МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

АВТОМАТИЗИРОВАННАЯ СИСТЕМА НАГРУЗОЧНОГО ТЕСТИРОВАНИЯ ЧАТ-БОТА

Студент 02.06.23 (подпись, дата) Ходакова Мария Алексеевна

Руководитель 02.06.23 (подпись, дата) Синев М.П.

Нормоконтролёр 02.06.23 (подпись, дата) Бычков А.С.

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой _____ (подпись) М.А. Митрохин (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 14.06.2023 г. № 11)

Секретарь ГЭК _____ (подпись) Е.И. Гурин

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«26» апреля 2023г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Ходакова Мария Алексеевна гр. 19ВВП1 факультета вычислительной техники

2.Тема работы «Автоматизированная система нагрузочного тестирования чат бота»

Тема утверждена приказом ПГУ №458/о от "26" апреля 2023г.

3. Руководитель бакалаврской работы к.н.т Синев М.П.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать «Автоматизиированную систему нагрузочного тестирования чат-бота», которая будет способна обеспечить стабильность, масштабируемость и высокую производительность чат-бота в условиях, когда количество запросов может значительно возрастать, например во время увеличения количества пользователей или отправки большого количества сообщений и запросов. Система должна включать в себя функционал анализа чтения и отправки как личных, так и сообщений в группе. Программа должна позволять пользователю получать полный отчет с информацией о результатах нагрузочного тестирования.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги существующих инструментов нагрузочного тестирования, изучить актуальные способы создания тестовых сценариев и автоматных моделей, изучить способы проектирования и разработки скрипта нагрузки для тестирования сервера, где последний поддерживает *REST API и WebSocket* методологии. Также необходимо изучить функционал фреймворков *Gatling* и *Maven*.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка приложения «Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения»	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023г.

Руководитель бакалаврской работы _____ Синев М.П.
(подпись, дата)

Задание к исполнению принял студент _____ Ходакова М.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 09.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 76 л., 31 рис., 17 источн., 3 прил.

СКРИПТ, АВТОМАТНАЯ МОДЕЛЬ, *REST API*, *API*, *WebSocket*, *scala*, ЧАТ-БОТ.

Цель работы – разработать систему нагрузочного тестирования для чат-бота.

Объектом разработки является система нагрузочного тестирования, которая предназначена для выполнения таких функций, как отправка и чтение сообщений в группах и личных чатах, организация и управление группами. Пользователи взаимодействуют с сервером системы по двум протоколам: *REST API* и *WebSocket*.

В результате работы была разработана структура, сценарии, автоматные модели и написан скрипт для проведения нагрузочного тестирования сервера чат-бота с использованием инструмента *Gatling*. Также было проведено нагрузочное тестирование с использованием разработанной программы.

Содержание

Перечень принятых сокращений и обозначений.....6
Введение.....7
1 Автоматизированная система нагрузочного тестирования чат бота.....8
1.1 Анализ предметной области.....8
1.1.1 Направления тестирования.....8
1.1.3 Нагрузочное тестирование.....14
1.2 Сравнение инструментов нагрузочного тестирования.....14
1.2.1 Анализ JMeter.....15
1.2.2 Анализ Taurus.....16
1.2.3 Анализ K6.....17
1.2.4 Инструмент разработки Gatling.....18
1.3 Архитектура сетевого взаимодействия.....19
1.4 Чат-бот.....20
1.4.1 Нагрузочное тестирование чат-бота.....20
1.4.2 REST API.....21
1.4.3 WebSocket.....25
1.5 Постановка задачи.....26
1.6 Разработка сценариев нагрузки.....27
1.6.1 Сценарии нагрузочного тестирования.....28
1.6.2 Тест сценариев и автоматные модели.....30
1.6.3 Результаты работы программы.....37
1.6.4 Разработка программы.....38
1.7 Разработка сценария в Gatling.....39
1.7.1 Создание HAR-файла.....39
1.7.2 Создание проекта Gatling из архетипа Maven.....44
1.7.3 Начало работы с тестом производительности Gatling.....46
1.7.4 Добавление скрипта Gatling.....47
1.7.5 Анализ результата.....51

2 Охрана труда.....52
2.1 Возможные риски и опасности при проведении нагрузочного тестирования.....52
2.2 Меры по обеспечению безопасности.....53
2.2.1. Организация рабочего места.....53
2.2.2. Использование средств индивидуальной защиты.....53
2.2.3. Предупреждение аварийных ситуаций.....53
2.2.4. Обучение и тренировка персонала.....53
2.2.5. Контроль соблюдения мер безопасности.....54
Заключение.....55
Список используемых источников.....56
Приложение А.1 - UML-диаграмма развертывания.....59
Приложение А.2 - UML-диаграмма сетевого взаимодействия.....60
Приложение А.3 - UML-диаграмма активности отправки сообщений.....61
Приложение А.4 - UML-диаграмма активности чтения сообщений.....62
Приложение Б - Листинг.....63
Приложение В - Презентация.....67

Перечень принятых сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

ПО (программное обеспечение) — программа или множество программ, используемых для управления компьютером

СУБД – система управления базами данных

API(Application Programming Interface) – программный интерфейс приложения;

CRUD (*Create* (создание), *Read* (чтение), *Update* (модификация) и *Delete* (удаление))— акроним, обозначающий четыре базовые функции, используемые при работе с базами данных

HTML (HyperText Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

ID (Identifier) – идентификатор

REST API - (*Representational State Transfer* — «передача репрезентативного состояния» или «передача „самоописываемого“ состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SOAP API - (*Simple Object Access Protocol* — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

WebSocket – протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером, используя постоянное соединение

XML (eXtensible Markup Language) – расширяемый язык разметки

Введение

Современные технологии продолжают быстро развиваться, тем самым выдвигая все новые требования к разработчикам. В связи с этим, важной задачей является обеспечение стабильности и надежности работы программного обеспечения. Нагрузочное тестирование – один из самых важных этапов тестирования программного обеспечения, которое позволяет оценить работоспособность приложения при высокой нагрузке.

Сегодня в современном мире чат-боты являются неотъемлемой частью бизнеса, которые помогают многим компаниям и организациям автоматизировать и облегчить работу в области клиентского сервиса. Вместе с ростом популярности и использования чат-ботов возникает необходимость в их надежности и стабильности при перегрузках.

Цель данной дипломной работы – проектирование и разработка автоматизированной системы нагрузочного тестирования чат-бота. Объектом исследования является чат-бот, используемый в бизнесе для автоматизации работы с клиентами.

В процессе разработки были рассмотрены основные подходы и методы нагрузочного тестирования, а также инструменты, которые используются для автоматизации этого процесса. Данная работа направлена на разработку инструментария, который позволит проводить нагрузочное тестирование чат-бота и оценить его работоспособность при высокой нагрузке.

Работа имеет практическую значимость и может быть использована в качестве руководства для разработки собственной системы нагрузочного тестирования чат-бота.

1 Автоматизированная система нагрузочного тестирования чат бота

1.1 Анализ предметной области

Тестирование ПО - процесс мониторинга качества ПО в рамках создания. Она нужна для того, чтобы обнаружить ошибки, недостатки и некорректное функционирование программы перед выпуском программы. Тестирование позволяет улучшить качество продукта, устранить потенциальные уязвимости и повысить надежность [1].

1.1.1 Направления тестирования

На данный момент существует несколько методов функциональной проверки, позволяющих выявлять закономерности работы определенного объекта, программы или элемента. На рисунке 1 представлен список направлений функциональных тестов.

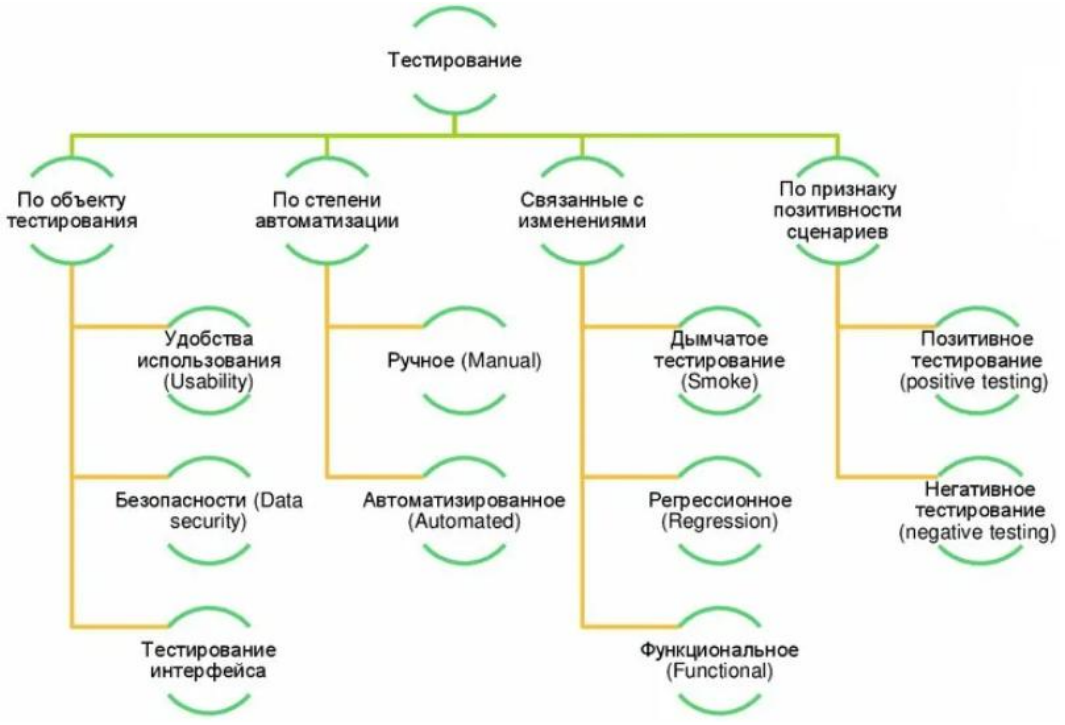


Рисунок 1 – Виды функционального тестирования

Рассмотрим каждое из них:

1.1.1.1 Тестирование удобства использования

Тестирование удобства использования (usability *testing*) – это процесс оценки сайта, приложения или продукта на простоту использования с точки зрения конечного пользователя. Он направлен на изучение того, насколько легко и быстро пользователи могут выполнять задачи в рамках систем, которые они используют. Цель тестирования удобства использования – улучшение качества продукта путем выявления и устранения недостатков в интерфейсе, взаимодействии с пользователем, компоновке информации и других элементах, через которые пользователь взаимодействует с продуктом. Результаты тестирования помогают разработчикам создавать высокопроизводительные и качественные приложения, отвечающие требованиям конечного пользователя.

1.1.1.2 Тестирование безопасности

Тестирование безопасности (тестирование защиты данных) - это процесс проверки уровня защиты системы или приложения от угроз, связанных с конфиденциальностью данных. Основная цель тестирования безопасности - выявление уязвимостей и потенциальных угроз для системы, которые могут привести к раскрытию или краже конфиденциальной информации.

Тестирование безопасности необходимо для обеспечения безопасности и надежности передачи данных между компьютерами, а также для оценки уровня защиты информационной системы в целом. Оно помогает выявить потенциальные проблемы с безопасностью и обеспечивает защиту от внешних угроз, таких как взлом, DDOS-атаки, кража паролей и другие уязвимости.

Тестирование безопасности позволяет разработчикам и тестировщикам выявлять и устранять уязвимости до запуска приложения, что снижает вероятность нарушения конфиденциальности данных, а также повреждения или потери информации. Кроме того, это неотъемлемая часть комплексного

тестирования программного обеспечения, поскольку позволяет гарантировать стабильность и надежность системы при реальных угрозах безопасности.

1.1.1.3 Тестирование интерфейса

Тестирование интерфейса - это процесс проверки пользовательского интерфейса приложения или веб-сайта на соответствие стандартам дизайна и функциональности. Оно помогает проверить, насколько удобен и понятен интерфейс для пользователей, а также выявить ошибки и недочеты, которые могут привести к негативному опыту использования приложения. Цель тестирования интерфейса - убедиться, что пользователи смогут легко найти нужную информацию, выполнить задачи и использовать функциональность приложения, не испытывая затруднений или раздражения.

1.1.1.4 Ручное тестирование

Ручное тестирование - это метод проверки функциональности программного обеспечения, который выполняется вручную. Он включает использование программы в соответствии с ее спецификацией, анализ ее реакций и сравнение их с ожидаемыми результатами. Ручное тестирование может обнаруживать ошибки, такие как ошибки интерфейса, внутренние логические ошибки в программе, возможности улучшения пользовательского опыта и т.д. Одним из основных преимуществ ручного тестирования является возможность создания тестов с точки зрения пользователя.

1.1.1.5 Автоматическое тестирование

Автоматическое тестирование - это процесс проверки программного обеспечения, который выполняется с помощью специальных сценариев и инструментов. Он подразумевает написание специальных программных сценариев, которые проводят систематические проверки функций программы с целью определения наличия ошибок. Автоматическое тестирование может быть полезно для повторяемых задач, таких как функциональное и регрессионное тестирование. Этот метод позволяет сэкономить время и

увеличивает производительность, так как выполнение тестов осуществляется автоматически. Однако, автоматическое тестирование может пропустить то, что было бы обнаружено вручную, так как некоторые аспекты могут быть упущены из-за ограничений программных сценариев.

В зависимости от требований и характеристик программного продукта, тестировщики могут использовать как ручное, так и автоматизированное тестирование. В идеале, оба метода должны использоваться в комбинации, чтобы обеспечить полное покрытие функциональности и выявить ошибки.

1.1.1.6 Дымчатое тестирование

Дымчатое тестирование (*smoke testing*) - это тип тестирования, который выполняется для проверки основных и наиболее важных функций программного продукта. Он проводится перед более подробным и глубоким тестированием с целью определения наличия основных ошибок или проблем в работе программы.

В основном дымчатое тестирование проводится на самых первых этапах разработки продукта, когда еще не готовы все функции, но уже есть основные элементы и возможности программы. Оно позволяет определить, насколько хорошо работает базовый функционал. Если во время дымчатого тестирования тестер обнаруживает какие-либо серьезные проблемы, то разработчикам необходимо исправить их незамедлительно.

Дымчатое тестирование является неотъемлемой частью проверки качества продукта. Оно ускоряет весь процесс разработки, помогает стабилизировать работу программы и повышает ее надежность.

1.1.1.7 Регрессивное тестирование

Регрессивное тестирование является процессом тестирования программного обеспечения, который используется для проверки, что изменения в коде или модели поведения программы не привели к появлению новых ошибок или не повлияли отрицательно на функциональность уже существующих частей программы.

В основе регрессивного тестирования лежит идея воспроизведения уже пройденных тестовых кейсов после каждого внесения изменений в программу. Это позволяет проверить, что новые изменения не привели к возникновению ошибок, которые могут отразиться на работе уже существующего программного кода.

Регрессивное тестирование также позволяет обеспечить стабильность работы программного обеспечения и убедиться в его соответствии требованиям функциональности. Таким образом, это необходимый шаг в процессе тестирования программного обеспечения и помогает в достижении высокой качества конечного продукта.

1.1.1.8 Функциональное тестирование

Функциональное тестирование – это процесс проверки соответствия функциональных требований к программному обеспечению, позволяющий убедиться, что система работает корректно и выполняет свои задачи в соответствии с заявленными требованиями.

Тестирование функциональности относится к тестированию черного ящика (black box *testing*), когда тестировщик не знает внутреннего устройства системы и сосредотачивается на проверке входных и выходных данных, а также поведения системы при различных условиях.

Цель функционального тестирования заключается в обнаружении дефектов в функциональности системы, которые могут привести к неправильной работе программы, некорректным результатам, потере данных или нарушению безопасности. В результате тестирования функциональности разработчики получают информацию о качестве и готовности продукта, что позволяет исправить возможные ошибки.

Функциональное тестирование проводится на всех уровнях тестирования, от модульного до системного, а также при тестировании взаимодействия между различными компонентами приложения. Все тесты функциональности документируются и регулярно обновляются по мере

внесения изменений в систему.

1.1.1.9 Позитивное тестирование

Позитивное тестирование - это процесс проверки программного обеспечения с целью выявления положительных результатов, то есть корректного поведения системы при заданных условиях. Это означает, что тестирование проводится с использованием данных, которые должны привести к успешному выполнению задачи.

Позитивное тестирование обычно используется в качестве подхода к тестированию функциональных требований или отдельных блоков кода, чтобы убедиться, что они выполняются правильно и не возвращают ошибки. Этот тип тестирования может быть применен в начальных стадиях создания программного обеспечения для убеждения в работоспособности и дальнейшей отладке, а также в конечных стадиях разработки для подтверждения соответствия программы заданным требованиям.

Позитивное тестирование - это один из ключевых инструментов в проверке качества программного обеспечения и позволяет выявить ошибки еще до того, как они станут критическими.

1.1.1.10 Негативное тестирование

Негативное тестирование - это процесс проверки программного обеспечения, в ходе которого тестируются его отрицательные сценарии работы (неправильное условие, неправильное значение, невалидные данные и т.д.). Целью негативного тестирования является обнаружение ошибок, сбоев и недочетов в программном продукте, которые могут возникнуть при работе с некорректными входными данными или параметрами. Негативное тестирование позволяет увеличить уверенность в качестве программного обеспечения и уменьшить риски возникновения ошибок при его использовании [2].

Проблемы, которые решает тестирование:

- проверка системы на устойчивость к взлому и другим видам

кибератак;

- проверка совместимости программы с разными операционными системами и конфигурациями;

- обнаружение и устранение ошибок программы;

- проверка возможностей оптимизации работы программы;

- проверка доступности программы для пользователей и

проверка ее правильной работы;

-выявление и устранение ошибок, которые могут привести к неправильной работе программы или краху системы;

-проверка надежности программы, ее работоспособности и способности к восстановлению после сбоев в работе [3].

1.1.3 Нагрузочное тестирование

Нагрузочное тестирование (*load testing*) - это процесс анализа производительности системы или приложения под реальными условиями нагрузки. Оно позволяет определить, как система будет работать при определенном количестве пользователей и запросов, и выявить ее пределы.

Нагрузочное тестирование несколько отличается от тестирования программного обеспечения. При тестировании ПО проверяются функциональность и соответствие требованиям. Но при нагрузочном тестировании основное внимание уделяется производительности системы или приложения в условиях нагрузки.

Тестирование производительности может помочь выявить следующие проблемы: недостаточность ресурсов (например, памяти или процессорного времени), неэффективные запросы к базе данных, проблемы с сетью или с нагрузкой на сервер [3].

1.2 Сравнение инструментов нагрузочного тестирования

Среди инструментов для создания нагрузки можно выделить *Apache JMeter*, *LoadRunner*, *Gatling*, *Tsung* и многие другие. Для сбора данных

используются такие инструменты, как *Graphite, InfluxDB, Elasticsearch, Kibana* и другие. Анализ результатов тестирования можно производить с помощью инструментов, таких как *Grafana, Splunk, Dynatrace* и многих других. Каждый из этих инструментов имеет свои особенности и преимущества, и выбор конкретного инструмента зависит от требований и целей, которые ставит перед собой команда разработчиков. Поэтому далее проведем анализ аналогов инструментов нагрузочного тестирования, разберем достоинства и недостатки.

1.2.1 Анализ *JMeter*

JMeter (*JMeter - Apache JMeter*™ - *Apache* Software Foundation) - это инструмент нагрузочного тестирования с открытым исходным кодом, который поддерживает тестирование веб-приложений и протоколов, включая *HTTP, HTTP*S, *FTP, JDBC, SOAP, JMS и LDAP*. *JMeter* также поддерживает тестирование различных типов приложений, включая веб-службы, *REST API* и *SOAP API*.

Достоинства :

-распространяется под лицензией *Apache* и является бесплатным ПО;

-может быть установлен и запущен на любой операционной системе, в том числе *Windows*, *Linux* и *Mac*;

-предоставляет простой в использовании графический интерфейс, который позволяет создавать и запускать тесты без написания скриптов;

-поддерживает большое количество протоколов и технологий, включая *HTTP*, *FTP*, *SMTP*, *JDBC*, *JMS*, *SOAP*, *REST*, *LDAP* и многие другие;

-имеет обширную коллекцию плагинов, которые позволяют расширить его функциональность и поддержку новых протоколов и технологий.

Недостатки :

-не предоставляет полноценное мониторинга приложения в реальном времени и не может применяться для непрерывного мониторинга производительности;

-настройка *JMeter* может быть сложной задачей, особенно для неопытных пользователей;

-может не соответствовать высоким требованиям к производительности для больших веб-приложений и высоконагруженных систем;

-не может быть масштабирован для распределенного тестирования без использования дополнительных инструментов и настроек;

-имеет относительно небольшое сообщество пользователей и разработчиков, что может ограничить доступность дополнительных плагинов и ресурсов.

