Geekbrains

Факультет Разработчик

Создание функционального телеграм-бота

Дипломный проект

студента 1 года обучения

специальности “Java-разработчик”

Мухин В. Д.

Балашиха

2024

[Введение 3](#_Toc656644852)

[Как написать бота 4](#_Toc138260949)

[Создание бота 5](#_Toc1915570729)

[Что выбрать для телеграм бота? Вебхуки или лонгполлинг? 5](#_Toc1409346045)

[Зачем использовать брокеры сообщений 9](#_Toc1037253211)

[Что такое RabbitMQ 10](#_Toc95570450)

[Почему выбирают RabbitMQ: 12](#_Toc1534025562)

[PostgreSQL в проекте 13](#_Toc1878362201)

[Написание кода 15](#_Toc350830310)

[Как писать чистый и красивый код 16](#_Toc1502823162)

[Имена 16](#_Toc1368651052)

[Одна функция — одна задача 17](#_Toc521390417)

[Код и комментарии 18](#_Toc808848623)

[Важность форматирования 19](#_Toc1426785323)

[Сначала — try-catch-finally, потом — всё остальное 21](#_Toc1740375408)

[Итог как писать чистый код 23](#_Toc1460843634)

[О проекте 24](#_Toc1874897262)

[Структура проекта: 25](#_Toc1407032578)

[Выбор технологий и инструментов разработки 25](#_Toc1865361611)

[В родительском pom-файле указаны: 25](#_Toc1621398870)

[Тесты 28](#_Toc777862312)

[Нагрузочное тестирование с JMeter 29](#_Toc1666745765)

[Тестирование производительности приложения с JMeter 29](#_Toc2115054274)

[Тестирование производительности с JMeter включает в себя: 30](#_Toc545353750)

[Создание плана тестирования производительности в JMeter 32](#_Toc2120448457)

[Шаг 1) Добавить Thread Group (группа потоков) 33](#_Toc1686118224)

[Шаг 2) Добавление элементов JMeter 36](#_Toc1452987804)

[Шаг 3) Добавление Graph Result (Графические результаты) 39](#_Toc1342353421)

[Шаг 4) Запустите тест и получите результат тестирования 40](#_Toc219530912)

[Итог 44](#_Toc581938958)

[Скриншоты результатов работы 45](#_Toc1345090292)

[Источники 47](#_Toc1920819444)

Цель: разработать телеграмм-бота, который будет предоставлять пользователям возможность сохранять, организовывать и управлять текстовыми данными, документами и изображениями прямо в чате.

Задачи:

-Изучить возможности API телеграмма для создания ботов.

-Создать и настроить базу данных для хранения текстовых данных, документов и изображений.

-Настроить функционал бота для сохранения и извлечения данных.

В современном мире создание функционального телеграм-бота имеет большую актуальность из-за следующих причин:

Большое количество пользователей: Телеграм — один из самых популярных мессенджеров в мире, с более чем 500 миллионами активных пользователей. Создание ботов для этой платформы предоставляет возможность достичь широкой аудитории.

Удобство использования: Телеграм-боты предлагают простой и удобный способ взаимодействия с пользователями. Благодаря чат-интерфейсу боты могут быть использованы как для автоматизации рутиных задач, так и для предоставления информации или услуг.

Интеграция с другими сервисами: Телеграм предоставляет API для разработчиков, что позволяет легко интегрировать ботов с другими сервисами и платформами. Это позволяет улучшить пользовательский опыт и расширить функционал бота.

Персонализация и адаптивность: Создание функционального телеграм-бота позволяет адаптировать его под конкретные потребности и предпочтения аудитории. Это помогает улучшить взаимодействие с пользователями и повысить их удовлетворенность.

Таким образом, разработка функционального телеграм-бота представляет собой актуальную и перспективную тему для дипломной работы, которая позволит студентам изучить современные технологии в области чат-ботов и приобрести ценный опыт в разработке программного обеспечения.

# Введение

Телеграмм - одно из самых популярных мессенджеров в мире, который предоставляет широкие возможности для общения, обмена файлами и информацией. Создание функционального телеграмм-бота, способного сохранять текст, документы и изображения, позволит пользователям удобно хранить и организовывать свои данные прямо в мессенджере.

Бот — это аккаунт в Телеграме, предназначенный для того, чтобы им управляла программа (сервер).

Боты могут участвовать в личных сообщениях с пользователями, быть участниками групп и каналов. Они могут отправлять сообщения, отвечать на команды пользователей, отправлять картинки и ещё много чего делать.

Боты ограничены в возможностях по сравнению с обычными пользователями, чтобы их приватность не нарушалась: например, боты не могут самостоятельно вступать в группы.

## Как написать бота

Если коротко:

1. Зарегистрируйте бота.

2. Выберите библиотеку на любимом языке программирования (в данном случае Java).

3. Пишите код, параллельно читая документацию к библиотеке.

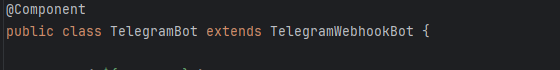
# Создание бота

Единственная информация о Телеграм-ботах, которой в интернете полным-полно: как создать бота. Это делается через специального бота BotFather. Когда вы создадите бота, BotFather даст вам его токен. Токен выглядит примерно так:

**110201543:AAHdqTcvCH1FgWJxfSsadfsSEDFPALDsaw**

Именно с помощью токена вы сможете управлять ботом.

# Что выбрать для телеграм бота? Вебхуки или лонгполлинг?



Polling, или опрос, – это процесс, при котором клиентский скрипт периодически отправляет запросы к серверу для проверки наличия новой инфы. В телеге это регулярный запрос к АПИ для получения новых обновлений или сообщений.

При увеличении нагрузки Polling может стать бутылочным горлышком. Такую проблемку можно решить увеличением интервала между запросами в моменты низкой активности и его уменьшение в пиковые часы.

Если бот обрабатывает состояния или сессии пользователей нужно учитывать, что при Polling состояние должно храниться на стороне сервера. Это означает необходимость использования БДшек и т.п

Постоянные запросы к серверу создают дополнительную нагрузку. Необходимо настраивать подходящий баланс, чтобы избежать чрезмерного трафика и потенциального блокирования вашего бота из-за слишком частых запросов. API в телеграме ожидаемо имеет ограничения на количество запросов в секунду. Необходимо убедиться, что бот не превышает эти лимиты.

Согласно документации максимальное количество сообщений, отправляемых ботом, ограничено примерно 30 сообщениями в секунду для обычных сообщений и около 20 сообщений в минуту для групповых чатов.

Если бот достигает этих лимитов, будут вылетать ошибки RetryAfter от API. Постоянные попытки повторной отправки сообщений, игнорируя ошибки API, могут привести к временной блокировке бота.

Бот каждые две секунды спрашивает у сервера Telegram, не пришло ли новых сообщений. В реальности тут конечно должна быть обработка ошибок, логирование и другие элементы

С Polling было связано множество трудностей. Во-первых, нагрузка на сервера. Каждый бот постоянно что то запрашивал у сервера. Представьте себе тысячи ботов, делающих это каждую секунду. Это было не только неэффективно с точки зрения трафика, но и могло вызывать задержки в доставке сообщений.

Во-вторых каждый раз, когда соединение прерывалось или сервер не отвечал, это могло привести к потере апдейтов. Приходилось тщательно продумывать механизмы повторных попыток и логирования, чтобы минимизировать потери данных.

Со временем, когда телега становилась всё более попсовой, разработчики начали искать более эффективные способы взаимодействия с API. И здесь появляются вебхуки. В отличие от Polling, где бот активно запрашивал данные, Webhook позволяет серверу телеграма самому отправлять обновления боту, как только они появляются.

WebhookWebhook – это, по сути, колллбек. Это URL, который вы предоставляете телеграму, и по которому телеграм будет отправлять обновления в формате JSON каждый раз, когда вашему боту приходит сообщение или происходит другое событие.

В начале вам нужно настроить Webhook. Это делается путем отправки запроса к Telegram API с указанием URL вашего сервера. Телега будет использовать этот URL для отправки обновлений.

Как только кто-то взаимодействует с ботом (например, отправляет ему сообщение), телеграм делает запрос на юрл вебхук с деталями этого события. Эти данные приходят в виде JSON-объекта, который содержат информацию, к примеру:

"update\_id": 12345661289,  
 "message": { "message\_id": 111, "from": { "id": 232, "is\_bot": false, "first\_name": "vladislav", "username": "vludisluv" }, "chat": { "id": 823, "first\_name": " vladislav ", "username": " vludisluv", "type": "private" }, "date": 16094523089200, "text": "привет" }}

Сервер затем обрабатывает этот запрос, парсит полученный JSON и выполняет необходимые действия, например, отвечает на сообщение пользователя.

Следующий шаг - это создание обработчика для входящих сообщений. В коде это обычно реализуется через определение функций с декораторами, указывающими на типы обрабатываемых сообщений (например, текст, изображения и т.д.).

|  |  |  |
| --- | --- | --- |
| Критерии | Polling | Webhook |
| Время отклика | Относительно медленное | Почти мгновенное |
| Нагрузка на сервер | Высокая, требует постоянных запросов | Низкая, сервер ждёт уведомлений |
| Сложность реализации | Простая, легко реализуемая | Средняя, требует настройки HTTPS, SSL |
| Масштабируемость | Ограничена, увеличение нагрузки на сервер с ростом числа пользователей | Высокая, эффективна при большом количестве пользователей |
| Использование ресурсов | Активное, постоянный опрос сервера | Пассивное, проще говоря принимай, когда пришло |
| Управление трафиком | Неэффективное, из-за постоянных запросов | Эффективное, обрабатываются только актуальные события |
| Безопасность | Более уязвим для ддос атак из-за открытых запросов | Более безопасный, использует зашифрованные соединения |

# Зачем использовать брокеры сообщений

Если сломается брокер, в худшем случае вы потеряете часть сообщений, но ядро сервиса всё ещё будет работать, так как берет информацию из базы данных. Сообщения будут накапливаться, и когда сервер вернётся, то быстро прочитает и обработает образовавшийся долг. Но иногда возникает частичная деградация: какая-то информация может оказаться не самой актуальной, хотя пользователи, скорее всего, этого даже не заметят. Пример — новостная лента в соцсетях. Если она не будет обновляться некоторое время, вы все равно сможете посмотреть опубликованные посты.

Бывает, что сервисы, спроектированные и написанные в краткие сроки, не имеют брокера сообщений. Потребность реализовать этот функционал осознаётся значительно позже. И здесь есть два варианта. Первый — сэкономить и в качестве брокера сообщений использовать базу данных. Это не очень хорошо, потому что решает проблему только на раннем этапе.

Второй вариант — настроить брокер сообщений. Так вы сможете обеспечить своему приложению технологический запас для будущего развития. Одним из наиболее популярных брокеров сообщений остаётся RabbitMQ.

# Что такое RabbitMQ

RabbitMQ — распределённый и горизонтально масштабируемый брокер сообщений.

Упрощённо его устройство можно описать так:

-паблишер, который отправляет сообщения;

-очередь, где хранятся сообщения;

-подписчики, которые выступают получателями сообщений.

RabbitMQ передаёт сообщения между поставщиками и подписчиками через очереди. Сообщения могут содержать любую информацию, например, о событии, произошедшем на сайте.

RabbitMQ отлично подходит для интеграции разных компонентов, создания микросервисов, потоковой передачи данных в режиме реального времени или при передаче работы удалённым работникам. Его используют крупные компании, в числе которых Bloomberg, Reddit, WeWork, NASA и др.

# Почему выбирают RabbitMQ:

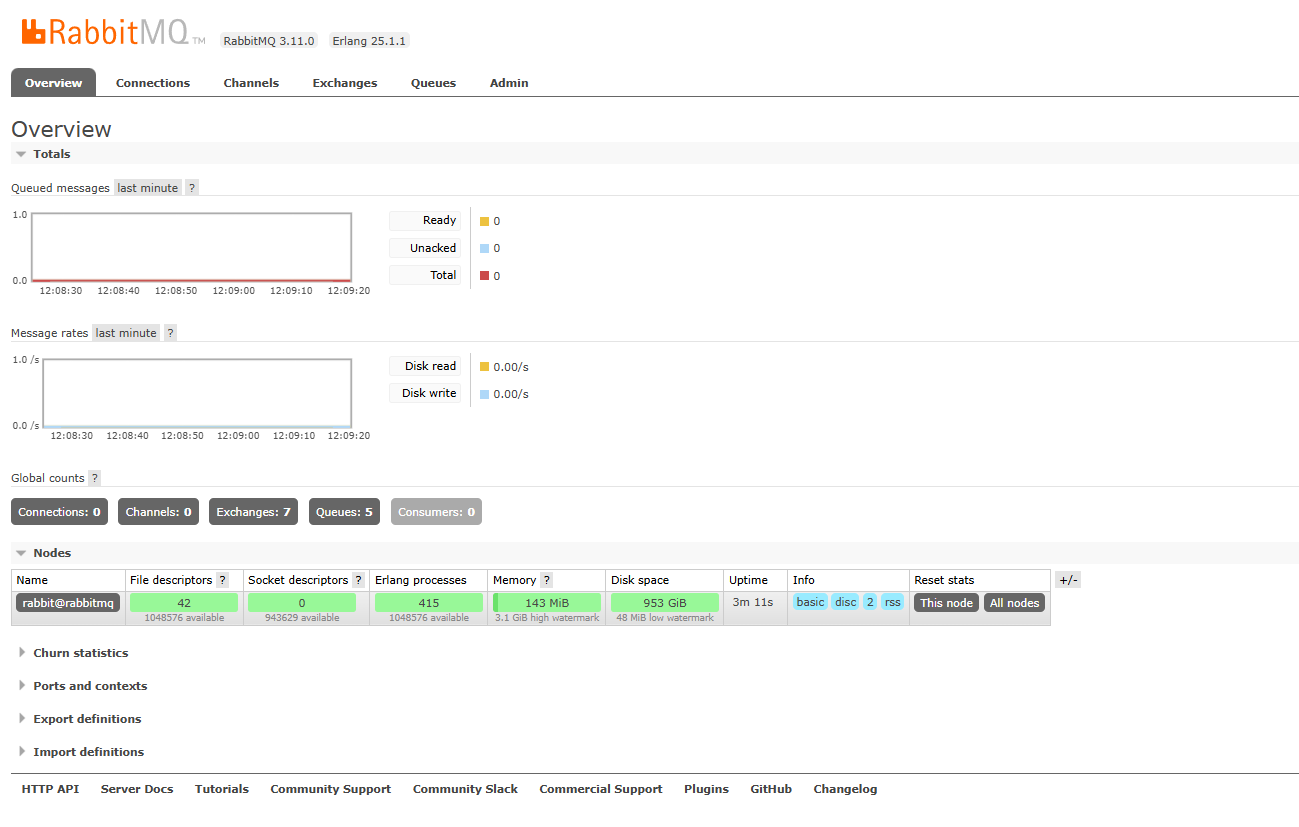
RabbitMQ поддерживает несколько протоколов: AMQP, MQTT, STOMP и др., что позволяет использовать его в разных сценариях.

RabbitMQ хранит сообщение до тех пор, пока принимающее приложение не подключится и не получит его из очереди. Клиент может подтвердить получение сообщения сразу или после того, как полностью обработает его. Как только такое подтверждение получено, сообщение удаляется из очереди. Для сравнения в Kafka очередь сообщений является постоянной — данные хранятся, пока не истечёт указанный период или не будет достигнуто ограничение по размеру. Поэтому важно убедиться, что событие, которое должно произойти один раз, не воспроизводится многократно.

Основное преимущество RabbitMQ — гибкая маршрутизация. Сообщения маршрутизируются через exchange (обменник) перед попаданием в очереди. RabbitMQ предлагает несколько встроенных типов обмена для типичной логики маршрутизации.

RabbitMQ поддерживает приоритезацию в очередях и позволяет настроить диапазон приоритетов. Приоритет каждого сообщения устанавливается при его публикации.

RabbitMQ предлагает простой пользовательский интерфейс управления. Он позволяет контролировать каждый аспект брокера сообщений.



# PostgreSQL в проекте

На сегодняшний день существует множество различных систем управления базами данных - СУБД, от коммерческих до открытых, от реляционных до новомодных NoSQL и подобных им.

Один из лидеров в области СУБД - PostgreSQL и его различные вариации, о которых мы подробно рассмотрим.

В данной статье мы начнем обсуждать СУБД PostgreSQL, рассмотрим различия между редакциями и некоторые особенности архитектуры, а также процесс установки.

Таким образом, схемой мы будем называть логическое объединение таблиц в базе данных, а сама база данных - физическое объединение таблиц. Индекс - отношение, содержащее данные, полученные из таблицы или материализованного представления. Его внутренняя структура обеспечивает быстрый доступ к исходным данным.

Еще одним важным термином является первичный ключ - частный случай ограничения уникальности, определенного для таблицы или другого отношения, который также гарантирует, что все атрибуты в первичном ключе не имеют нулевых значений. Для каждой таблицы может быть только один первичный ключ, хотя возможно иметь несколько уникальных ограничений, которые также не допускают атрибуты со значением null.

И, наконец, транзакция - это комбинация команд, которые должны действовать как единая атомарная операция. Все они либо завершаются успешно, либо неудачно как единое целое, и их результаты не видны другим сеансам до завершения транзакции, и, возможно, даже позже, в зависимости от уровня изоляции. Если хотя бы одна команда внутри транзакции завершится ошибкой, вся транзакция завершится ошибкой.

# Написание кода

## Как писать чистый и красивый код

Что же формирует первое впечатление о коде? Это то, насколько он чист и красив. Такой код - показатель профессионализма человека, который его написал.

Путь к хорошему коду проходит через две важные вехи. Это знания и труд. Знания дают шаблоны, принципы, практические методы и эвристические правила. Однако эти знания необходимо закрепить. Знания должны прочно укорениться в вас благодаря постоянной практике и усилиям.

Короче говоря, научиться писать чистый код — это тяжелый труд. Нужно приложить много усилий. Нужна практика. Шаг за шагом вы приближаетесь к совершенству и повторяете это снова и снова, пока все не станет так, как должно быть. Здесь нет никаких обходных путей.

Вот несколько идей, которые помогут вам овладеть искусством написания чистого, красивого кода.

## Имена

Кендрик Ламар однажды сказал: "Если вы хотите рассказать правдивую историю, начните со своего имени".

В мире программного обеспечения имена буквально повсюду. Функции, классы, аргументы, пакеты - у всего есть имя. Мы называем файлы программного кода, папки и все, что в них находится. Имена часто загрязняют код, потому что мы постоянно придумываем имена.

Имя, которое мы даем сущности, должно раскрывать ее назначение, намерение использовать ее. Выбор хорошего имени требует времени, но больше времени можно сэкономить, когда страсти разгораются. Поэтому обращайте внимание на имя, а когда найдете то, которое лучше справляется со своей задачей, замените им уже используемое имя. Любой, кто читает ваш код, оценит ваше внимание к именам.

Имена переменных, функций и классов должны отвечать на три вопроса о сущности.

Придумывание хороших имен требует не только умения находить нужные слова. Это требует знакомства с культурным контекстом, характерным для программистов всей планеты. Этому нельзя научить. Этому каждый учится сам, например, читая качественный код других людей.

## Одна функция — одна задача

Каждая система построена на основе языка, предназначенного для конкретной предметной области, который спроектирован программистами для точного описания этой области. Функции — глаголы этого языка, а классы — это имена существительные. Функции обычно являются первой линией организации любого языка программирования, и их качественное написание — это суть создания хорошего кода.  
  
Есть всего два правила написания чистых функций:

-Они должны быть маленькими.

-Они должны решать лишь одну задачу и должны делать это хорошо.

Это означает, что функция не должна быть слишком большой, она не должна содержать вложенные структуры. Таким образом, если говорить об отступах при форматировании кода, их не должно быть больше, чем один-два. Такой подход упрощает чтение функций, их понимание и практическое применение. В дополнение к этому, мы, кроме того, должны обеспечить то, чтобы все выражения в функции находились на одном и том же уровне абстракции.  
  
Смешивание уровней абстракции в функции всегда сильно сбивает с толку и ведёт, со временем, к появлению кода, которым невозможно нормально управлять. Опытные программисты видят в функциях нечто вроде историй, которые они рассказывают, нежели код, который они пишут.  
  
Они используют механизмы выбранного ими языка программирования для создания чистых, выразительных, обладающих широкими возможностями блоков кода, которые, и правда, могут быть отличными рассказчиками историй.

## Код и комментарии

Вот интересный комментарий Венус Уильямс: "Все делают собственные комментарии. Так рождаются слухи".

Нет ничего более полезного, чем комментарий, в котором излагается суть дела. С другой стороны, ничто так не загромождает код, как пустые, бесполезные комментарии, которые только занимают место на экране. И нет ничего более разрушительного, чем комментарии, распространяющие дезинформацию и ложь.

Комментарии - это в лучшем случае необходимое зло. Почему? Они не всегда являются злом, но в большинстве случаев это так. Чем старше становится комментарий, тем сложнее поддерживать его в актуальном состоянии. Репутация многих программистов была запятнана тем, что их комментарии перестали соответствовать коду в процессе разработки проекта.

Код меняется и эволюционирует. Блоки кода перемещаются. Но комментарии не меняются. Это уже проблема.

Всегда помните, что аккуратно изложенный код с минимумом комментариев намного лучше сложного программного текста, перегруженного многочисленными пояснениями. Вместо того чтобы тратить время на объяснение созданного вами беспорядка, потратьте время на приведение кода в порядок.

## Важность форматирования

Роберт Мартин однажды очень точно подметил: «Форматирование кода направлено на передачу информации, а передача информации является первоочередной задачей профессионального разработчика».

Я не думаю, что эту мысль стоит недооценивать. Внимание к форматированию кода - одна из важнейших характеристик действительно хорошего программирования.

Отформатированный код - это окно в ваш разум. Мы хотим произвести впечатление на других своей аккуратностью, вниманием к деталям и ясностью мысли. Но если кто-то, читая код, видит беспорядочную структуру без начала и конца, это сразу же бросает тень на репутацию автора кода. В этом нет никаких сомнений.

Если вы думаете, что самое главное для профессионального разработчика - это "заставить программу работать", то вы далеки от истины. Функции, созданные сегодня, могут измениться в следующем выпуске программы, но читаемость кода - это то, что влияет на все, что с ним происходит, с самого начала его существования.

Стиль программирования и читаемость кода продолжают влиять на удобство сопровождения программы даже после того, как ее первоначальный вид изменился до неузнаваемости.

Помните, что именно ваш стиль и аккуратность запомнятся тем, кто будет читать текст вашей программы, и очень редко - функциям, реализованным в коде. Поэтому не забывайте о форматировании, например, придерживайтесь принятых в вашей организации правил.

## Сначала — try-catch-finally, потом — всё остальное

Все программисты имеют дело с обработкой ошибок. Откуда берутся ошибки? В систему поступают ненормальные входные данные или происходит сбой в работе устройства. ...... От разработчиков ожидают, что создаваемые ими программы будут работать правильно. Однако проблема заключается не в самой обработке ошибок. Проблема в том, как обрабатывать ошибки, не нарушая читабельности и чистоты основного кода.

Многие программы перегружают синтаксис обработки ошибок. В результате полезный код оказывается бессистемно разбросанным среди этих конструкций. В результате понять цель программы становится практически невозможно. Это абсолютно неправильно. Код должен быть чистым и надежным, а ошибки должны обрабатываться изящно и элегантно. Такой подход к обработке ошибок - признак программиста, который хорошо знает свою работу.

Один из способов обеспечить качественную обработку ошибок - правильное использование блоков try-catch-finally, которые содержат потенциально опасные места и используют их для организации перехвата и обработки ошибок. Эти блоки можно рассматривать как изолированные области видимости в коде: когда код выполняется в блоке try, он указывает читателю кода, что выполнение может быть прервано в любой момент и продолжено в блоке catch.

Поэтому рекомендуется выделять блок try-catch-finally в самом начале программирования. В частности, блок try-catch-finally помогает читателю кода определить, чего от него можно ожидать, независимо от того, выполняется ли код без ошибок или же фрагмент, заключенный в блок try, не работает.

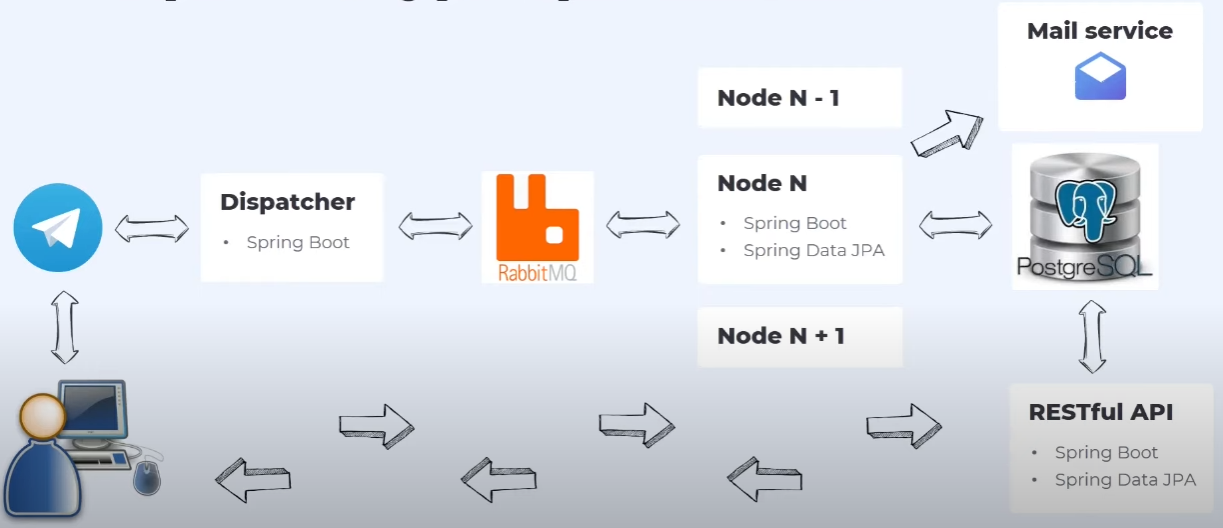
Кроме того, всегда следите за тем, чтобы каждое исключение, выброшенное вашей программой, содержало достаточную контекстную информацию, позволяющую определить источник ошибки и местоположение сгенерированного кода. Конструктивные и информативные сообщения об ошибках будут запоминаться даже после того, как программист перейдет к новой работе.

## Итог как писать чистый код

Как выразить, буквально в двух словах, всё то, о чём мы говорили? Ответ на это вопрос — термин «чувство кода». Это, в мире программирования, эквивалент здравого смысла.  
  
Вот что говорит об этом Роберт Мартин: «Чтобы написать чистый код, необходимо сознательно применять множество приёмов, руководствуясь приобретённым усердным трудом чувством «чистоты». Ключевую роль здесь играет чувство кода. Одни с этим чувством рождаются. Другие работают, чтобы развить его. Это чувство не только позволяет отличить хороший код от плохого, но и демонстрирует стратегию применения наших навыков для преобразования плохого кода в чистый код». По мне — так это золотые слова.  
  
Чувство кода помогает программисту выбрать лучший из имеющихся у него инструментов, использование которого поможет ему в его стремлении создавать чистый и красивый код, приносящий пользу другим людям.  
  
Программист, пишущий чистый код — это художник. Он способен превратить пустой экран в элегантное произведение искусства, которое будут помнить долгие годы.

