурс]. – Режим доступа: <http://javascript.ru/ajax/comet/long-poll>
6. TelegramBots: Java library to create bots using Telegram Bot API –[Электронный ресурс]. – Режим доступа: <https://github.com/rubenlagus/TelegramBots>
7. Конкуренты и аналоги JIRA | Atlassian – [Электронный ресурс]. – Режим доступа: [https://ru.atlassian.com/software/jira/comparison#!jira-ibm-rational-clearquest](https://ru.atlassian.com/software/jira/comparison%252525252523!jira-ibm-rational-clearquest)
8. JIRA REST API Reference – [Электронный ресурс]. – Режим доступа: <https://docs.atlassian.com/jira/REST/cloud/>
9. JIRA REST Java Client Library – [Электронный ресурс]. – Режим доступа: <https://ecosystem.atlassian.net/wiki/spaces/JRJC>
10. A Simple JIRA REST Client for Java – [Электронный ресурс]. – Режим доступа: <https://github.com/rcarz/jira-client>
11. JIRA REST Client Library for Java – [Электронный ресурс]. – Режим доступа: <https://github.com/lesstif/jira-rest-client>
12. TeamCity – Википедия – [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/TeamCity>
13. REST API – TeamCity 10.x Documentation – [Электронный ресурс]. – Режим доступа: <https://confluence.jetbrains.com/display/TCD10/REST+API>
14. Jersey – [Электронный ресурс]. – Режим доступа: <https://jersey.java.net>
15. SQLite, MySQL и PostgreSQL: сравниваем наиболее популярные реляционные СУБД – [Электронный ресурс]. – Режим доступа: <https://tproger.ru/translations/sqlite-mysql-postgresql-comparison>

# ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код некоторых алгоритмов программного модуля

# ПРИЛОЖЕНИЕ Б

(обязательное)

# ПРИЛОЖЕНИЕ В

(обязательное)

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов