СОДЕРЖАНИЕ

ВВЕДЕНИЕ....6
1 ОБЗОР ЛИТЕРАТУРЫ....8
1.1 Обзор аналогов....8
1.2 Взаимодействие с Telegram....9
1.3 Взаимодействие с JIRA....14
1.4 Взаимодействие с TeamCity....17
1.5 Выбор СУБД....20
1.6 Выводы....23
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ....24
2.1 Структура программного средства....24
2.2 Проектирование схемы данных....29
2.3 Инфраструктура программного средства....31
2.4 Выводы....32
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ....33
3.1 Модуль взаимодействия с Telegram....33
3.2 Модуль аутентификации....35
3.3 Модуль обработки команд из чата....36
3.4 Модуль интеграции с TeamCity....39
3.5 Модуль интеграции с JIRA....41
3.6 Модуль отправки уведомлений....42
3.7 Модуль доступа к данным....43
3.8 Модуль взаимодействия с пользователем....48
3.9 Выводы....49
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ....50
4.1 Запуск сборки проекта в TeamCity....50
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ....51
5.1 Функциональное тестирование....51
5.2 Модульное тестирование....53
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ....57
7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ....58
7.1 Характеристика программного средства....58
7.2 Расчет затрат на разработку программного средства....58
7.3 Расчет экономического эффекта от продажи программного продукта....62
7.4 Расчет показателей эффективности разработки программного продукта....64
ЗАКЛЮЧЕНИЕ....67
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....68
ПРИЛОЖЕНИЕ А....69
ПРИЛОЖЕНИЕ Б....70

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, так и для крупных предприятий.

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 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 с Telegram

Telegram — бесплатный кроссплатформенный мессенджер 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 ко-

личество 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 взаимодействия пользователя с чат-ботом может быть использован простой текст. Помимо простого текста в Telegram Bot API дополнительно реализованы более формальные инструменты, а именно команды, клавиатуры и встроенные клавиатуры. Рассмотрим подробнее эти инструменты ниже.

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. 1.1) и подсветку команд в тексте сообщений.

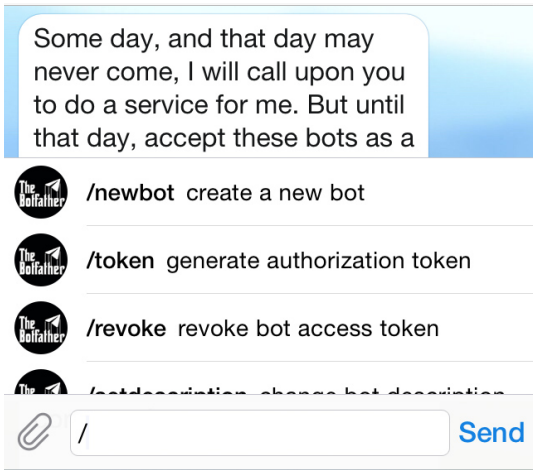


Рисунок 1.1 — Снимок экрана со списком поддерживаемых ботом команд

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. 1.2). Нажатие любой из клавиш сразу же отправит боту соответствующую команду. Таким образом значительно упрощается взаимодействие пользователя с ботом. Поддерживается простой текст и смайлы.

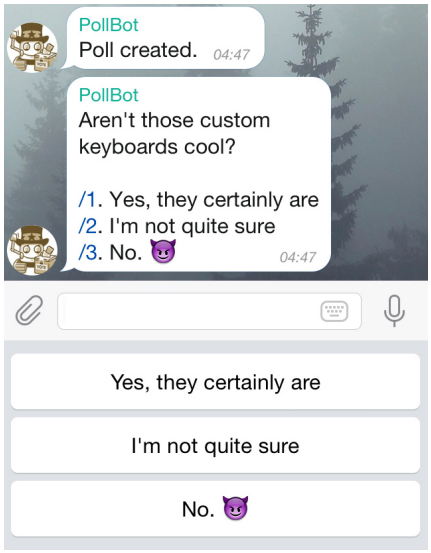


Рисунок 1.2 — Снимок экрана с набором клавиш

Некоторые пользователи могут предпочитать делать что-нибудь, не отправляя никаких сообщений в чат. В таких случаях могут использоваться встроенные клавиатуры (см. рис. 1.3), 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.

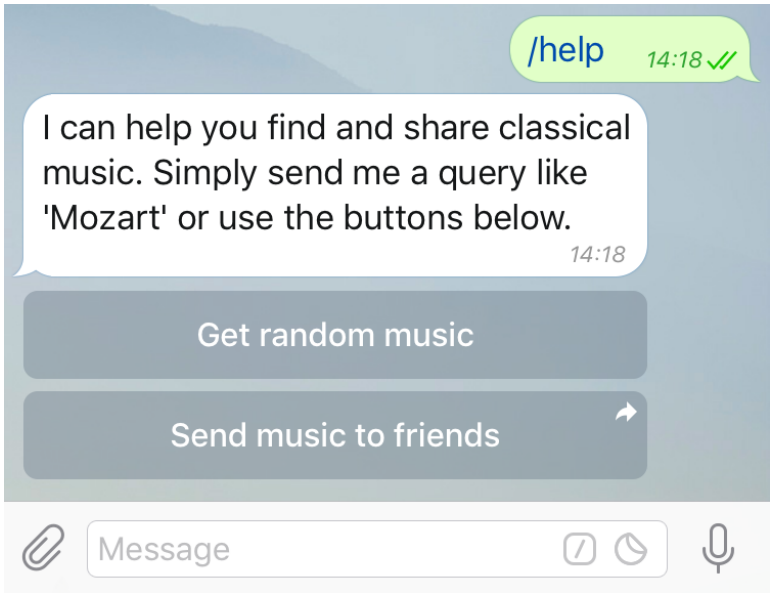


Рисунок 1.3 — Снимок экрана со встроенной клавиатурой

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. 1.4) [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.

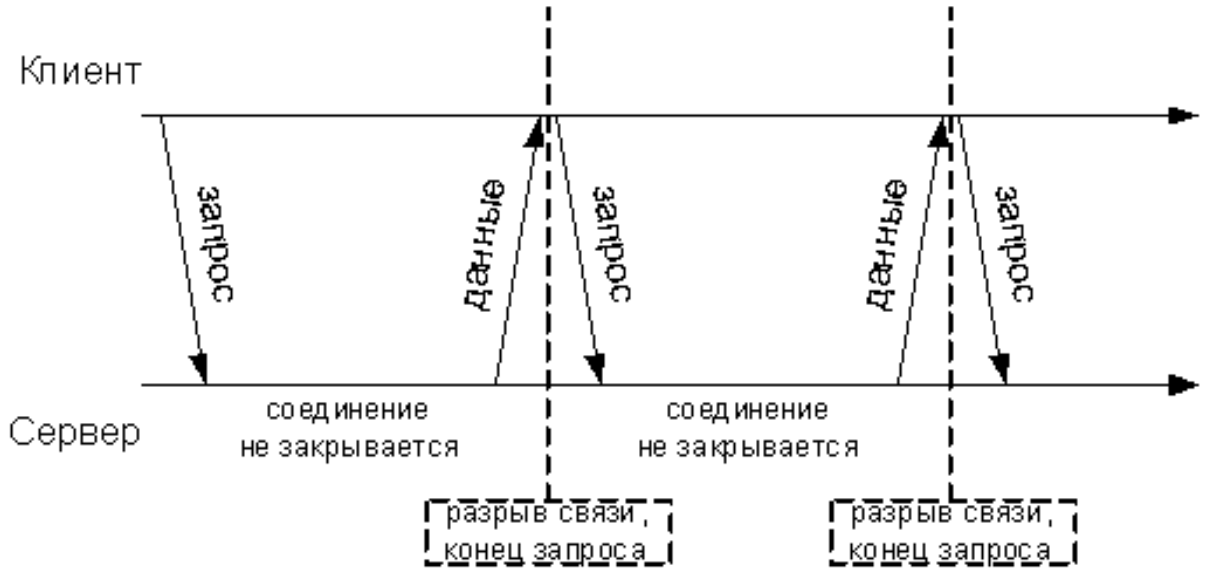


Рисунок 1.4 — Схема коммуникации сервера и клиента способом длинных опросов

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 Jet-Brains, 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 определяет 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 сборщика 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 config = 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 реляционные СУБД (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 реляционная СУБД, 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, так как 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-сов-

местимости, 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 СИСТЕМНОЕ 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. Диаграмма вариантов использования программного средства представлена на рисунке 2.1. На диаграмме представлены следующие действующие лица:

— руководитель проекта, член команды (разработчик/тестировщик) являются конечными пользователями программного средства;

— Telegram, является связующим звеном между конечным пользователем и программным средством, предоставляет удобный для пользователя интерфейс взаимодействия;

— программное средство, взаимодействует с Telegram, TeamCity и JIRA, обрабатывает команды конечного пользователя;

— TeamCity, предоставляет информацию о проектах и их конфигурациях, производит сборку проекта;

— JIRA, предоставляет информацию о проектах и задачах.

На диаграмме также представлены следующие функции:

— взаимодействие с TeamCity (получение списка конфигураций, запуск сборки проекта);

— взаимодействие с JIRA (получение списка задач, изменение задач);

— возможность получения команд от пользователей;

— возможность доставки уведомлений пользователю о каком-либо событии.

Помимо выделенных на диаграмме, немаловажными являются следующие функции:

— взаимодействие с пользователем понятными ему способами;

— обработка команд пользователя;

— возможность сохранения пользовательских данных;

— возможность аутентификации в TeamCity, JIRA.

2.1 Структура программного средства

Структурная схема представлена в приложении Б. Как видно из структурной схемы, программное средство взаимодействует с рядом внешних систем. Эти системы представлены на схеме модулями Telegram, TeamCity и JIRA. Рассмотрим каждый модуль подробнее.

Telegram — система непрерывного обмена сообщениями. Посредством чата будет обеспечиваться взаимодействие с пользователем. Текст и команды,

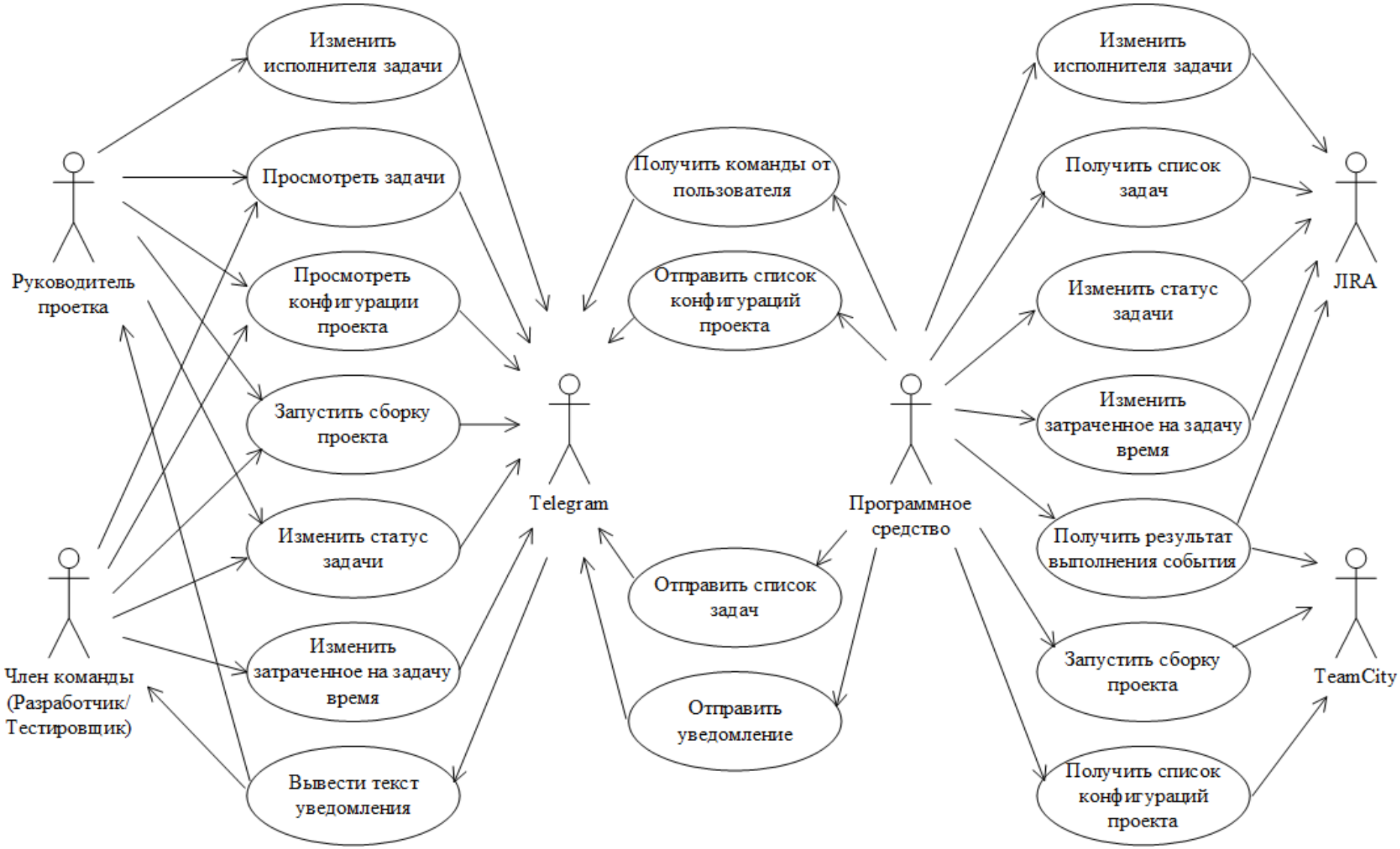


Рисунок 2.1 — Диаграмма вариантов использования

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.

Atlassian JIRA — система отслеживания задач. Отвечает за учет задач, ошибок и связанной с ними дополнительной информации. Данная внешняя система будет использована программным средством для закрытия задач как командой от пользователя, так и неявно по результатам сборки проекта системой непрерывной интеграции, а также для изменения статуса, исполнителя, затраченного времени задач и для получения уведомлений об изменении задач извне. С помощью чата пользователь сможет получать информацию о задачах, закрывать задачи, изменять статус, исполнителя и затраченное время задач. Уведомления также пользователь будет получать через чат.

TeamCity — система непрерывной интеграции. Отвечает за сборку проектов, тестирование, выявление ошибок. Программное средство с помощью TeamCity будет получать информацию о конфигурациях, запускать сборку проекта, получать и обрабатывать результаты сборки и уведомлять пользователя в чате об окончании сборки сообщением с необходимой информацией. Пользователь сможет с помощью чата запускать сборку проекта, получать информацию о существующих конфигурациях проекта, а также получать уведомления.

На структурной схеме также представлено непосредственно программное средство (см. рис. 2.2), 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.

Модуль взаимодействия с Telegram будет реализовывать логику работы с Telegram. Модуль будет отвечать за получение и отправку сообщений, а также за необходимый для корректной работы чат-бота функционал. Взаимодействие с Telegram будет осуществлено посредством Telegram Bot API. Для программной реализации работы с мессенджером будет использоваться библиотека TelegramBots.

В модуле интеграции с TeamCity будет реализована логика работы с TeamCity. В качестве основных задач модуля планируется реализовать 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. Для программной реализации работы с TeamCity REST API будет использоваться фреймворк Jersey.

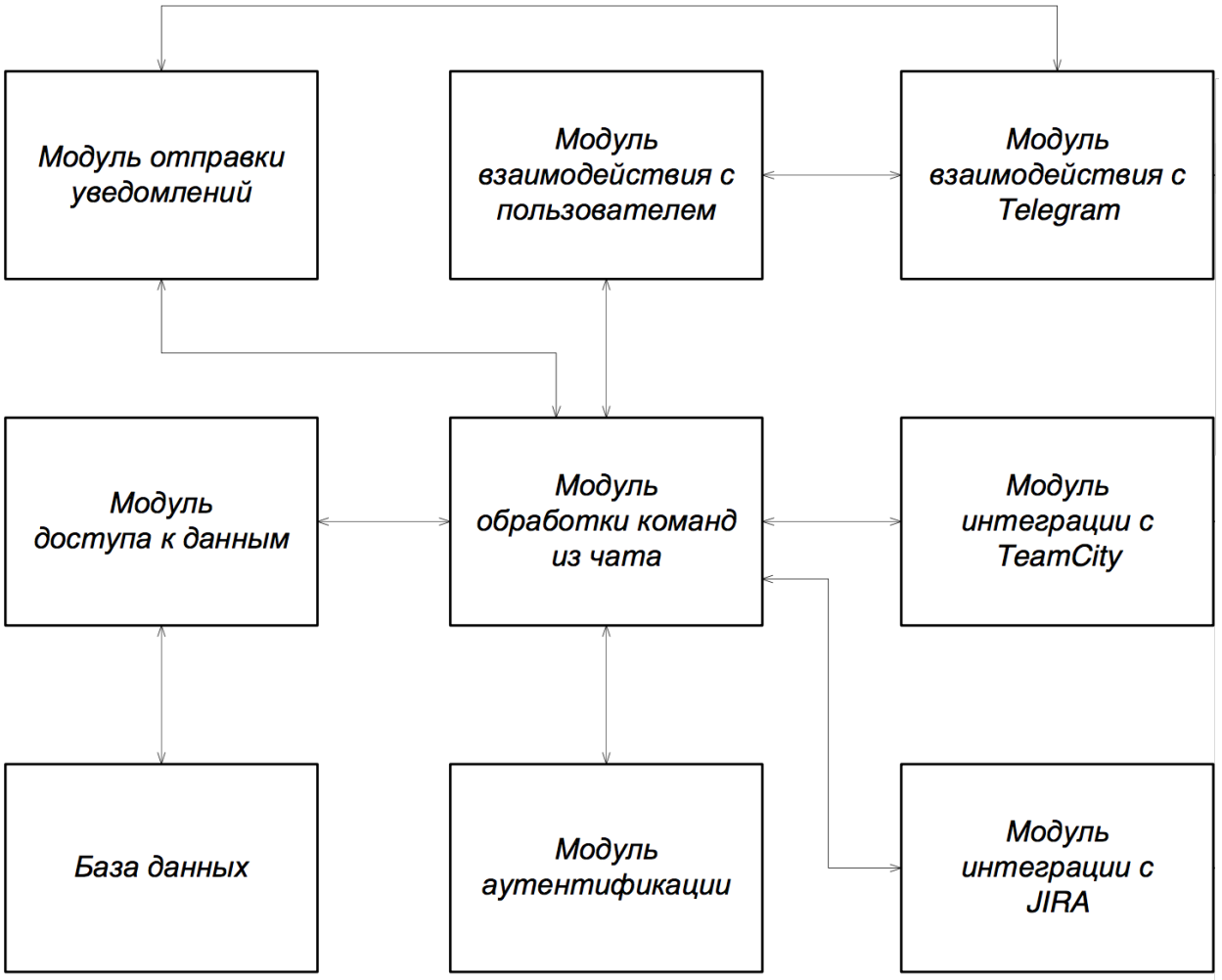


Рисунок 2.2 — Структурная схема программного средства

Модуль интеграции с JIRA будет реализовывать логику работы с JIRA. Основной функционал будет представлен работой 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. Для программной реализации с JIRA REST API будет использоваться библиотека rcarz/jira-client.

Модуль взаимодействия с пользователем планируется реализовать с помощью Telegram Bot API. Для взаимодействия пользователя с чат-ботом может быть использован использоваться простой текст. Помимо простого текста могут быть использованы именно команды, клавиатуры и встроенные клавиатуры. Таким образом, сделан вывод, что модуль взаимодействия с пользователем будет отвечать как за формирование сообщений-ответов поль-

зователю, так и за предоставление отличного от отправки простых текстовых сообщений метода взаимодействия.

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 производить 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-client и фреймворк Jersey соответственно.

Модуль обработки команд из чата. Данный модуль будет обрабатывать полученные от пользователя через модуль взаимодействия с Telegram сообщения. Модуль будет выделять их всех сообщений те, которые предназначены для бота и сторонние сообщения. Сторонние сообщение будут игнорироваться. Команда может быть передана простым текстом или же специальной кнопкой. Когда команда представлена текстом, по ключевым словам определяется, на какую внешнюю систему направлена команда. По нажатии на команду-кнопку, как и после обработки простого текста команда передается конкретному модулю, отвечающему за работу с определенной внешней системой. В случае неоднозначного трактования или некорректности полученной команды отправляется сообщение, уведомляющее пользователя об этом.

Модуль доступа к данным будет реализовывать логику работы с базой данных, а именно: позволять обращаться к таблицам в базе данных и выполнять с ними различные операции: чтение и поиск, вставку и удаление объектов, а также вызов хранимых процедур. Работа с базой данных будет производится с использованием технологии *Object-Relational Mapping (ORM)*, которая связывает реляционные базы данных с концепциями объектно-ориентированного программирования. В качестве ORM фреймворка будет использоваться библиотека для языка программирования Java Hibernate. Библиотека не только решает задачу связи классов Java с таблицами базы данных и типов данных Java с типами данных SQL, но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора

данных и преобразования объектов, максимально облегчая перенос приложения на любые базы данных SQL.

База данных представляет собой хранилище для данных, полученных от пользователя для аутентификации во внешних системах, информации о существующих проектах и задачах в JIRA, проектах и их конфигураций в TeamCity, а также результатов работы программного средства. Будет использоваться СУБД PostgreSQL.

2.2 Проектирование схемы данных

Для хранения данных, используемых программным средством, и данных, полученных в ходе работы, будет использована реляционная система управления базами данных PostgreSQL. Реляционные системы хранят информацию как набор связанных записей и атрибутов в таблице. Каждая запись обладает уникальным ключом.

Схема данных программного средства представлена в приложении В. Далее будет рассмотрена структура записей в каждой таблице, наиболее важные поля, связи с другими таблицами и цели хранения соответствующей таблице информации.

В таблице application_project будет хранится информация о проектах, разрабатываемых компанией. Как правило, в компании разрабатываются одновременно несколько проектов. Также один проект может иметь несколько версий: стабильная, находящаяся в разработке, трестируемая, предыдущая. В данной таблице будет хранится общая информация о каком-либо конкретном проекте. Поле application_project_id — это первичный ключ. Поле name хранит название проекта, а descripton — краткое описание. Поле telegram_lead_id является внешним ключом и ссылается на таблицу telegram_lead, тем самым указывая на руководителя проекта.

Таблица telegram_lead будет хранить информацию о руководителе какого-либо проекта. В рамках данного программного средства, руководитель проекта будет иметь возможность добавлять новый чат к существующему проекту. Для этого программному средству следует знать, как выделить руководителя среди обычных пользователей. Поэтому каждая запись в таблице будет содержать поле telegram_id, которое будет хранить идентификатор руководителя как пользователя Telegram, и поле name, которое будет хранить имя руководителя. Первичный ключ — поле telegram_lead_id,

Таблица telegram_group хранит информацию о групповых чатах, связанных с определенным проектом с помощью внешнего ключа application_project_id. Поле telegram_chat_group_id хранит идентификатор чата в Telegram, а поле name — название чата. Первичным ключом является поле telegram_group_ud. Потребность в хранении подобной информации заключена в том, что в разработке проекта участвуют многие специалисты, например разработчики и тестировщики. Поэтому каждому роду

специалистов, работающих над одним проектом, целесообразно иметь свой собственный чат, для обсуждения свойственной данному роду специалистов информации.

В таблице credentials будет хранится информация, необходимая программному средству для аутентификации во внешней системе для возможности дальнейшего работы с этой системой. Поле credentials_id является первичным ключом. Для аутентификации в таблице присутствуют поля url, для хранения пути, по которому можно получить доступ к внешней системе, а также login и password.

В таблице teamcity_project будет хранится информации о существующих в TeamCity проектах. Как было определено ранее, компания может заниматься проектом, который может иметь разрабатываемую и поддерживаемую или предыдущую версии. Каждая из этих версий может требовать непрерывной интеграции. Поэтому записи в таблице будут содержать следующие поля:

— teamcity_project_id, которое является первичным ключом;

— teamcity_id, для хранения идентификатора проекта в TeamCity;

— name, для хранения названия проекта;

— description, для хранения описания проекта;

— application_project_id, внешний ключ, который указывает на принадлежность данного TeamCity проекта к проекту компании;

— credetnials_id, внешний ключ, который ссылается на информацию, необходимую для аутентификации в TeamCity;

— last_update, для хранения даты и времени последнего обновления данных таблицы с TeamCity сервера.

Таблица teamcity_project_configuration спроектирована для хранения информации о существующих конфигурациях определенного проекта в TeamCity. Помимо того, что в компании может быть несколько версий проекта, каждая версия также может иметь несколько конфигураций. Количество конфигураций может зависеть от количества ветвей, в которых вносятся изменения в проект. Первичным ключом в таблице является поле teamcity_project_configuration_id. Поле teamcity_id хранит идентификатор конфигурации проекта в системе TeamCity, а поле name — название конфигурации. Внешний ключ teamcity_project_id ссылается на TeamCity проект, к которому относится каждая конфигурация.

В таблице teamcity_last_build будет хранится информация о последней сборке какой-либо конфигурации какого-либо TeamCity проекта. Первичный ключ teamcity_project_configuration_id является и внешним ключом, связывающий последнюю сборку проекта с соответствующей конфигурацией. Таким образом связь этой таблицей с таблицей teamcity_project_configuration будет один к одному. Также в данной таблице присутствуют поле teamcity_id, которое является идентификатором сборки в TeamCity, поле message, которое хранит сообщение для пользователей о результатах сборки, поле notified, которое указывает, были ли уведомлены пользователи об

окончании сборки и её результатах. Поле teamcity_build_status_id ссылается на таблицу teamcity_build_status, тем самым указывая на один из возможных статусов, который может иметь сборка, например, успешно завершенный и завершенный с ошибками.

Таблица teamcity_build_status спроектирована для хранения статусов, один из которых имеет каждая сборка проекта в TeamCity. Поле teamcity_build_status_id является первичным ключом, а поле status хранит строковое обозначение статуса сборки.

Таблица jira_project будет хранить информации о существующих в JIRA проектах. Записи в таблице будут содержать следующие поля:

— jira_project_id, которое является первичным ключом;

— jira_id, для хранения идентификатора проекта в JIRA;

— name, для хранения названия проекта;

— application_project_id, внешний ключ, который указывает на принадлежность данного JIRA проекта к проекту компании;

— credetnials_id, внешний ключ, который ссылается на информацию, необходимую для аутентификации в JIRA;

— last_update, для хранения даты и времени последнего обновления данных таблицы с JIRA сервера.

2.3 Инфраструктура программного средства

Диаграмма развертывания программного средства представлена на рисунке 2.3. На диаграмме показаны устройства, которые будут вовлечены в процесс работы приложения. Также на диаграмме показаны компоненты, необходимые для корректной работы программного средства.

Для взаимодействия с программным средством на устройствах работников, будь то персональный компьютер, или же мобильное устройство, должна быть установлена система мгновенного обмена сообщениями Telegram.

Так как программное средство планируется разрабатывать на языке Java, то для его работы необходимым является установка Java SE Runtime Environment на той машине, на которой в дальнейшем данное программное средство будет функционировать.

Как было определено ранее, для хранения данных будет использована реляционная СУБД PostgreSQL. Для этого необходимо установить СУБД на сервере программного средства.

Так как системы TeamCity и JIRA также разработаны на языке программирования Java и являются промышленными системами , то для их корректной работы необходима установка Java EE Runtime Environment на соответственно сервере TeamCity и сервере JIRA. Помимо Java EE Runtime Environment, на машинах должны быть установлены и сами системы.