# О проекте

## Структура проекта:

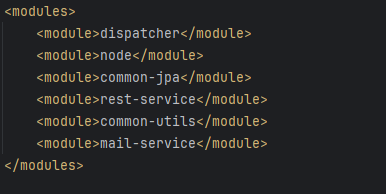


# Выбор технологий и инструментов разработки

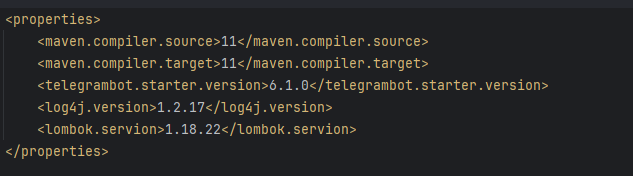
Для создания телеграм-бота будет использован брокер сообщений в виде RabbitMQ, бд была выбрана PostgreSQL, писаться проект будет на языке Java с использованием библиотеки Telegrambots.

## В родительском pom-файле указаны:

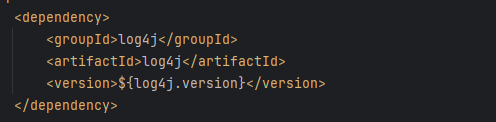
Все модули проекта



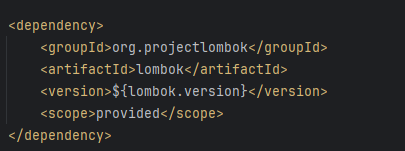
Версии используемых библиотек:



Используемая во всех модулях, зависимость логирования:



Зависимость для удобства написания кода lombok:



Основной модуль Dispatcher разделен на config, controller, utils, service.

В config лежит файл конфигурации RabbitMQ для обработки входящих сообщений.

В папке controller лежит основной файл телеграм-бота, обработчик апдейтов и контроллер вебхуков.

В папке service лежат файлы добавляющие входящие файлы в очереди RabbitMQ.

В utils реализация отправки сообщений.

В модуле common-jpa лежат репозитории и entity.

В модуле common-utils лежит dto файл для отправки email письма и CryptoTool для шифрования.

В модуле mail-service реализована регистрация через email.

В модуле Node реализовано взаимодействие с клиентом, обработка сообщений и обработка exception связанных с загрузкой файлов. В папке test показан один тест на сохранение.

В модуле rest-service реализована активация аккаунта после регистрации и менеджмент файлов.

# Тесты

## Нагрузочное тестирование с JMeter

Apache JMeter — это инструмент для проведения нагрузочного тестирования, представляющий собой десктопное приложение с открытым исходным кодом на базе Java. JMeter позволяет определить, может ли тестируемое веб-приложение удовлетворять требованиям высокой нагрузки или нет. Он также помогает проанализировать работу всего сервера под высокой нагрузкой.

## Тестирование производительности приложения с JMeter

JMeter помогает тестировать как статические, так и динамические ресурсы, помогает обнаружить пользователей, одновременно присутствующих на сайте, и включает визуальные инструменты для предоставления аналитики. Тестирование производительности веб-приложений с JMeter включает в себя нагрузочные тесты и стресс-тесты.

Этот инструмент предлагает следующие преимущества при тестировании производительности:

JMeter можно использовать для тестирования производительности как статических ресурсов, таких как JavaScript и HTML, так и динамических ресурсов, таких как JSP, сервлеты и AJAX.

С помощью JMeter можно определить максимальное количество пользователей в единицу времени, которое может выдержать тестируемый сайт.

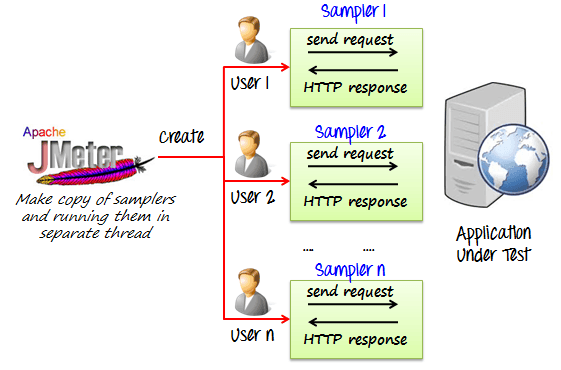
JMeter предоставляет разнообразные визуальные отчеты проводимого анализа производительности.

## Тестирование производительности с JMeter включает в себя:

Нагрузочное тестирование: моделирование ожидаемого использования веб-приложения путем имитации одновременного доступа множества пользователей.

Стресс-тестирование: каждый веб-сервер имеет предел максимальной нагрузки. Когда нагрузка превышает предел, веб-сервер начинает отвечать медленно и выдавать ошибки. Цель стресс-тестирования — определить максимальную нагрузку, которую может выдержать веб-сервер.

На рисунке ниже показано, как JMeter во время нагрузочного тестирования имитирует высокую нагрузку:



# Создание плана тестирования производительности в JMeter

Для начала мы проводим анализ производительности [Google.com](http://google.com/) для 1000 пользователей с использованием различных инструментов тестирования производительности. Наша цель — убедиться, что сайт может справиться с нагрузкой и обеспечить непрерывный и беспроблемный пользовательский опыт.

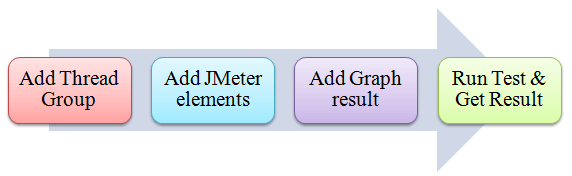
Перед тестированием производительности веб-приложения мы должны определить:

Нормальную нагрузку: среднее количество пользователей, посещающих сайт.

Большую нагрузку: максимальное количество пользователей, посещающих сайт.

Цель проводимого тестирования.

Вот дорожная карта этого практического примера



Performance Testing using Jmeter

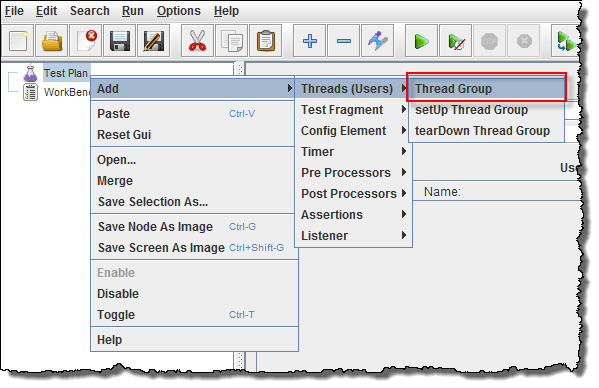
## Шаг 1) Добавить Thread Group (группа потоков)

Запустите JMeter

Выберите Test Plan (План теста)

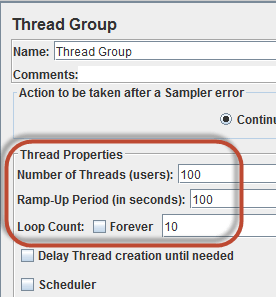
Добавить Thread Group

Щелкните правой кнопкой мыши на "Test Plan" и добавьте новую группу потоков: Add -> Threads (Users) (Потоки (Пользователи)) -> Thread Group