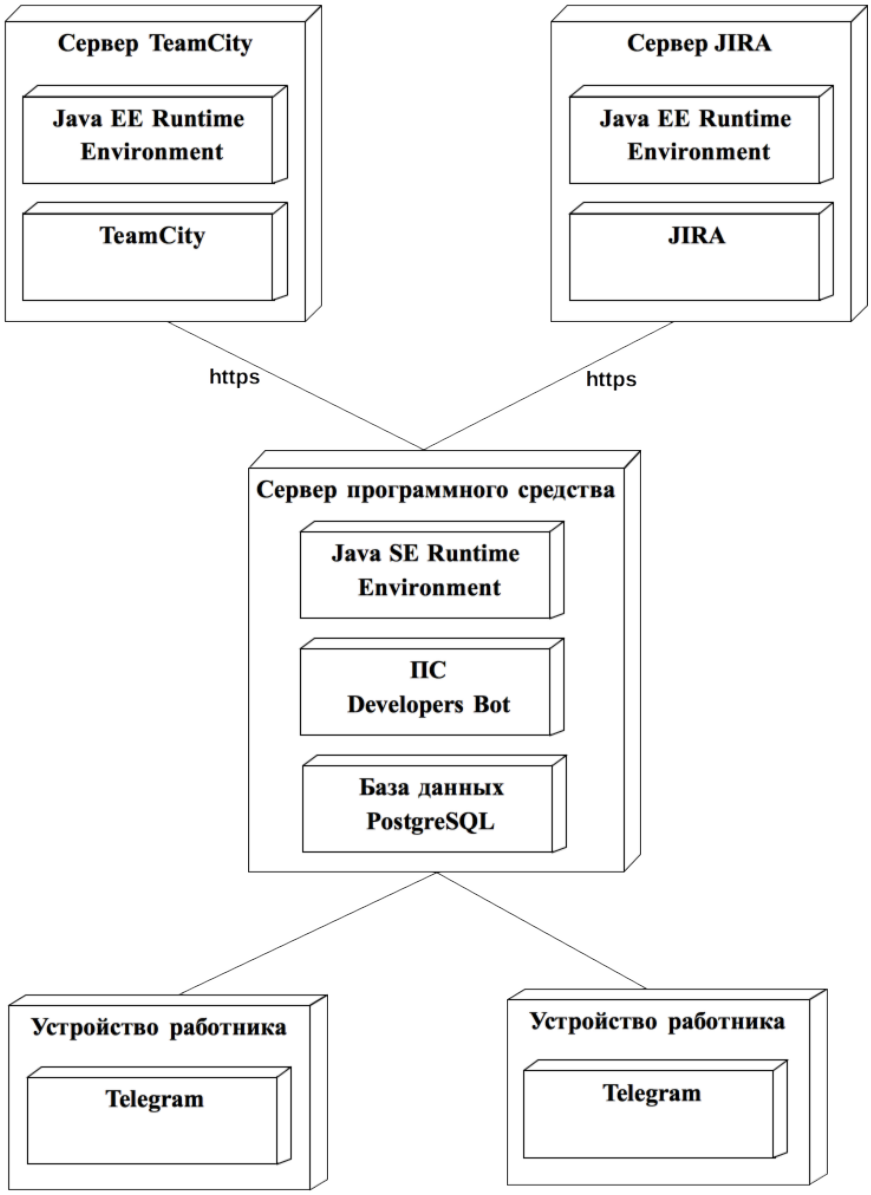


Рисунок 2.3 — Диаграмма развертывания программного средства

2.4 Выводы

В данной главе была проведена декомпозиции программного средства на структурные модули исходя из планируемых возможностей программного средства, были определены назначения и задачи каждого модуля, связи модулей между собой. Также была спроектирована схема данных, определены цели хранения информации в таблицах, основные поля таблиц и связи таблиц между собой. Была спроектирована инфраструктура программного средства, выделены компоненты, которые необходимо будет установить на каждом устройстве для корректной работы разрабатываемого программного средства.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Одним из наиболее значимых этапов разработки программного средства является функциональное проектирование, а именно описание целей и структуры классов и взаимодействия между ними. Для описания структуры классов в данной главе будут перечислены основные классы программного модуля с указанием способов их взаимодействия между собой, методов и полей, используя диаграмму классов (см. приложение В) в соответствии с модулями, выделенными на этапе системного проектирования (см. приложение Б).

— 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;

3.1 Модуль взаимодействия с Telegram

Данный модуль будет реализовывать логику работы с Telegram. Модуль будет отвечать за получение обновлений из чата и отправку сообщений. Взаимодействие с Telegram будет осуществлено посредством Telegram Bot API. Для реализации работы с мессенджером будет использована библиотека TelegramBots.

Диаграмма классов этого модуля представлена на рисунке 3.1

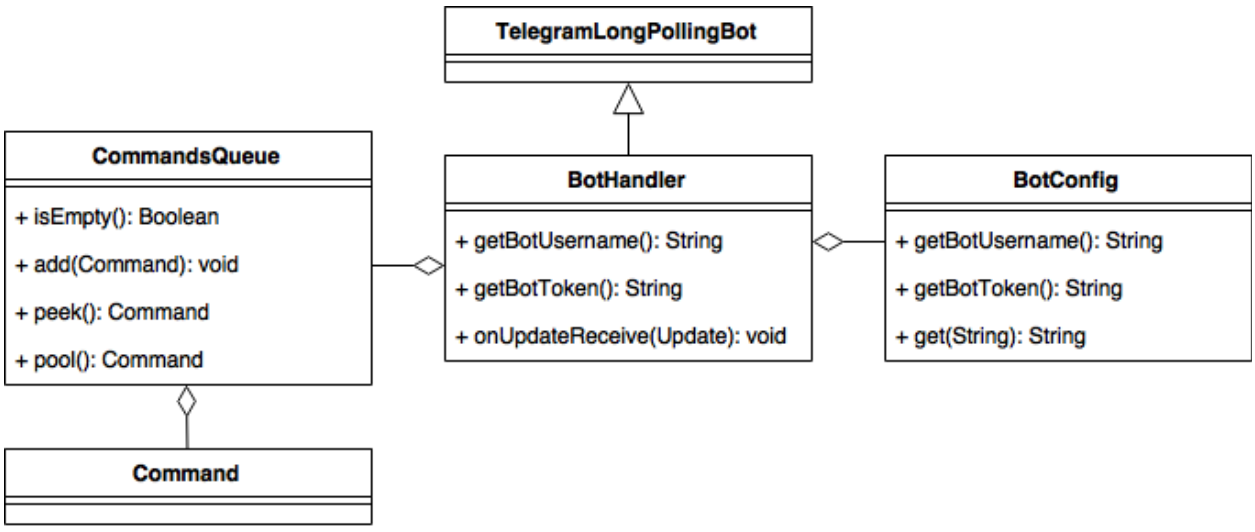


Рисунок 3.1 — Диаграмма классов модуля взаимодействия с Telegram

Основным классом данного модуля является класс `BotHandler`. Данный класс является наследником класса `TelegramLongPollingBot`,

предоставляемого библиотекой TelegramBots. Как видно из названия класса, взаимодействие с Telegram будет реализовано на основе длинных опросов. Класс-наследник должен реализовывать следующие методы:

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

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

3) `public void onUpdateReceived(Update update)`. Этот метод будет вызываться всякий раз при получении обновления `Update`. Объект `Update` хранит полученное ботом через чат обновление, будь-то текстовое сообщение, документ или фотография.

Отправка сообщений реализуется методом `sendMessage(SendMessage message)`. Класс `SendMessage` предоставляется библиотекой TelegramBots для формирования сообщения, а метод sendMessage используется для отправки ботом сообщений в чат.

Конфигурационные данные для корректной работы бота, такие как имя бота и токен, предоставленные Telegram, будут хранится в конфигурационном файле config.bot.properties в формате, представленном в листинге 3.1.

```
config.bot.name=<BOT_NAME>
config.bot.token=<TOKEN>
config.lead.id=<LEAD_TELEGRAM_ID>
config.lead.name=<LEAD_NAME>
config.project.name=<PROJECT_NAME>
```

Листинг 3.1 — Пример формата записей в конфигурационном файле

Для считывания данных из файла и их разбора будет использоваться класс `BotConfig`. Класс `BotConfig` будет загружать данные из файла и предоставлять к ним доступ с помощью методов `getBotUsername`, `getBotToken`, так же можно использовать метод `get(String property)`. Этот метод принимает строку с названием свойства, которое публично определено в классе `BotConfig`, и возвращает значение свойства в виде строки.

Для того, чтобы пользовательские команды не были потеряны, в случае запроса на выполнения какой-либо команды во время выполнения другой команды, будет использоваться класс `CommandsQueue`. В данном классе реализованы следующие методы: `isEmpty` (метод возвращает true, если в очереди на выполнение нет команд, или false, если есть команды, которые еще не выполнены), `add` (метод принимает экземпляр класса `Command` и добавляет его в очередь на выполнение), `peek` (метод возвращает первую команду в очереди, не удаляя эту команду из очереди) и метод `poll` (возвращает команду из начала очереди, при этому удаляя ее из очереди).

3.2 Модуль аутентификации

Модуль аутентификации необходим для аутентификации пользователей во внешних системах, для того чтобы программное средство имело возможность получать от этих систем данные и производить с этой системой взаимодействие. Аутентификация производится по логину и паролю, предоставленными пользователем. Для аутентификации пользователь также должен предоставить URI сервера, на котором установлена система. Данная информация будет предоставлена пользователем через чат Telegram. Для аутентификации в TeamCity и JIRA будет использоваться REST API, метод Basic HTTP.

Диаграмма классов данного модуля представлена на рисунке 3.2.

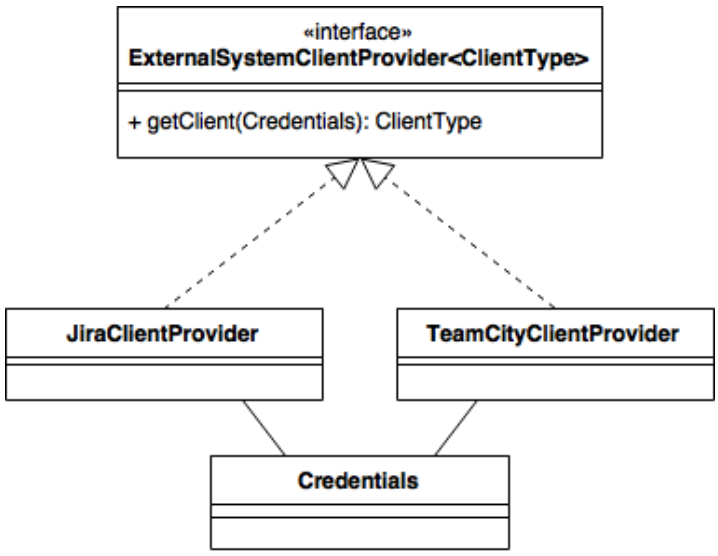


Рисунок 3.2 — Диаграмма классов модуля аутентификации

Модуль представлен интерфейсом `ExternalSystemClientProvider` и двумя классами, которые его реализуют, `JiraClientProvider` и `TeamClient-Provider`.

Как видно из диаграммы, интерфейс `ExternalSystemClientProvider` имеет один метод — `getClient`, возвращаемый тип которого является параметризованным. Этот метод принимает объект класса `Credentials`, устанавливает соединение с внешней системой и возвращает объект класса клиента, с помощью которого будет производится дальнейшее взаимодействие с системой. Классы-наследники `JiraClientProvider` и `TeamClientProvider` реализуют этот метод в соотвествии с тем, с какой системой они должны устанавливать соединение.

Для аутентификации будут использоваться объекты класса `Creden-tials`, которые хранят в себе учетные данные для входа, а именно URL, логин и пароль, а также имеют методы для получения и установки этих данных.

3.3 Модуль обработки команд из чата

Данный модуль будет обрабатывать полученные от пользователя через модуль взаимодействия с Telegram сообщения. Модуль будет выделять из всех сообщений те, которые предназначены для бота и сторонние сообщения.

В данный модуль входят классы `CommandRecognizer` и `CommandFactory`, класс-перечисление `CommandName`, а также интерфейс `Command` (см. рисунок 3.3), и классы, реализующие этот интерфейс.

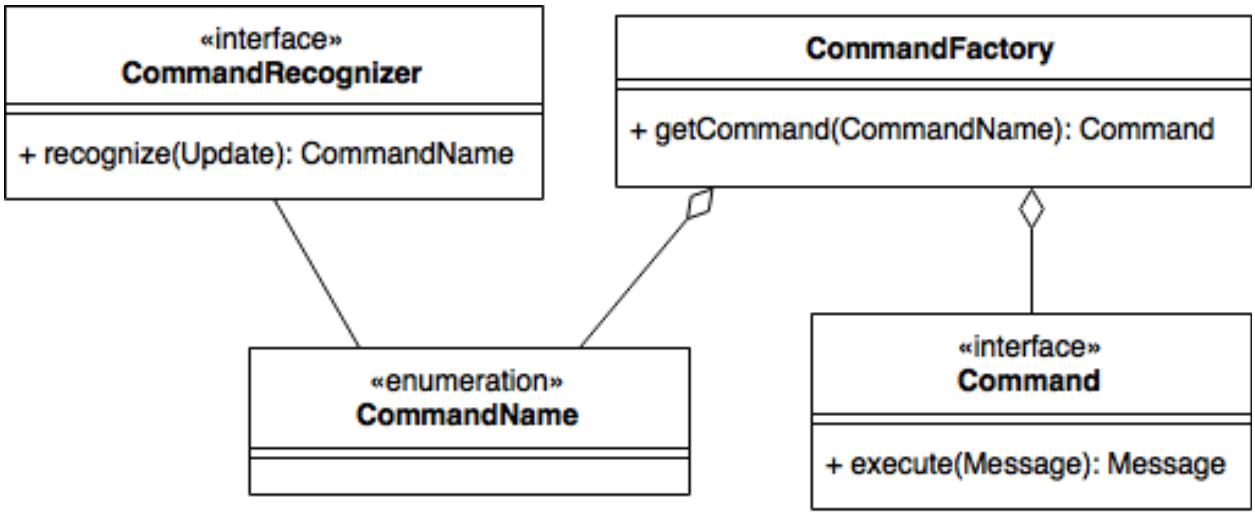


Рисунок 3.3 — Диаграмма классов модуля обработки команд

Класс-перечисление `CommandName` содержит перечисление всех реализованных и поддерживаемых программным средством команд, а также идентификатор неизвестной команды и идентификатор сообщения, которое не является командой. Примером членов данного класса-перечисления являются `NON_COMMAND`, `UNKNOWN_COMMAND`, `START`, `HELP`, `AUTHENTICATE_JIRA`, `AUTHENTICATE_TEAMCITY` и другие.

Класс `CommandHandler` формирует объекты команд и предоставляет интерфейс для получения объекта-команды по её имени. Данный класс содержит единственный открытый метод `Command getCommand(CommandName commandName)`. Данный метод принимает имя команды `CommandName` и возвращает объект-команды `Command`.

Интерфейс `Command` содержит единственный метод `Message execute(Message message)`, который принимает объект сообщения `Message`, который в свою очередь содержится в объекте обновления `Update`, и возвращает объект `Message`. Наследники интерфейса реализуют данный метод в соотвествии с поставленной перед ними задачей. Интерфейс `Command` составляет вместе со своими наследниками основу модуля обработки команд.

На рисунке 3.4 представлены основные классы-наследники интерфейса `Command`. Ниже будет рассмотрен каждый класс подробнее.

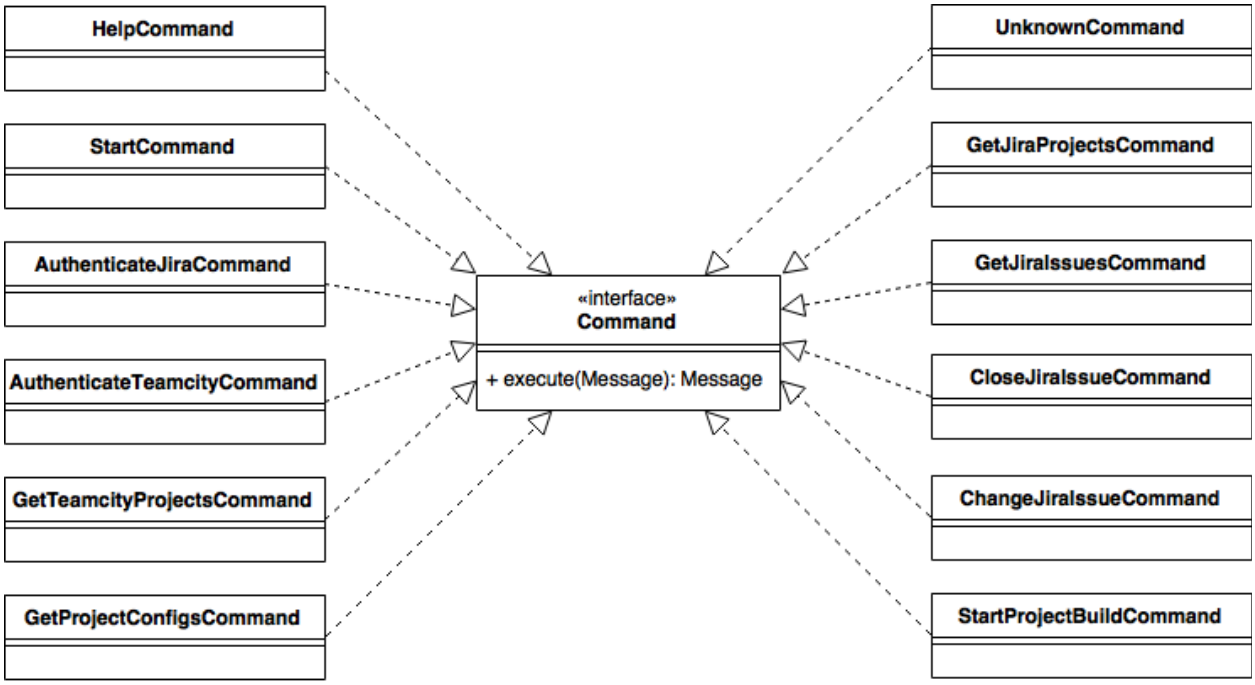


Рисунок 3.4 — Интерфейс Command и классы наследники

Класс `StartCommand` будет обрабатывать первое сообщение пользователя обращенное к боту. Данный класс будет подготавливать сообщение-ответ, записывая туда список команд, доступный у этого бота, а также будет предлагать пользователю пройти аутентификацию в JIRA и TeamCity.

Класс `HelpCommand` будет подготавливать сообщение, содержимым которого будет являться список доступных пользователю команд.

Класс `UnknownCommand` будет обрабатывать неизвестные боту команды. Результатом работы будет сообщение-предупреждение о возможном некорректном формировании команды в случае, если сообщение пришло из чата с одним пользователем, или бот будет просто игнорировать входящее сообщение в случае, если сообщение пришло из чата группы пользователей, так как сообщение может быть адресовано другому боту, который может находится в этой же группе.

Класс `AuthenticateJiraCommand` будет обрабатывать аутентификацию пользователя в JIRA. Прежде всего будет сформирован объект класса Credentials на основе полученных от пользователя через чат данных. Далее этот объект и дальнейшая обработка команды будет передана модулю аутентификации. По завершению обработки команды, в чат будет отправлено сообщение об успешности аутентификации, или, наоборот, о возникших в ходу аутентификации проблемах.

Класс `AuthenticateTeamcityCommand` будет работать аналогично классу `AuthenticateJiraCommand`, за исключением того, что будет выполнятся аутентификация в TeamCity.

Классы `GetTeamcityProjectsCommand`, `GetProjectConfigsCommand` и `StartProjectBuildCommand` будут взаимодействовать с модулем интеграции

с TeamCity и результатом их работы будут соответственно: формирование сообщения со списком проектов и дополнительной информации (идентификатор проекта в самой системе TeamCity, краткое описание) к каждому проекту; формирование сообщения со списком конфигураций каждого проекта в случае, если конкретный проект не указан, или, в ином случае, со списком конфигураций одного указанного проекта; запуск сборки проекта и формирование сообщения об успешности запуска, или о возникших проблемах, в случае неуспешного запуска.

Классы `GetJiraProjectsCommand`, `GetJiraIssuesCommand`, `CloseJiraIssueCommand` и `ChangeJiraIssueCommand` будут взаимодействовать с модулем интеграции с JIRA и результатом их работы будут соответственно: формирование сообщения со списком проектов и дополнительной информации (идентификатор проекта в самой системе JIRA, ключ проекта и краткое описание) к каждому проекту; формирование сообщения со списком задач каждого проекта в случае, если конкретный проект не указан, или, в ином случае, со списком задач одного указанного проекта; закрытие задачи в JIRA и формирование сообщения об изменении статуса задачи на «Done», или о возникших проблемах, в случае неуспешного изменения статуса задачи; изменение таких атрибутов задачи, как исполнитель, затраченное время, статус, а также формирование сообщения о результате изменения.

Классы, которые реализуют интерфейс `CommandRecognizer`, определяют на выполнение какой команды адресовано сообщение от пользователя. Основным методом интерфейса `CommandRecognizer` является `CommandName recognize(Update update)`. Данный метод принимает полученное от Telegram обновление `Update` и возвращает соответствующее данному обновлению имя команды `CommandName` (см. рисунок 3.5).

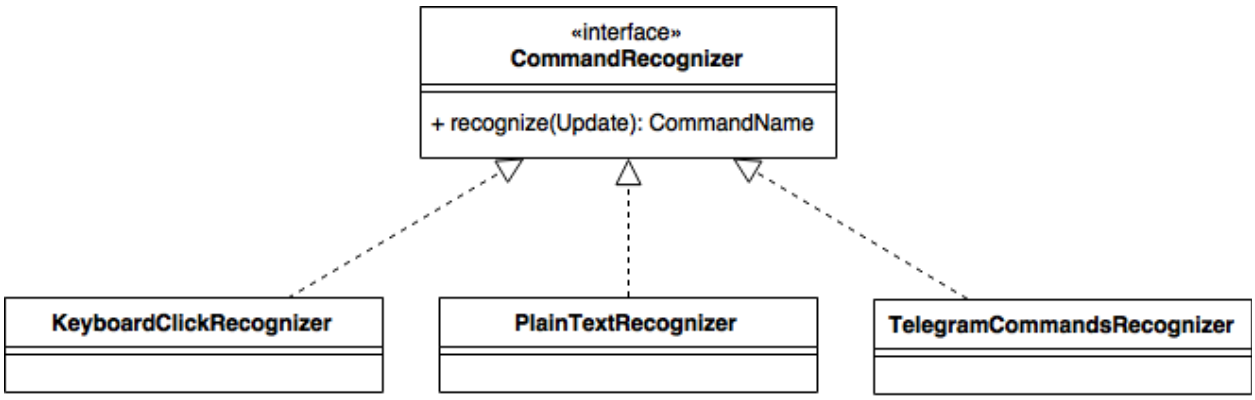


Рисунок 3.5 — Интерфейс CommandRecognizer и его реализации

Классы-наследники могут реализовывать методы интерфейса `CommandRecognizer` для определения команды на основе различных событий, пришедших от Telegram. Так, например, класс `PlainTextRecognizer` определяет команду на основе пришедшего простого текстового сообщения от пользова-

теля, класс `TelegramCommandsRecognizer` на основе текстового сообщения, включающего в себя Telegram команду, начинающуюся с символа «/», а также класс `KeyboardClickRecognizer` на основе обновления от Telegram, пришедшего в результате нажатия на какую-либо кнопку.

3.4 Модуль интеграции с TeamCity

Данный модуль будет реализовывать логику работы с TeamCity. TeamCity взаимодействует с внешними системами с помощью REST API. Для программной реализации работы с TeamCity REST API будет использоваться фреймворк Jersey.

В данном модуле можно выделить следующие классы: `TeamcityController`, `TeamCityUrlBuilder` (см. рисунок 3.6), а также классы-модели `Project`, `Build`, `BuildType`, `Change`.

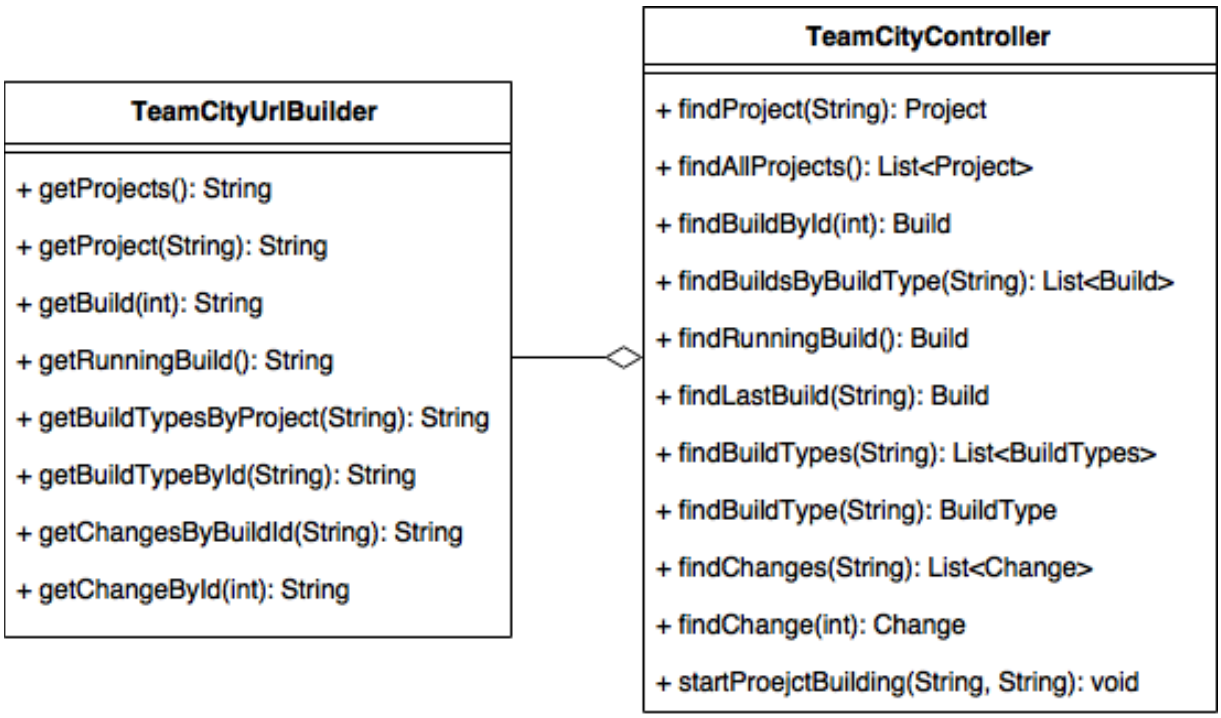


Рисунок 3.6 — Диаграмма классов модуля интеграции с TeamCity

Классы `Project`, `Build`, `BuildType` и `Change` являются классами-моделями, хранящими информацию, получаемую от TeamCity. Именно в эти классы будут преобразовываться получаемые от сервера xml-файлы. Каждый класс имеет свое поле соответствующее xml-элементу или xml-атрибуту своего xml-файла. Преобразование будет реализовано при помощи JAXB аннотаций, например `XmlRootElement`, `XmlElement`, `XmlAttribute` и др.

Класс `Project` будет хранить информацию о TeamCity проекте, например название проекта, идентификатор, описание, url-ссылку на проект, а также список созданных для этого проекта конфигураций.

Класс `BuildType` будет хранить информацию о конфигурации проекта, например название и идентификатор конфигурации, название и идентификатор проекта, url-ссылку.

Класс `Build` будет хранить информацию о сборке, например статус после завершения, время начала и окончания, продолжительность сборки, инициатора и список изменений.

Класс `Change` будет хранить список изменений, внесенных перед соответствующей сборкой.

Класс `TeamCityUrlBuilder` предназначен для формирования URL-строк REST API запросов. Используется классом `TeamCityController`. Класс `TeamCityUrlBuilder` имеет множество методов для получения строк, по которым в системе доступны различные сущности. Далее будут рассмотрены некоторые из этих методов. Метод `getProjects` возвращает строку, по которой можно получить список всех проектов в TeamCity. Метод `getProject` принимает строковый идентификатор проекта в системе и возвращает строку, по которой можно получить этот проект. Метод `getBuildTypesByProject` принимает идентификатор проекта в TeamCity и возвращает строку, по которой можно получить список конфигураций соответствующего проекта. Метод `getBuildTypeById` принимает идентификатор конфигурации и возвращает строку, по которой в системе доступна информация об этой конфигурации.

Класс `TeamCityController` является основным классом данного модуля. Данный класс предоставляет интерфейс для взаимодействия с TeamCity, а именно: получение каких-либо сущностей и запуск сборки проекта. Класс реализует основную логику для работы с TeamCity: формирование запроса и обращение к серверу, получение ответа и преобразование xml-ответов в объекты классов. Данный класс использует класс `TeamCityUrlBuilder` для получения строки-запроса для TeamCity REST API. Ниже будут рассмотрены некоторые методы данного класса.

— `startProjectBuilding`, принимает идентификаторы проекта и конфигурации, инициирует начало сборки;

— `findAllProjects`, возвращает список всех проектов в TeamCity;

— `findProject`, принимает идентификатор проекта и возвращает соответствующий проект;

— `findBuildTypes`, принимает идентификатор проекта и возвращает список созданных конфигурация для данного проекта;

— `findBuildById`, принимает идентификатор сборки и возвращает информацию о соответствующей сборке;

— `findChanges`, принимает идентификатор сборки и возвращает список предшествующих сборке изменений;

— `findRunningBuild`, возвращает информацию о выполняющейся на момент вызова данной функции сборки, если такова существует;

— `findLastBuild`, принимает идентификатор конфигурации и возвра-

щает информацию по последней завершенной сборки соответствующей конфигурации.

3.5 Модуль интеграции с JIRA

Данный модуль будет реализовывать логику работы с JIRA. JIRA взаимодействует с внешними системами с помощью REST API. Для программной реализации с JIRA REST API будет использоваться библиотека rcarz.

Основным классом данного модуля является `JiraController` (см. рисунок 3.7). Все классы-модели предоставляются библиотекой rcarz.

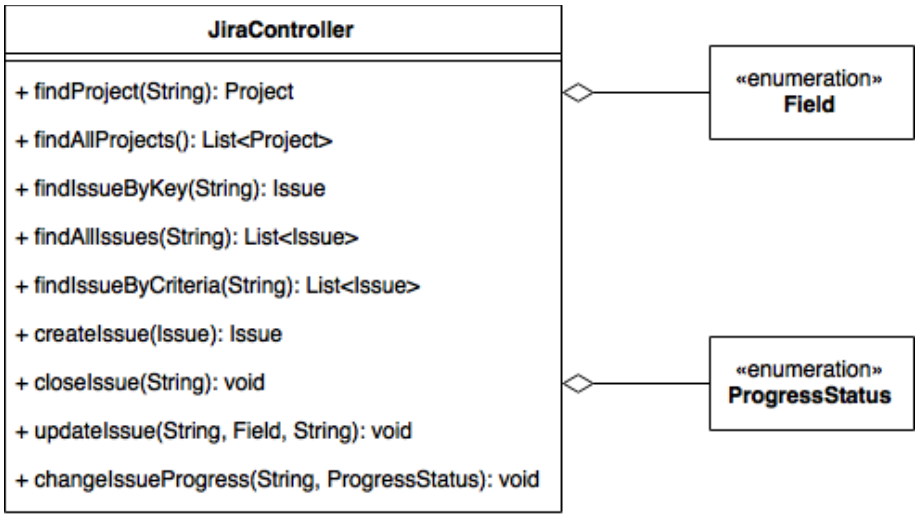


Рисунок 3.7 — Диаграмма классов модуля интеграции с JIRA

Класс `JiraController` предоставляет интерфейс для взаимодействия с JIRA. В основном взаимодействие направлено на работу с задачами: поиск по идентификатору или ключу, поиск по другим критериям, изменение различных полей, создание и закрытие. Также есть методы для получения проекта по идентификатору или списка всех проектов. Ниже будут рассмотрены методы подробнее.

— `findProject`, принимает идентификатор проекта и возвращает соответствующий проект;

— `findAllProjects`, возвращает список всех проектов в JIRA;

— `findIssueByKey`, принимает идентификатор задачи и возвращает соответствующую задачу;

— `findAllIssues`, принимает идентификатор проекта и возвращает все задачи, созданные для этого проекта;

— `findIssueByCriteria`, принимает критерии поиска в виде строки и возвращает все задачи, подходящие по данным критериям;

— `createIssue`, принимает новую задачу и возвращает эту же задачу,

созданную в системе JIRA;

— `closeIssue`, принимает идентификатор задачи и устанавливает в JIRA статус этой задачи как «закрыта»;

— `updateIssue`, принимает идентификатор задачи и экземпляр класса-перечисления `Field`, который сопоставляется с одним из полей задачи, и новое значение этого поля и изменяет значение этого поля в JIRA;

— `changeIssueProgress`, принимает идентификатор задачи и экземпляр класса-перечисления `ProgressStatus`, который сопоставляется с одним из возможных состояний прогресса задачи, и изменяет значение прогресса в JIRA;

Класс-перечисление `Field` хранит идентификаторы полей задачи, и используется для обновления задач. Класс-перечисление `ProgressStatus` хранит возможные статусы задачи, и также используется для обновления задач.

3.6 Модуль отправки уведомлений

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-client и фреймворк Jersey соответственно.

Непосредственным отправителем сообщений будут являться классы `JiraSubscriber`, и наследники класса `TeamcitySubscriber`: `Configurtion-ChangedSubscriber` и `BuildFinishedSubscriber` (см. рисунок 3.8).

Класс `JiraSubscriber` реализует интерфейс `Runnable`. Экземпляры класса будут обращаться через короткий интервал времени к внешним системам и, в случае завершения какого-либо события, отравлять уведомления в чат. Для получения экземпляра данного класса используется статический метод `newInstance`, который принимает экземпляр класса `Jira-Controller`, с помощью которого будет производится взаимодействие с системой JIRA и экземпляр класса `AbsSender`, предоставляемый библиотекой TelegramBots, и вызывать у этого объекта метод `sendMessage(SendMessage)` с подробной информацией о завершении события. Данный класс будет использоваться для уведомления пользователей об каких-либо изменениях задач в системе JIRA.

Абстрактный класс `TeamcitySubscriber` также реализует интерфейс `Runnable`. Является базовым классов для классов `BuildFinishedSubscriber` и `ConfigurationChangedSubscriber`. В этом классе будет реализована основная логика по работе с TeamCity, отправкой уведомлений же будут заниматься классы наследники.

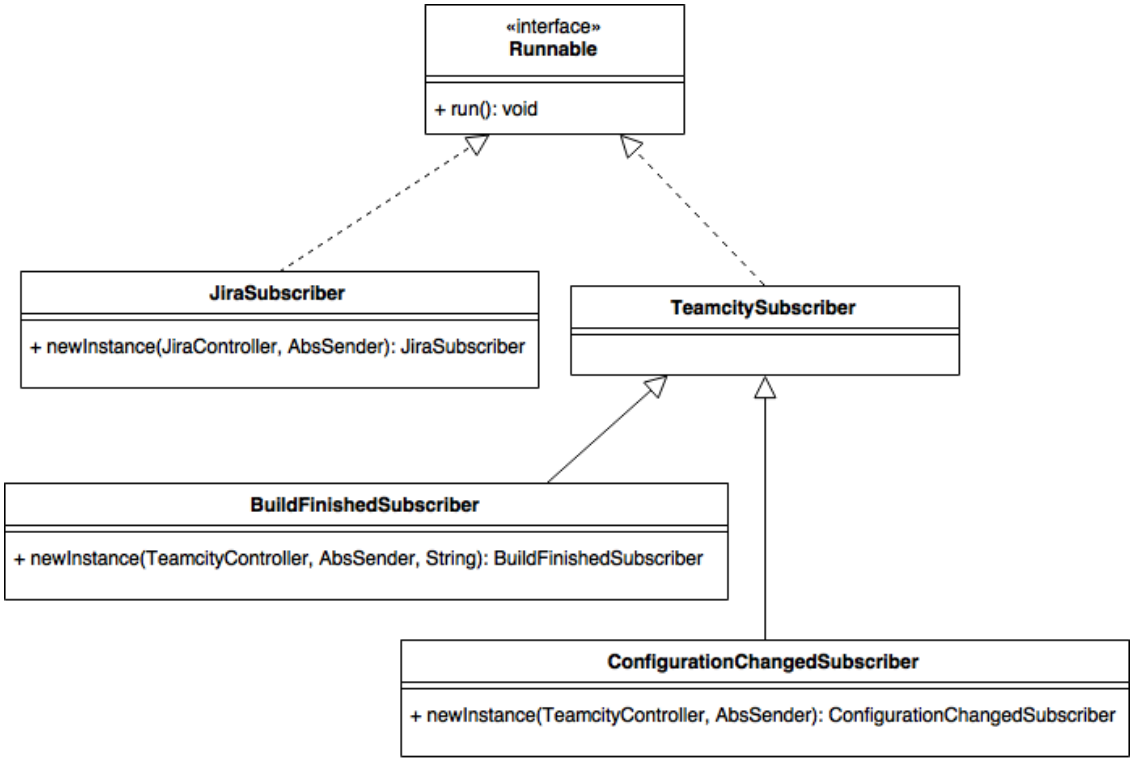


Рисунок 3.8 — Диаграмма классов модуля отправки уведомлений

Класс `ConfigurtionChangedSubscriber` предназначен для отслеживания изменений в конфигурациях проектов. Класс `BuildFinished-Subscriber` предназначен для отслеживания завершения сборки проекта. Экземпляр этого класса будет создаваться каждый раз при запуске сборки, далее он будет обращаться к TeamCity через короткие интервалы времени и, после завершения сборки, отправлять сообщение в чат с результатами сборки, при этом завершая свою работу.

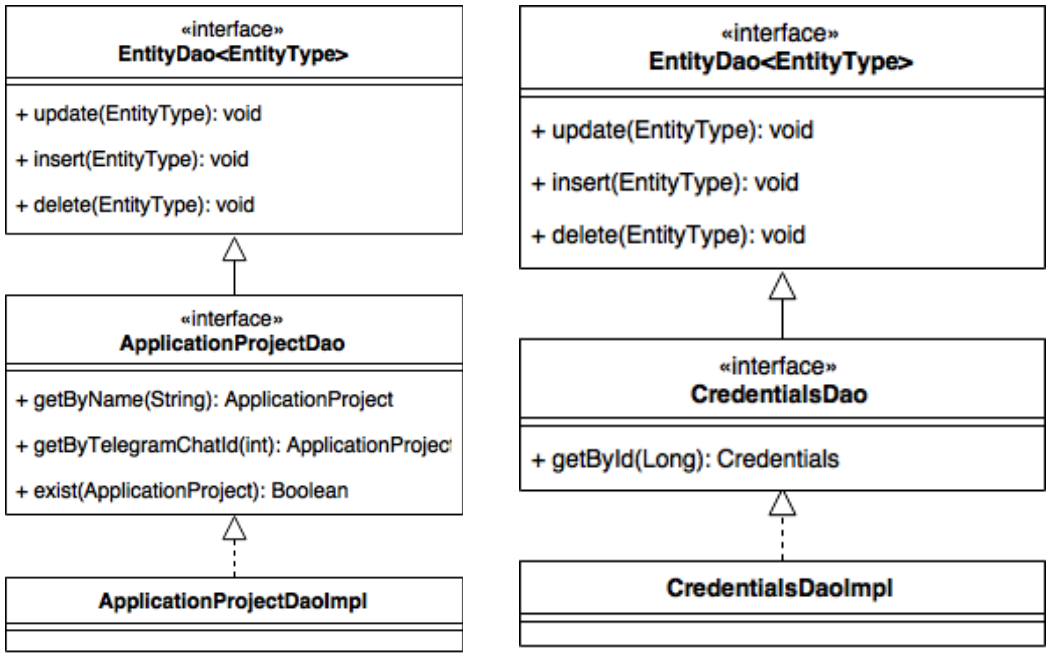
Для получения экземпляров класса `ConfigurtionChangedSubscriber` и `BuildFinishedSubscriber` у каждого реализован статический метод `new-Instance`, который принимает экземпляр класса `TeamcityController`, с помощью которого будет производится взаимодействие с системой TeamCity, и экземпляр `AbsSender` для отправки сообщений в чат. Класс `BuildFin-ishedSubscriber` дополнительно принимает идентификатор конфигурации проекта в TeamCity.

3.7 Модуль доступа к данным

Модуль доступа к данным будет реализовывать логику работы с базой данных, а именно: позволять обращаться к таблицам в базе данных и выполнять с ними различные операции. Работа с базой данных будет производится с использованием технологии Object-Relational Mapping (ORM). В качестве ORM фреймворка будет использована библиотека Hibernate.

Класс `ApplicationProjectDaoImpl` (см. рисунок 3.9, а) является точкой доступа к данным о проектах компании. Данный класс реализует следующие методы: `getByName` (метод принимает строку с именем проекта и возвращает объект `ApplicationProject`), `getByTelegramChatId` (метод принимает идентификатор чата группы в Telegram и возвращает соответствующий этому чату объект `ApplicationProject`), `update` (метод принимает экземпляр класса `ApplicationProject` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `ApplicationProject` и создаёт новую запись о проекте в базе данных), `delete` (принимает экземпляр класса `ApplicationProject` и удаляет соответствующую запись из базы данных), метод `exist` (принимает экземпляр класса `ApplicationProject` и проверяет, существует ли такая запись в базе данных, и возвращает значение true/false).

Класс `CredentialsDaoImpl` (см. рисунок 3.9, б) предоставляет интерфейс для доступа к учетным данным для входа во внешнюю систему. Данный класс реализует следующие методы: `getById` (метод принимает идентификатор записи в базе данных и возвращает соответствующий объект `Credentials`), `update` (метод принимает экземпляр класса `Credentials` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `Credentials` и создаёт новую запись с учетными данными в базе данных), `delete` (принимает экземпляр класса `Credentials` и удаляет соответствующую запись из базы данных).