Performance Testing using Jmeter

На панели управления Thread Group введите Thread Properties (Свойства потока) следующим образом:



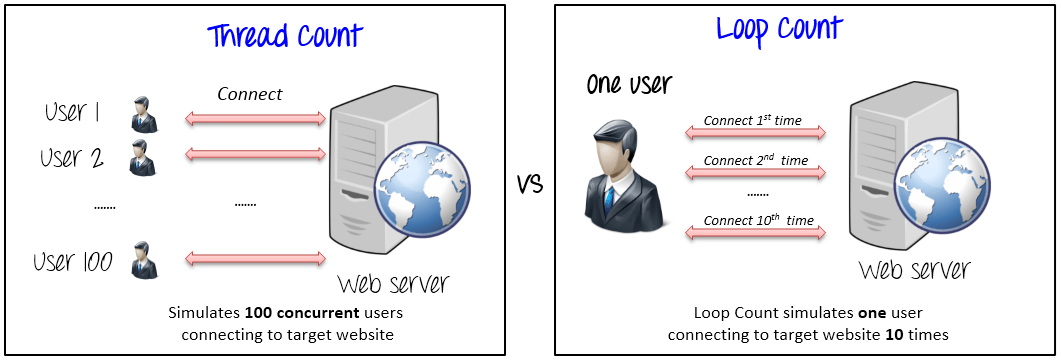
Performance Testing using Jmeter

Number of Threads (Количество потоков): 100 (Количество пользователей, подключенных к веб-сайту: 100)

Loop Count (Количество итераций): 10 (Количество раз выполнения тестирования)

Ramp-Up Period: 100

Thread Count (Счетчик потоков) и Loop Counts (Счетчик итераций) отличаются.



Performance Testing using Jmeter

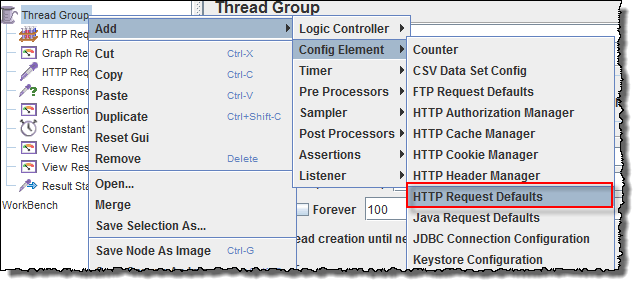
Ramp-Up Period указывает JMeter, какую задержку перед запуском следующего пользователя нужно сделать. Например, если у нас 100 пользователей и период Ramp-Up 100 секунд, то задержка между запуском пользователей составит 1 секунду (100 секунд /100 пользователей).

## Шаг 2) Добавление элементов JMeter

Теперь мы определяем, какие элементы JMeter будут в этом тесте. Этими элементами являются

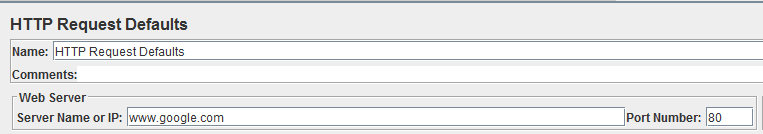
HTTP request Default

Этот элемент можно добавить, щелкнув правой кнопкой мыши по Thread Group и выбрав: Add -> Config Element (Конфигурационные элементы) -> HTTP Request Defaults.



Performance Testing using Jmeter

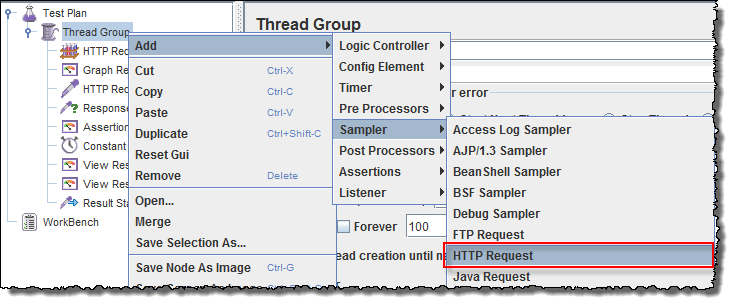
В панели управления HTTP Request Defaults введите адрес тестируемого веб-сайта ([http://www.google.com](http://www.google.com/)).



Performance Testing using Jmeter

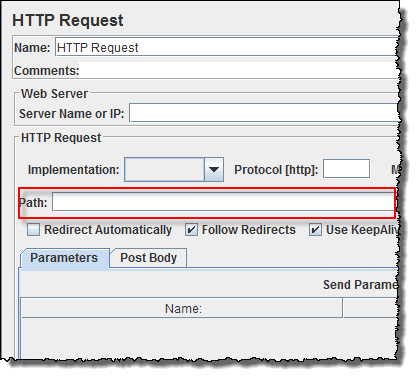
HTTP Request (HTTP-запрос)

Щелкните правой кнопкой мыши на Thread Group и выберите: Add -> Sampler (Сэмплер) -> HTTP Request.



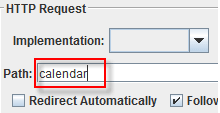
Performance Testing using Jmeter

В панели управления HTTP-запросами (HTTP Request) поле Path указывает, какой URL-запрос (URL request) вы хотите отправить на сервер Google.



Performance Testing using Jmeter

Например, если вы введете "calendar" в поле Path, JMeter создаст URL-запрос <http://www.google.com/calendar> к серверу Google.



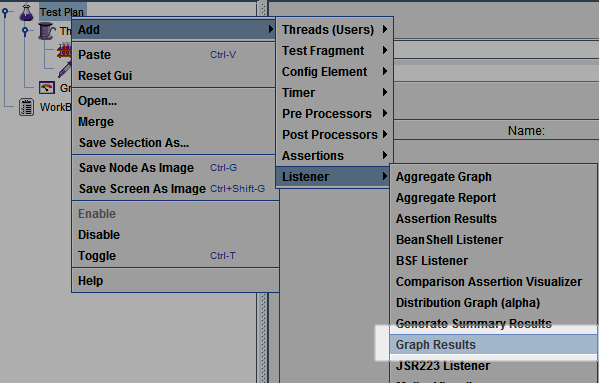
Performance Testing using Jmeter

Если оставить поле Path пустым, JMeter создаст URL-запрос [http://www.google.com](http://www.google.com/) на сервер Google.

В этом тесте оставляем поле Path пустым, чтобы JMeter создал URL-запрос [http://www.google.com](http://www.google.com/) на сервер Google.