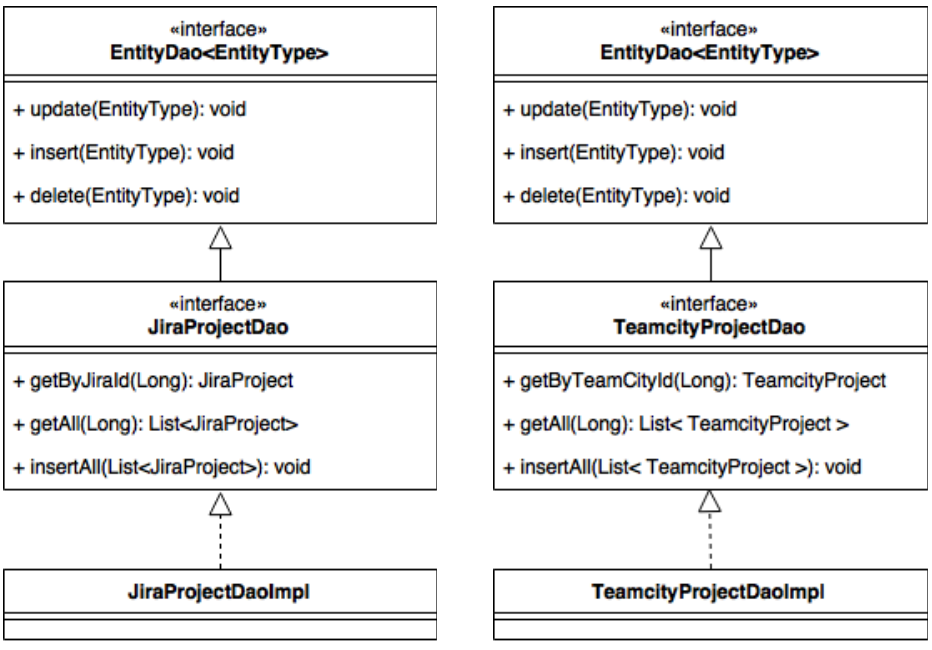


а) б)

Рисунок 3.9 — Диаграмма классов модуля доступа к данным:
а) ApplicationProjectDao, б) CredentialsDao

Класс `JiraProjectDaoImpl` (см. рисунок 3.10, а) является точкой доступа к данным о проектах в JIRA, реализует следующие методы: `getByJiraId` (метод принимает идентификатор проекта в JIRA и возвращает объект `JiraProject`), `getAll` (метод принимает идентификатор чата группы в Telegram и возвращает список всех проектов для данной группы), `update` (метод принимает экземпляр класса `JiraProject` и обновляет информацию о нём), метод `insert` (принимает объект класса `JiraProject` и создаёт новую запись), метод `insertAll` (принимает список объектов класса `JiraProject` и создаёт соответствующее количество записей), `delete` (принимает экземпляр класса `JiraProject` и удаляет соответствующую запись).

Класс `TeamCityProjectDaoImpl` (см. рисунок 3.10, б) является точкой доступа к данным о проектах в TeamCity, реализует следующие методы: `getByTeamCityId` (метод принимает идентификатор проекта в TeamCity и возвращает объект `TeamcityProject`), `getAll` (метод принимает идентификатор чата группы в Telegram и возвращает список всех проектов для данной группы), `update` (метод принимает экземпляр класса `TeamcityProject` и обновляет информацию о нём), метод `insert` (принимает объект класса `TeamcityProject` и создаёт новую запись о проекте), метод `insertAll` (принимает список объектов класса `TeamcityProject` и создаёт соответствующее количество записей о проектах), `delete` (принимает экземпляр класса `TeamcityProject` и удаляет соответствующую запись).

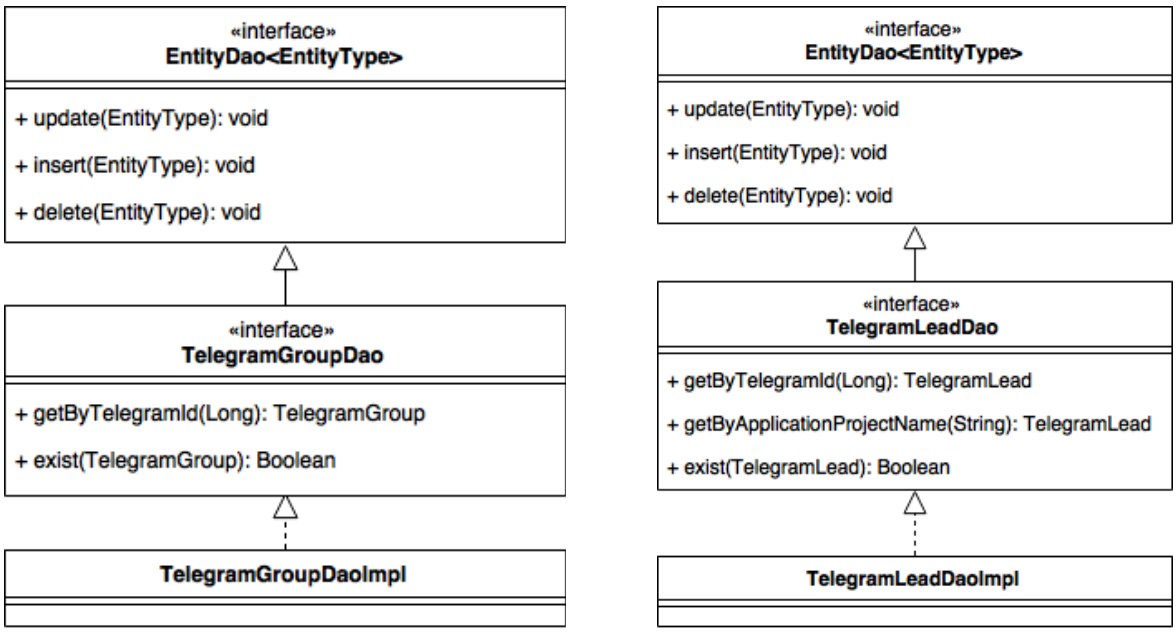


а) б)

Рисунок 3.10 — Диаграмма классов модуля доступа к данным:
а) JiraProjectDao, б) TeamcityProjectDao

Класс `TelegramGroupDaoImpl` (см. рисунок 3.11, а) является точкой доступа к данным, связанным с чат-группами в Telegram. Данный класс реализует следующие методы: `getByTelegramId` (метод принимает идентификатор группы в Telegram и возвращает объект `TelegramGroup`), `update` (метод принимает экземпляр класса `TelegramGroup` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `TelegramGroup` и создаёт новую запись о проекте в базе данных), `delete` (принимает экземпляр класса `TelegramGroup` и удаляет соответствующую запись из базы данных), `exist` (принимает экземпляр класса `TelegramGroup` и проверяет, существует ли такая запись в базе данных, и возвращает значение true или false).

Класс `TelegramLeadDaoImpl` (см. рисунок 3.11, б) является точкой доступа к данным, связанным с лидерами чат-групп в Telegram. Данный класс реализует следующие методы: `getByTelegramId` (метод принимает идентификатор группы в Telegram и возвращает объект `TelegramLead`), `getByApplicationProjectName` (метод принимает название проекта компании и возвращает объект класса `TelegramLead`), `update` (метод принимает экземпляр класса `TelegramLead` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `TelegramLead` и создаёт новую запись о проекте в базе данных), `delete` (принимает экземпляр класса `TelegramLead` и удаляет соответствующую запись из базы данных), `exist` (принимает экземпляр класса `TelegramLead` и проверяет, существует ли такая запись в базе данных, и возвращает значение true/false).

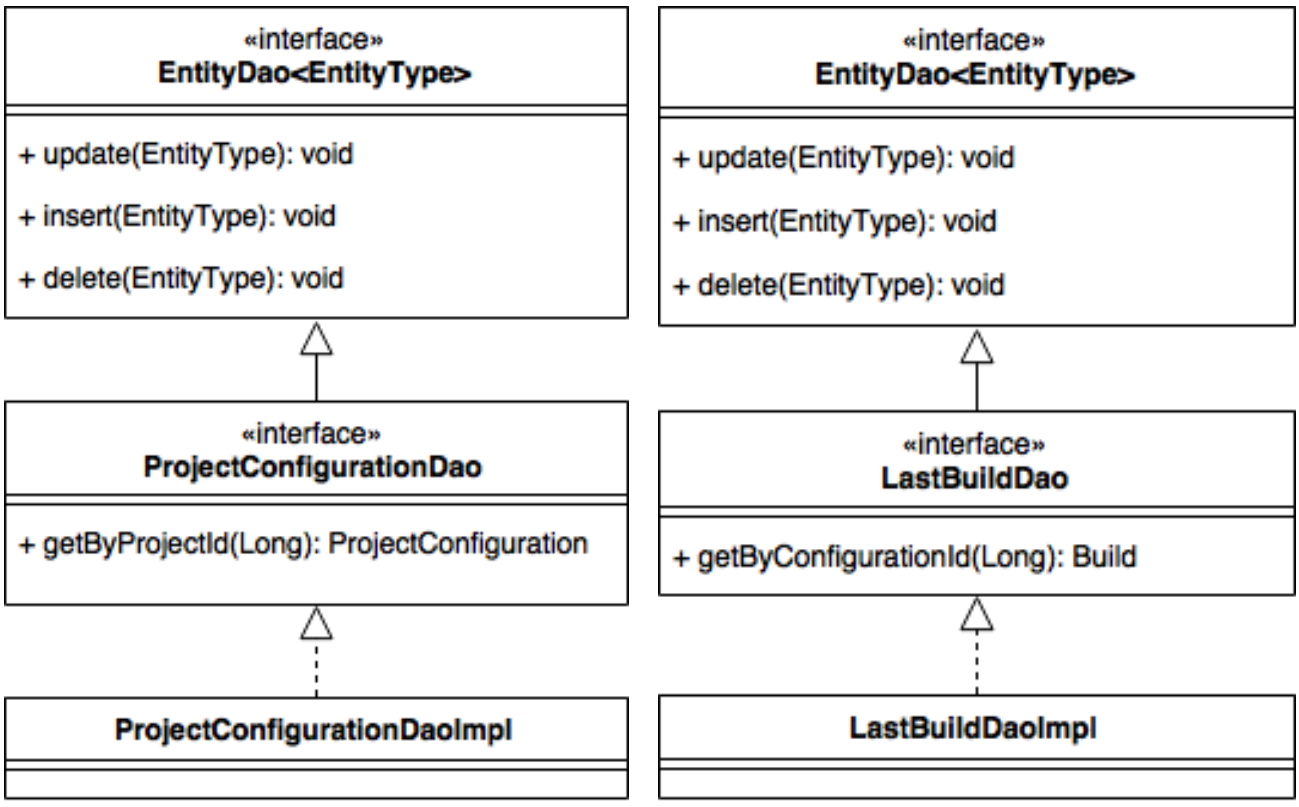


а) б)

Рисунок 3.11 — Диаграмма классов модуля доступа к данным:
а) TelegramGroupDao, б) TelegramLeadDao

Класс `ProjectConfigurationDaoImpl` (см. рисунок 3.12, а) предоставляет интерфейс для доступа к данным, связанным с конфигурациями проектов в TeamCity. Данный класс реализует следующие методы: `getByProjectId` (метод принимает идентификатор проекта в TeamCity и возвращает объект `ProjectConfiguration`), `update` (метод принимает экземпляр класса `ProjectConfiguration` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `ProjectConfiguration` и создаёт новую запись о проекте в базе данных), `delete` (принимает экземпляр класса `ProjectConfiguration` и удаляет соответствующую запись из базы данных).

Класс `LastBuildDaoImpl` (см. рисунок 3.12, б) является точкой доступа к данным, связанным с лидерами чат-групп в Telegram. Данный класс реализует следующие методы: `getByConfigurationId` (метод принимает идентификатор конфигурации проекта в TeamCity и возвращает объект `LastBuild`), `update` (метод принимает экземпляр класса `LastBuild` и обновляет информацию о нём в базе данных), метод `insert` (принимает экземпляр класса `LastBuild` и создаёт новую запись о проекте в базе данных), `delete` (принимает экземпляр класса `LastBuild` и удаляет соответствующую запись из базы данных).



а) б)

Рисунок 3.12 — Диаграмма классов модуля доступа к данным:
а) ProjectConfigurationDao, б) LastBuildDao

3.8 Модуль взаимодействия с пользователем

Данный модуль будет отвечать за формирование сообщений-ответов пользователю, в том числе и за предоставление отличного от отправки простых текстовых сообщений метода взаимодействия. Модуль взаимодействия с пользователем планируется реализовать с помощью Telegram Bot API.

Данный класс представлен интерфейсом `OutgoingMessageBuilder` и классами `KeyboardMessageBuilder`, `JiraTaskChangedMessageBuilder`, `TeamcityConfigChangedMessageBuilder` и `BuildFinishedMessageBuilder` (см. рисунок 3.13).

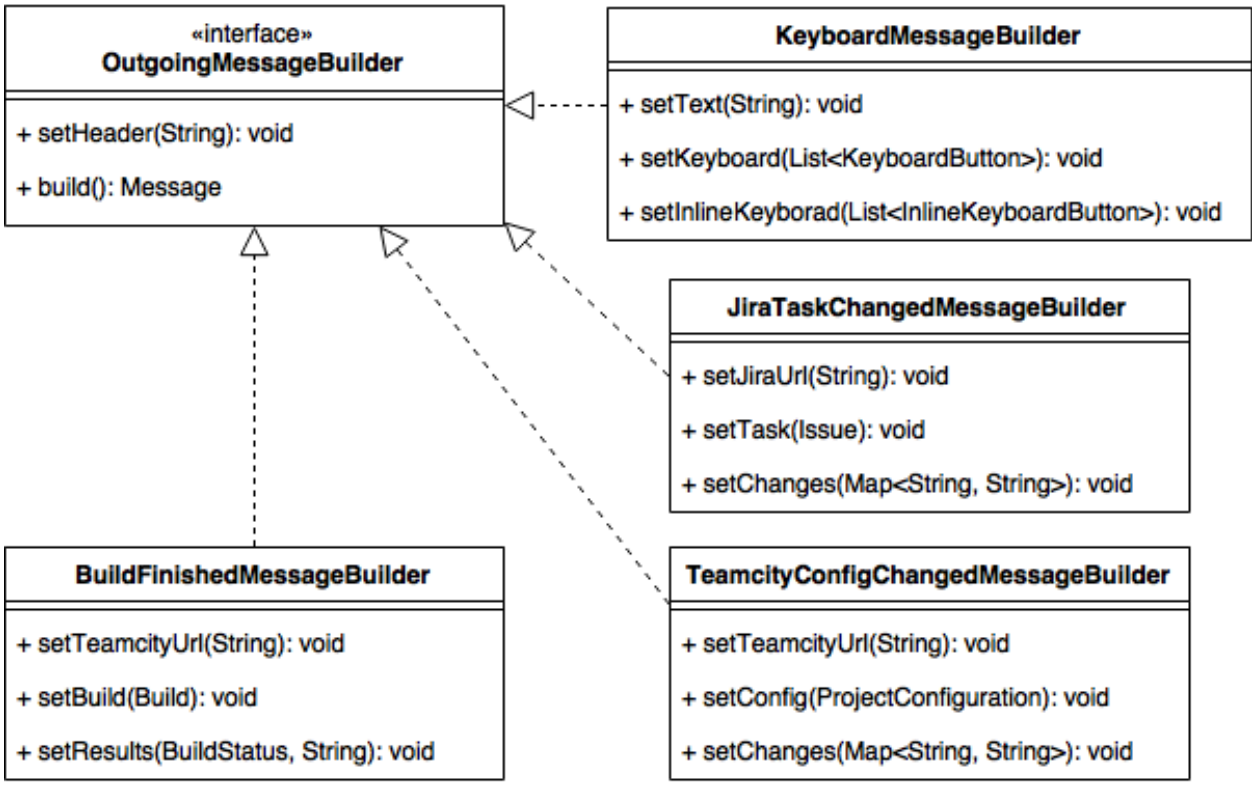


Рисунок 3.13 — Диаграмма классов модуля
взаимодействия с пользователем

Классы, которые реализуют интерфейс `OutgoingMessageBuilder` предоставляют методы для построения сообщения. У этого интерфейса описаны следующие методы: `setHeader` (метод принимает строку с надписью, которая кратко описывает все остальное сообщение, например «Сборка проекта окончена» для сообщения, которое будут получать пользователи после завершения сборки проекта) и `build` (метод собирает все части сообщения и возвращает объект класса `Message`, который будет передан в метод `sendMessage` класса `BotHandler` для отправки сообщения в чат)

Класс `KeyboardMessageBuilder` предназначен для построения сообщений, содержащих Telegram клавиатуры и встроенные клавиатуры. Класс имеет следующие публичные методы: `setText`, который принимает строку с текстом сообщения, методы `setKeyboard` и `setInlineKeyboard`, которые соответственно принимают списки `List<KeyboardButton>` и `List<InlineKeyboardButton>`, где `KeyboardButton` это кнопка обычной клавиатуры а `InlineKeyboardButton` — кнопка встроенной клавиатуры.

Класс `JiraTaskChangedMessageBuilder` предназначен для создания сообщений об изменении задач в системе JIRA. Реализованы следующие методы: `setJiraUri` (метод принимает строку со ссылкой на задачу в JIRA и добавляет ссылку в сообщение для возможности перехода по ссылке из чата), `setTask` (метод принимает экземпляр класс `Issue`, который содержит информацию о задаче) и `setChanges` (метод принимает коллекцию вида «ключ-значение», в которой ключ это поле задачи, которое было изменено, а значение — соответственно новое значение этого поля).

Класс `TeamcityConfigChangedMessageBuilder` предназначен для построения сообщений об изменении конфигурации проекта в TeamCity. Реализованы следующие методы: `setTeamCityUri` (метод принимает строку со ссылкой на конфигурацию в TeamCity и добавляет ссылку в сообщение для возможности перехода по ссылке из чата), `setConfig` (метод принимает экземпляр класс `ProjectConfiguration`, который содержит информацию о конфигурации) и `setChanges` (метод принимает коллекцию вида «ключ-значение», в которой ключ то, чтобы было изменено в конфигурации, а значение — соответственно на что было изменено).

Класс `BuildFinishedMessageBuilder` предназначен для построения сообщений о завершении сборки проекта в TeamCity. Реализованы следующие методы: `setTeamCityUri` (метод принимает строку со ссылкой на конфигурацию в TeamCity и добавляет ссылку в сообщение для возможности перехода по ссылке из чата), `setBuild` (метод принимает экземпляр класс `Build`, который содержит информацию о конфигурации) и `setResults` (метод принимает экземпляр класса-перечисления BuildStatus и строку с описанием результатов сборки).

3.9 Выводы

В данной главе была проведена декомпозиция программного средства на функциональные блоки исходя из полученных в ходе проектирования структурных модулей. Для каждого блока была приведена структура классов, были описаны цели каждого класса, их методы и поля, а также взаимодействие классов между собой. Также была спроектирована диаграмма классов.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Запуск сборки проекта в TeamCity

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Одной из наиболее значимых этапов разработки программного средства 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 программного обеспечения сопровождается написанием большого количества разнообразных тестов. Каждая новая функциональность программы тщательно проверяется различными методами. Ниже приведены основные виды тестирования:

— модульное тестирование;
— функциональное тестирование;
— тестирование производительности;
— тестирование совместимости;
— тестирование интерфейса пользователя.

При написании тестов для различных уровней приложения значительно снижаются риски нарушения работоспособности приложения при внесении различных изменений в существующий функционал. Полное покрытие тестами занимает значительную часть от общего времени работы программиста над проектом, но временные затраты окупаются надежностью и стабильностью программного обеспечения. По мере усложнения кода проекта стоимость устранения дефектов ПО может экспоненциально возрастать. Инструменты статического и динамического анализа помогают снизить эти затраты благодаря обнаружению программных ошибок на ранних этапах жизненного цикла ПО.

В ходе разработки программного средства были использованы следующие виды тестирования: модульное и функциональное. Модульное тестирование производилось с помощью библиотеки JUnit, версии 4.12. Функциональное тестирование производилось при помощи мобильного устройства Huawei GR5, версия ОС Android 5.1.1 , версия Telegram v3.17.1 for Android, а также на персональном компьютере MacBook Pro, версия macOS Sierra 10.12.4, версия Telegram 2.96.94323 for macOS. В ходе функционального тестирования на этих устройствах были проверены тестовые случаи, которые будут рассмотрены далее в этой главе.

5.1 Функциональное тестирование

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной

спецификации), при условии, что функции отрабатывали на различных значениях. При тестировании предполагается обработка данных и предсказуемая реакция приложения, когда данные, поданные на вход не являются корректными. Результаты проведенного над приложением функционального тестирования представлены в таблице 5.1.

Таблица 5.1 — Тестовые случаи и результаты их выполнения

Компонент	Название тестового случая	Шаги теста и ожидаемый результат	Результат выполнения теста
1	2	3	4
Взаимодействие с TeamCity	Аутентификация с корректными url, логином и паролем	1 Ввести команду authenticate_teamcity 2 Ввести url, логин и пароль 3 Отправить сообщение в чат Ожидается: сообщение об успешной аутентификации	Тест пройден успешно
	Получение списка проектов	1 Пройти аутентификацию в TeamCity 2 Ввести команду get_teamcity_projects 3 Отправить сообщение Ожидается: сообщение со списком проектов	Тест пройден успешно
	Получение списка конфигураций проекта	1 Пройти аутентификацию в TeamCity 2 Ввести команду get_project_configs 3 Ввести ID проекта 4 Отправить сообщение Ожидается: сообщение со списком конфигураций проекта	Тест пройден успешно
	Запуск сборки проекта	1 Пройти аутентификацию в TeamCity 2 Ввести команду build 3 Ввести ID проекта и ID конфигурации 4 Отправить сообщение Ожидается: сообщение о запуске сборки проекта	Тест пройден успешно

Окончание таблицы 5.1

1	2	3	4
Взаимодействие с JIRA	Аутентификация с корректными url, логином и паролем	1 Ввести команду authenticate_jira 2 Ввести url, логин и пароль 3 Отправить сообщение Ожидается: сообщение об успешной аутентификации	Тест пройден успешно
	Закрытие задачи	1 Пройти аутентификацию в JIRA 2 Ввести команду close_issue 3 Ввести ID задачи 4 Отправить сообщение Ожидается: сообщение об успешном закрытии задачи	Тест пройден успешно
	Получение списка задач	1 Пройти аутентификацию в JIRA 2 Ввести команду get_issues 3 Отправить сообщение Ожидается: сообщение со списком задач	Тест пройден успешно
	Изменение параметров задачи	1 Пройти аутентификацию в JIRA 2 Ввести команду update_issue 3 Ввести название обновляемого поля и новое значение 4 Отправить сообщение Ожидается: сообщение об успешном изменении задачи	Тест пройден успешно

Из таблицы 5.1 следует, что разработанное программное средство хорошо справилось с тестами, что свидетельствует о его работоспособности. Все выявленные в процессе тестирования недостатки были оперативно устранены.

5.2 Модульное тестирование

Модульное тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода.

Суть состоит в написании тестов для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

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

Разработка через тестирование — техника разработки программного обеспечения, 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, 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.

Для тестирования будет использована библиотека JUnit. Для подключения библиотеки к проекту следует добавить с помощью фреймворка Apache Maven следующую зависимость:

```
<dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.12</version>
</dependency>
```

Полный перечень тестов находится в приложении Д. Ниже будут рассмотрены тесты наиболее важного функционала программного средства.

В классе `TeamcityController` наиболее важным функционалом является запуск сборки проекта. Для этого используется метод `startProjectBuilding`, который принимает идентификатор проекта и идентификатор конфигурации в виде строк. Тесты для метода представлены ниже.

```
@Test(expected = IllegalArgumentException.class)
public void startProjectBuildingWithEmptyArguments() {
  String projectId = "";
  String configurationId = "";
  TeamcityController controller = new TeamcityController();
  controller
     .startProjectBuilding(projectId, configurationId);
}
```

Листинг 5.1 — Тест метода startProjectBuilding с пустыми аргументами

```
    @Test(expected = IllegalArgumentException.class)
    public void startProjectBuildingWithNullArguments() {
      String projectId = null;
      String configurationId = null;
      TeamcityController controller = new TeamcityController();
      controller
         .startProjectBuilding(projectId, configurationId);
}
```

Листинг 5.2 — Тест метода startProjectBuilding с аргументами со значением null

```
    @Test(expected = NonExistProjectException.class)
    public void startProjectBuildingWithNonExistingProject() {
      String projectId = "QWE";
      String configurationId = "Build_DEV";
      TeamcityController controller = new TeamcityController();
      controller
         .startProjectBuilding(projectId, configurationId);
}
```

Листинг 5.3 — Тест метода startProjectBuilding несуществующего проекта

```
    @Test(expected = NonExistConfigurationException.class)
    public void startProjectBuildingWithNonExistingConfig() {
      String projectId = "Graduation Project";
      String configurationId = "Build_QWE";
      TeamcityController controller = new TeamcityController();
      controller
         .startProjectBuilding(projectId, configurationId);
}
```

Листинг 5.4 — Тест метода startProjectBuilding с несуществующей конфигурациуй

В листинге 5.1 представлен тест для этого метода в случае, если будут переданы пустые строки вместо идентификаторов. В листинге 5.2 представлен тест для случая, когда в метод будут переданы идентификаторы со значением `null` каждый. Во этих случаях ожидается, что будет сгенерировано исключение `IllegalArgumentException`.

В листинге 5.3 представлен тест для случая, когда передан идентификатор несуществующего в TeamCity проекта, ожидается, что будет сгенерировано исключение `NonExistProjectException`.

В листинге 5.4 представлен тест для случая, когда предпринята попытка запустить сборку несуществующей конфигурации существующего проекта. В этом случае будет сгенерировано исключение `NonExistConfiguration-Exception`.

Для класса `JiraController` наиболее важным функционалом является работа с задачами. Для этого в классе реализованы методы `createIssue`, `closeIssue`, `updateIssue`, `changeIssueProgress`. Методы и тесты для них представлены ниже.

1) `createIssue`, используется для создания новой задачи, принимает объект класса `Issue`.

2) `closeIssue`, используется для закрытия задачи, принимает идентификатор задачи.

3) `updateIssue`, используется для изменения каких-либо полей задачи, принимает идентификатор задачи, идентификатор изменяемого поля в виде экземпляра класса `Field` и новое значение поля.

4) `changeIssueProgress`, используется для изменения статуса задачи, принимает идентификатор задачи и новый статус задачи в виде экземпляра класса `ProgressStatus`.

В таблице 5.1 представлено покрытие тестами функционала наиболее важных модулей программного средства.

Таблица 5.1 — Покрытые программного средства тестами

Название модуля	Покрытие
Модуль аутентификации	8 случаев, ошибок не выявлено
Модуль интеграции с JIRA	22 случая, 4 ошибки исправлены
Модуль интеграции с TeamCity	14 случаев, 1 ошибка исправлена
Модуль обработки команд	4 случая, ошибок не выявлено
Модуль взаимодействия с Telegram	9 случаев, 1 ошибка исправлена

Покрытие JUnit тестами вышеприведенных в таблице 7.1 модулей позволило исправить 6 ошибок в реализации связей между модулями системы между собой и с внешними системами и снизить долю ручного тестирования в общем времени тестирования.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО СРЕДСТВА ДЛЯ УДАЛЕННОГО УПРАВЛЕНИЯ СБОРКОЙ ПРИЛОЖЕНИЙ И ВЗАИМОДЕЙСТВИЯ С СИСТЕМОЙ ОТСЛЕЖИВАНИЯ ЗАДАЧ

7.1 Характеристика программного средства

Целью дипломного проекта является разработка программного средства для обеспечения интеграции с системами Atlassian JIRA и TeamCity посредством чат-ботов, предоставляемых системой мгновенных сообщений Telegram. С помощью данного ПО пользователь сможет управлять запуском сборки проектов в TeamCity, изменять и закрывать задачи в JIRA, а также получать уведомления. Программ-аналогов данного продукта нет, так как многие процессы только начинают автоматизировать посредством чат-ботов. Предполагаемыми пользователями данного программного продукта являются IT-компании любого размера, которые в своем рабочем процессе используют системы Atlassian JIRA и TeamCity.