## Шаг 3) Добавление Graph Result (Графические результаты)

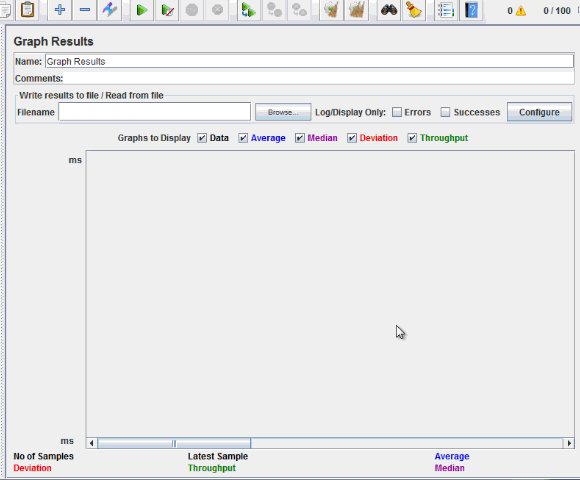
JMeter может показать результат теста в формате графика. Щелкните правой кнопкой мыши на Test Plan (План тестирования), Add (Добавить) -> Listener (Слушатель) -> Graph Results



Performance Testing using Jmeter

## Шаг 4) Запустите тест и получите результат тестирования

Нажмите кнопку Run (Ctrl + R) на панели инструментов, чтобы запустить процесс тестирования. Результат тестирования будет показан на графике в режиме реального времени. На рисунке ниже представлен график плана тестирования, в котором мы смоделировали 100 пользователей, зашедших на сайт [www.google.com](http://www.google.com/).



Performance Testing using Jmeter

В нижней части рисунка приведена следующая статистика, представленная в цветах:

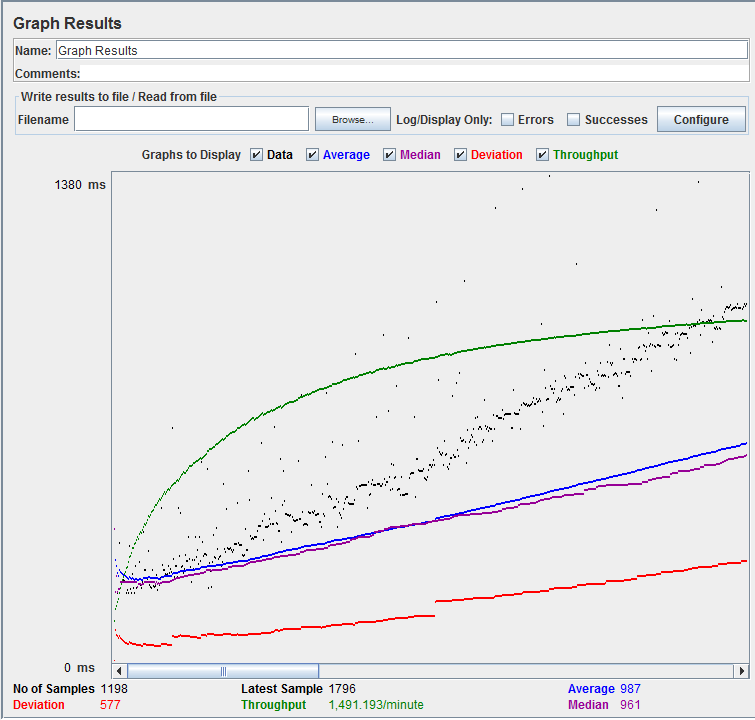
Черный: Общее текущее количество отправленных сэмплов.

Синий: Текущее среднее значение всех отправленных сэмплов.

Красный: Текущее стандартное отклонение (Deviation).

Зеленый: Пропускная способность (Throughput), которая представляет собой количество запросов в минуту, обработанных сервером.

Проанализируем производительность сервера Google на рисунке ниже.



Performance Testing using Jmeter

Чтобы проанализировать производительность тестируемого веб-сервера, необходимо сосредоточиться на 2 параметрах:

Пропускная способность (Throughput)

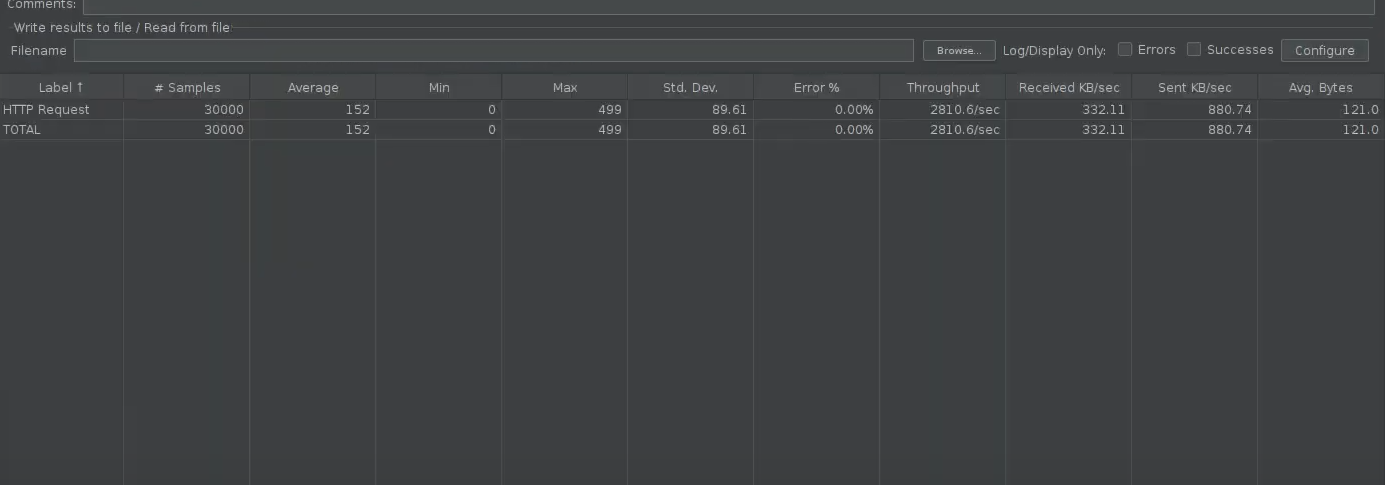
Отклонение/погрешность (Deviation)

Пропускная способность является наиболее важным параметром. Она отражает способность сервера справляться с большой нагрузкой. Чем выше пропускная способность, тем выше производительность сервера.

В данном тесте пропускная способность сервера Google составляет 1 491,193 в минуту. Это означает, что сервер Google может обрабатывать 1 491,193 запросов в минуту. Это довольно высокое значение, поэтому можно сделать вывод, что у сервера Google хорошая производительность.

Отклонение показано красным цветом — оно указывает на отклонение от среднего значения. Чем меньше, тем лучше.

Для telegram-бота характерно будет тестирование на сохранение большого объёма данных (нагрузочное тестирование)



На скрине видно, что не было никаких ошибок.

# Итог

В результате работы над дипломным проектом был создан функциональный телеграмм-бот, способный сохранять текст, документы и изображения. Бот имеет удобный и интуитивно понятный интерфейс, позволяющий пользователям легко сохранять и организовывать свои данные.