Распространение программного средства планируется осуществлять путем размещения его на специальной площадке продажи копий лицензий чат-ботов от Telegram — Telegram Bot Store.

Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течение 4 лет; планируется продать 50 лицензий в 2017 году, 75 лицензий в 2018 году, 150 лицензий в 2019 году и 300 лицензий в 2020 году.

Целью технико-экономического обоснования является определение экономической выгодности создания рассматриваемого программного обеспечения и дальнейшего его применения. Экономическая целесообразность инвестиций в разработку и реализацию программного продукта определяется на основе расчета и оценке следующих показателей:

— чистый дисконтированный доход;

— срок окупаемости инвестиций;

— рентабельность инвестиций в разработку программного продукта.

7.2 Расчет затрат на разработку программного средства

Затраты на основную заработную плату команды разработчиков определяется исходя из состава и численности команды, размеров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки программного обеспечения. Расчет производится по формуле:

$$З_о = \sum_{i=1}^{n} Т_{ч.i} \cdot Т_ч \cdot Ф_{эф.i} \cdot К_п, \qquad (7.1)$$

где n — количество исполнителей на конкретное программное средство;

$Т_{ч.i}$ — часовая тарифная ставка i-го исполнителя, руб.;

$Т_{ч}$ — количество рабочих часов в день, ч.;

$Ф_{эф.i}$ — эффективный фонд рабочего времени i-го исполнителя, дн.;

$К_{п}$ — коэффициент премирования (можно принять $К_{п} = 1,5$).

Примем тарифную ставку 1-го разряда равной 180,00 рублей. Среднемесячная норма рабочего времени составляет 168 часов.

Часовой тарифный оклад руководителя проекта с 14 разрядом составляет 180 · 3,25 / 168 = 3,48 рубля.

Часовой тарифный оклад инженера-программиста 10 разряда составляет 180 · 2,48 / 168 = 2,66 рубля.

Результаты расчета основной заработной платы исполнителей представлены в таблице 7.1.

Таблица 7.1 — Результаты расчета основной заработной платы

Исполнитель	Разряд	Тарифный коэффициент	Месячная тарифная ставка, руб	Часовая тарифная ставка, руб	Плановый фонд рабочего времени, дн.	Заработная плата, руб
Руководитель проекта	14	3,25	584,60	3,48	30	835,10
Инженер-программист	10	2,48	446,90	2,66	90	1915,30
Основная заработная плата						2750,40

Затраты на дополнительную заработную плату команды разработчиков ($З_{д}$) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

$$З_{д} = \frac{З_{о} \cdot Н_{д}}{100}, \qquad (7.2)$$

где $Н_{д}$ — норматив дополнительной заработной платы ($Н_{д} = 15\%$).

В нашем случае, дополнительная зарплата будет равна:

$$З_д = \frac{2750{,}40 \cdot 15}{100} = 412{,}56 \text{ руб}.$$

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной заработной платы и вычисляются по формуле:

$$З_{сз} = \frac{(З_о + З_д)}{100} \cdot Н_{соц}, \qquad (7.3)$$

где $Н_{соц}$ — норматив отчисления на социальные нужды (34+0,6%).

Отчисления на социальные нужды составляют:

$$З_{сз} = \frac{(2750{,}40 + 412{,}56)}{100} \cdot (34 + 0.6) = 1094{,}38\ руб$$

Расходы по статье «Машинное время» ($Р_м$), включающие оплаты машинного времени, необходимого для разработки и отладки программного продукта, осуществляется по формуле:

$$Р_м = Ц_м \cdot Т_ч \cdot С_р, \qquad (7.4)$$

где $Ц_м$ — цена одного машино-часа, руб;

$Т_ч$ — количество часов работы в день, ч.;

$С_р$ — длительность проекта, дн.

Стоимость машино-часа на предприятии составляет 1,50 рублей. Разработка проекта займет 90 дней. Количество рабочих часов в день — 8. Таким образом затраты по статье «Машинное время» составят:

$$Р_м = 1{,}50 \cdot 8 \cdot 90 = 1080{,}00\ руб.$$

Расходы по статье «Прочие затраты» включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются в процентах к основной заработной плате.

$$П_З = \frac{З_о \cdot Н_{пз}}{100}, \qquad (7.5)$$

где $H_{пз}$ — норматив прочих затрат, %.

Возьмем норматив прочих расходов $Н_{пз} = 10\%$ от основной заработной платы, тогда расходы по статье «Прочие затраты составят»:

$$П_З = \frac{2750{,}40 \cdot 10}{100} = 275{,}04\ руб.$$

Затраты по статье «Накладные расходы» ($Р_н$), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств, а также с расходами на общехозяйственные нужды, рассчитываются по формуле:

$$Р_н = \frac{З_о \cdot Н_{рн}}{100}, \qquad (7.6)$$

где $H_{рн}$ — норматив накладных расходов, %.

Примем $H_{рн}$ равным 70%, тогда:

$$Р_н = \frac{2750{,}40 \cdot 70}{100} = 1925{,}28\ руб.$$

Общая сумма расходов по всем статьям сметы на программный продукт рассчитывается по формуле:

$$С_р = З_о + З_д + З_{сз} + Р_м + П_з + Р_н \qquad (7.7)$$

Рассчитаем сумму расходов по всем статьям сметы:

$$С_р = 2750{,}40 + 412{,}56 + 1094{,}38 + 1080{,}00 + 275{,}04 + 1925{,}28 = 7537{,}66\ руб.$$

Кроме того, потребуются дальнейшие затраты на сопровождение и адаптацию $Р_{са}$, которые определяются по нормативу $Н_{рса}$:

$$Р_{са} = \frac{С_р \cdot Н_{рса}}{100}, \qquad (7.8)$$

где $Н_{рса}$ — норматив расходов на сопровождение и адаптацию, %;

$С_р$ — смета расходов без расходов на сопровождение и адаптацию, руб.

Примем $Н_{рса}$ равным 5%, тогда:

$$\mathrm{P_{са}} = \frac{7537{,}66 \cdot 5}{100} = 376{,}88\ руб.$$

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программного продукта $\mathrm{C_п}$ определяется по формуле:

$$C_п = C_р + P_{са} \tag{7.9}$$

Общая сумма расходов на разработку:

$$C_п = 7537{,}66 + 376{,}88 = 7914{,}54 \text{ руб.}$$

7.3 Расчет экономического эффекта от продажи программного продукта

Экономический эффект для разработчика программного обеспечения заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного программного средства.

Распространение приложения планируется осуществлять через онлайн-магазин Telegram чат-ботов — Telegram Bot Store.

Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течение 4 лет; планируется продать 50 лицензий в 2017 году, 75 лицензий в 2018 году, 150 лицензий в 2019 году и 300 лицензий в 2020 году. На основании маркетингового исследования отпускная цена одной копии лицензии составила 50 рублей.

Прибыль от продажи одной лицензии программного продукта определяется по формуле:

$$П_{ед} = Ц - НДС - \frac{З_р}{N}, \tag{7.10}$$

где Ц — отпускная цена одной копии лицензии программного продукта;

НДС — сумма налога на добавленную стоимость;

N — количество лицензий, которые купят клиенты;

$З_р$ — сумма расходов на разработку и реализацию.

Сумма налога на добавленную стоимость рассчитывается по формуле:

$$НДС = \frac{Ц \cdot Н_{дс}}{100 + Н_{дс}}, \tag{7.11}$$

где $Н_{дс}$ — ставка налога на добавленную стоимость, равняется 20 %.

Рассчитаем сумму налога на добавленную стоимость:

$$НДС = \frac{50 \cdot 20}{100 + 20} = 8{,}33\ руб.$$

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

$$З_р = C_п + \frac{C_п \cdot 15}{100} = 9101{,}72\ руб.$$

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

$$П_{ед} = 50 - 8.33 - \frac{9101{,}72}{575} = 25{,}84\ руб.$$

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

$$ЧП_{ед} = П_{ед} \cdot (1 - \frac{H_п}{100}), \qquad (7.12)$$

где $H_п$ — ставка налога на прибыль, 18%.

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

$$ЧП_{ед} = 25{,}84 \cdot \left(1 - \frac{18}{100}\right) = 21{,}19\ руб.$$

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

$$ЧП = ЧП_{ед} \cdot N, \qquad (7.13)$$

Прибыль по проекту за каждый год продаж составляет:

$$ЧП_1 = 21{,}19 \bullet 50 = 1054{,}50\ руб.$$
$$ЧП_2 = 21{,}19 \bullet 75 = 1589{,}25\ руб.$$
$$ЧП_3 = 21{,}19 \bullet 150 = 3178{,}50\ руб.$$
$$ЧП_4 = 21{,}19 \bullet 300 = 6357{,}00\ руб.$$

7.4 Расчет показателей эффективности разработки программного продукта

Для проведения сравнительного анализа размера суммы затрат на разработку программного средства и получаемого экономического эффекта необходимо привести их к одному единому моменту времени — началу расчетного периода, что обеспечит их сопоставимость. Для этого необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования (α) соответствующего года t, который определяется по формуле:

$$\alpha = (1 + E_н)^{t-t_p}, \qquad (7.14)$$

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

t_p — расчетный год, $t_p = 1$;

t — порядковый номер года.

На 01.05.2017 г. ставка рефинансирования составляет 15%. Используя формулу (7.13) рассчитаем коэффициенты дисконтирования:

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

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

2019 г.; $t_p = 3$; $\alpha = (1 + 0{,}15)^{1-3} = 0{,}76$

2020 г.; $t_p = 4$; $\alpha = (1 + 0{,}15)^{1-4} = 7{,}66$

Расчет показателей эффективности инвестиций по разработке продукта представлен в таблице 7.2.

Таблица 7.2 — Результаты расчета эффективности инвестиционного проекта по разработке программного продукта.

Показатель	Единицы измерения	Расчетный период			
		2017 г.	2018 г.	2019 г.	2020 г.
1	2	3	4	5	6
РЕЗУЛЬТАТ					
1 Экономический эффект	руб.	1054,50	1589,25	3178,5	6357,00

Продолжение таблицы 7.2

1	2	3	4	5	6
Коэффициент дисконтирования	доли ед.	1	0,87	0,76	0,66
2 Дисконтированный результат	руб.	1054,50	1382,87	2415,66	4195,62
3 Затраты на разработку программного средства	руб.	7914,54			
4 Дисконтированные инвестиции	руб.	7914,54			
ЭКОНОМИЧЕСКИЙ ЭФФЕКТ					
5 Чистый дисконтированный доход по годам	руб.	-6860,04	1382,87	2415,66	4195,62
6 Чистый дисконтированный доход нарастающим итогом	руб.	-6860,04	-5477,17	-3061,51	1134,11

Так как чистый дисконтированный доход больше нуля, то проект эффективен, то есть инвестиции в разработку данного ПО экономически целесообразны.

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта (P_u) по формуле:

$$P_u = \frac{ЧП_{ср}}{З_р} \cdot 100, \qquad (7.15)$$

где $ЧП_{ср}$ — среднегодовая величина чистой прибыли за расчетный период.

Среднегодовая величина чистой прибыли за расчетный период определяется по формуле:

$$ЧП_{ср} = \frac{\sum_{i=1}^{n} ЧП_i}{n}, \qquad (7.16)$$

где $ЧП_i$ — величина чистой прибыли за i-ый расчетный год;

n — расчетное количество лет.

Среднегодовая величина чистой составит:

$$ЧП_{ср} = \frac{1054{,}50 + 1589{,}25 + 3178{,}50 + 6357{,}0}{4} = 3043{,}56$$

Таки образом рентабельность инвестиций составит:

$$P_и = \frac{3043{,}66}{9101{,}72} \cdot 100 = 33\ \%$$

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

— чистый дисконтированный доход за четыре года составит 1134,11 руб.;

— затраты на разработку программного продукта окупятся на четвертый год его использования;

— рентабельность инвестиций составит 33 %.

Таким образом, разработка и реализация программного продукта являются эффективными, а также является целесообразным осуществлять инвестиции в его разработку.

ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом было реализовано программное средство, которое позволит объединить взаимодействие с TeamCity и JIRA с помощью Telegram чат-бота. Был реализован следующий функционал:

— запуск сборки проекта из чата;

— отправка уведомлений о результатах сборки проекта в групповой чат;

— закрытие задач в JIRA;

— 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).

Для взаимодействия с Telegram была использована библиотека TelegramBots, основанная на Telegram Bot API. Для реализации взаимодействия с JIRA REST API была использована библиотека rcarz/jira-client, а для взаимодействия с TeamCity REST API — фреймворк Jersey. Хранение данных осуществлялось с помощью СУБД PostgreSQL.

При разработке дипломного проекта была проведена декомпозиции программного средства на модули. Исходя из возможностей программного средства были определены назначения и задачи каждого модуля, связи модулей между собой. Также была спроектирована схема данных, определены цели хранения информации в таблицах, основные поля таблиц и связи таблиц между собой. Была спроектирована инфраструктура программного средства, выделены компоненты, которые необходимо установить на каждом устройстве для корректной работы программного средства.

В ходе функционального проектирования была проведена декомпозиция программного средства на функциональные блоки исходя из полученных в ходе проектирования структурных модулей. Для каждого блока была приведена структура классов, были описаны цели каждого класса, их методы и поля, а также взаимодействие классов между собой. Также была спроектирована диаграмма классов.

// TODO: остальные модули

Также было проведено технико-экономическое обоснование эффективности разработки и реализации дипломного проекта. В результате обоснования были получены следующие значения показателей эффективности: чистый дисконтированный доход за четыре года составит 1134,11 руб., затраты на разработку программного продукта окупятся на четвертый год его использования, рентабельность инвестиций составит 33 %. Таким образом, разработка и реализация программного продукта являются эффективными, осуществлять инвестиции в его разработку целесообразно.

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).

[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

[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)

Вводный плакат

ПРИЛОЖЕНИЕ Б
(обязательное)

Структурная схема