Бот обладает следующими функциями:

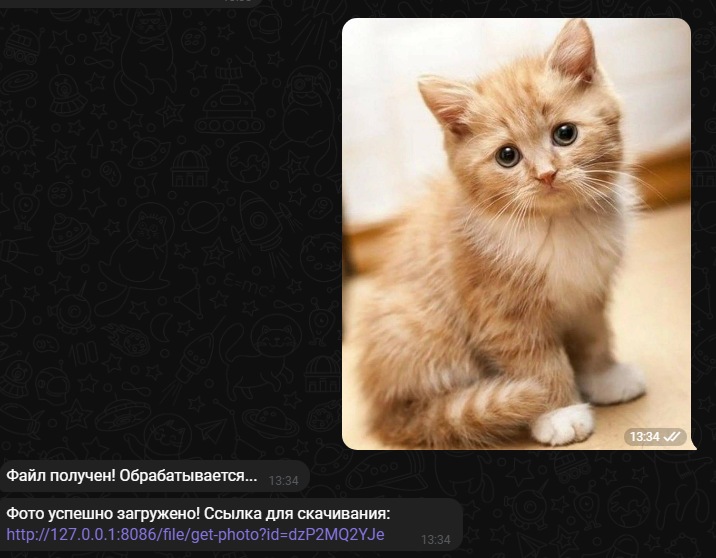
-Возможность сохранения текстовой информации в удобном формате.

-Возможность сохранения документов различных форматов, таких как Word, PDF, Excel и другие.

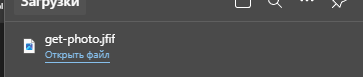
-Возможность сохранения изображений в форматах JPG, PNG и др.

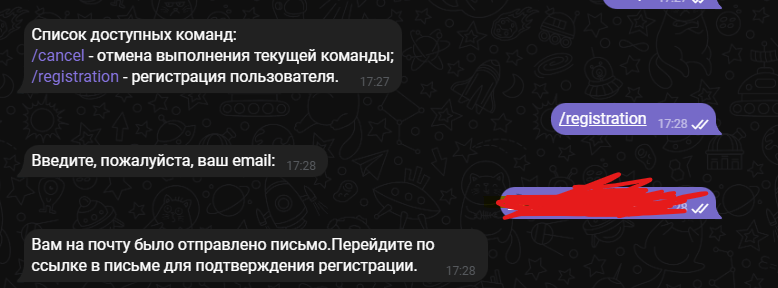
Благодаря созданному боту пользователи могут легко и быстро сохранять свои данные и иметь к ним доступ в любое время и в любом месте через мессенджер Telegram. Этот проект представляет собой удобный инструмент для организации и хранения информации, который может быть использован как в личных целях, так и в рабочих целях.

# Скриншоты результатов работы

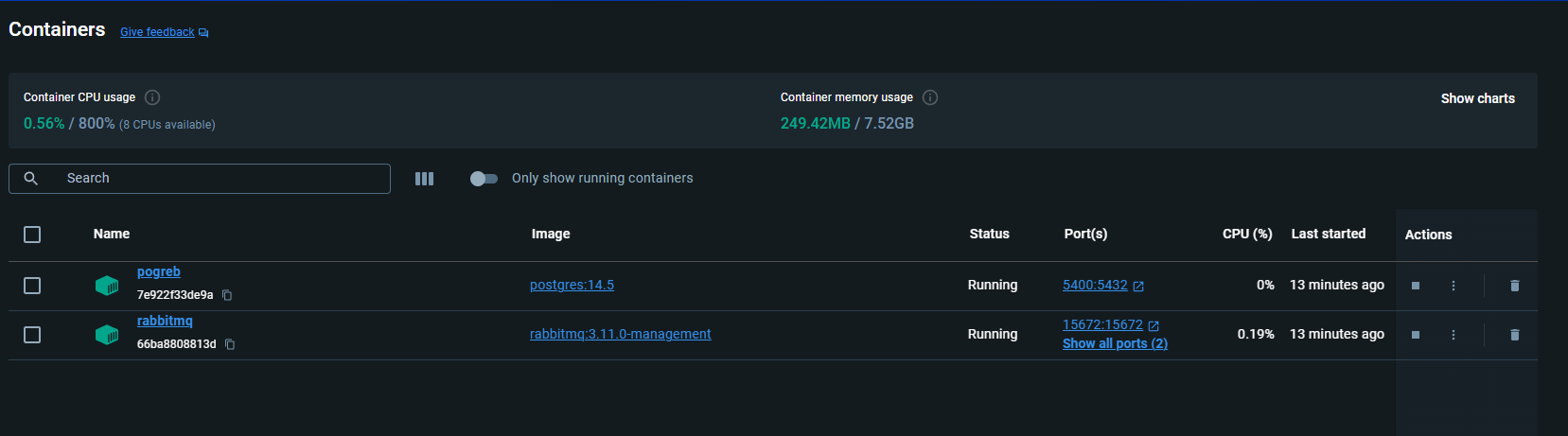


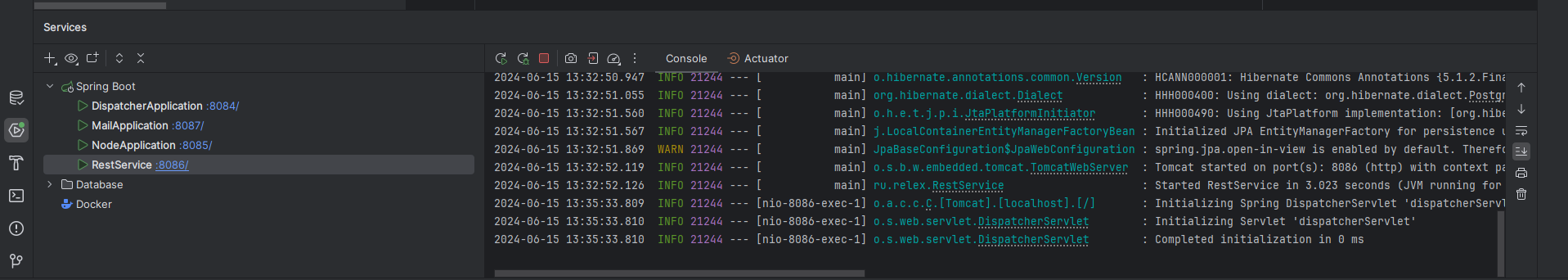
По переходу по ссылке автоматически скачался файл











# Источники

<https://core.telegram.org/bots/api>

[Bots: An introduction for developers (telegram.org)](https://core.telegram.org/bots)

[Всё, о чём должен знать разработчик Телеграм-ботов / Хабр (habr.com)](https://habr.com/ru/articles/543676/)

[Пишем асинхронного Телеграм-бота / Хабр (habr.com)](https://habr.com/ru/companies/kts/articles/598575/)

[Когда и зачем нужен RabbitMQ / Хабр (habr.com)](https://habr.com/ru/companies/slurm/articles/684412/)

[Telegram-бот на вебхуках — Java+Spring, Redis, клавиатуры и деплой на Heroku — снова для самых маленьких / Хабр (habr.com)](https://habr.com/ru/articles/646017/)

[Чем Telegram API отличается от Bot API — хендбук (tmat.me)](https://handbook.tmat.me/ru/dev/api)