1.2.2 Анализ *Taurus*

Taurus (*Taurus - BlazeMeter*) - это инструмент для автоматизации тестирования нагрузки, который используется для управления сценариями тестирования нагрузки и управления процессом выполнения тестов. *BlazeMeter* является облачной платформой для тестирования нагрузки, которая может быть использована вместе с *Taurus* для создания, запуска и мониторинга тестов.

Достоинства :

-имеет легкий и простой в использовании синтаксис, что делает его доступным для людей с любым уровнем опыта;

-поддержка множества языков программирования и форматов конфигурации, что позволяет адаптировать его под любой проект;

-является бесплатным инструментом с открытым исходным кодом;

-может быть использован в сочетании с *BlazeMeter* для создания, запуска и мониторинга тестов;

-поддерживает множество протоколов, таких как *HTTP*, *HTTPS*, *FTP*, *SMTP*, *POP3*, *IMAP*, *LDAP*, *JMS* и другие.

Недостатки :

-имеет некоторые ограничения в сравнении с другими инструментами для тестирования нагрузки;

-не обладает графическим интерфейсом пользователя, что может быть неудобно для новичков;

-требует определенных технических знаний, чтобы настроить и использовать его;

-может быть неэффективным для больших проектов, требующих высокой производительности и расширенных функций.

1.2.3 Анализ *K6*

K6 (*K6 - Performance testing for the modern developer | load Impact*) - это инструмент для производительности и нагрузочного тестирования, который разработан для современных разработчиков. Он предоставляет возможность тестировать веб-приложения и *API* с интуитивно понятным сценариями и простым *API*.

Достоинства :

-имеет простой интерфейс и легко интегрируется с инструментами разработки, такими как Jenkins или *GitLab CI*;

-может использоваться как для небольших, так и для очень сложных тестовых сценариев;

- у *K6* очень хорошая документация, что делает его более доступным для использования;

-даёт возможность пользоваться *JavaScript* для написания тестовых сценариев, что делает его очень гибким и мощным.

Недостатки :

-*K6* не имеет панели управления, что может вызвать некоторые трудности в начале работы;

-для работы с *K6* необходимо знание *JavaScript*, что может быть проблемой для некоторых разработчиков;

-работает с помощью командной строки, и для некоторых пользователей это может быть неудобно.

В целом, *K6* - это мощный инструмент для нагрузочного тестирования,

который удобен для использования и легко масштабируется, но может иметь некоторые ограничения в интерфейсе и требует знания *JavaScript* для использования [4].

1.2.4 Инструмент разработки *Gatling*

В результате изучения существующих инструментов нагрузочного тестирования и анализа их возможностей, было принято решение использовать *Gatling* в качестве основного инструмента для автоматизированного нагрузочного тестирования чат бота.

Gatling - это мощный инструмент для нагрузочного тестирования, который обладает рядом преимуществ перед аналогами. Он имеет открытый исходный код, что позволяет разработчикам дополнительно настраивать его под свои нужды. *Gatling* удобен в использовании, так как он имеет удобный интерфейс и язык скриптования(*scala*), а также обладает высоким уровнем расширяемости.

Одним из важных преимуществ *Gatling* является его масштабируемость. Он способен обрабатывать большое количество запросов в секунду, что позволяет убедиться в том, что *API* сервер способен выдерживать высокие нагрузки и работать без сбоев при большом потоке пользователей.

Также *Gatling* позволяет проводить различные виды тестирования: нагрузочное, стрессовое, производительности и др. Благодаря этому, можно получить подробные результаты о производительности и надежности *API* сервера, что позволяет устранять проблемы и улучшать его работу [5,6].

Таким образом, использование *Gatling* в качестве инструмента для автоматизированного нагрузочного тестирования чат бота дает возможность получить достоверные и полные данные об производительности и надежности *API* сервера. так же он изначально разрабатывался для протокола *WebSocket* и более точно работает с данным протоколом обмена сообщений и более производительный, чем *JMeter*, но в то же время сложнее в освоении и изучении.

1.3 Архитектура сетевого взаимодействия

Рассматриваемая схема является стандартной архитектурой для использования чата до 1000 пользователей.

Общая структура решения показана на следующей диаграмме (рисунок 2).

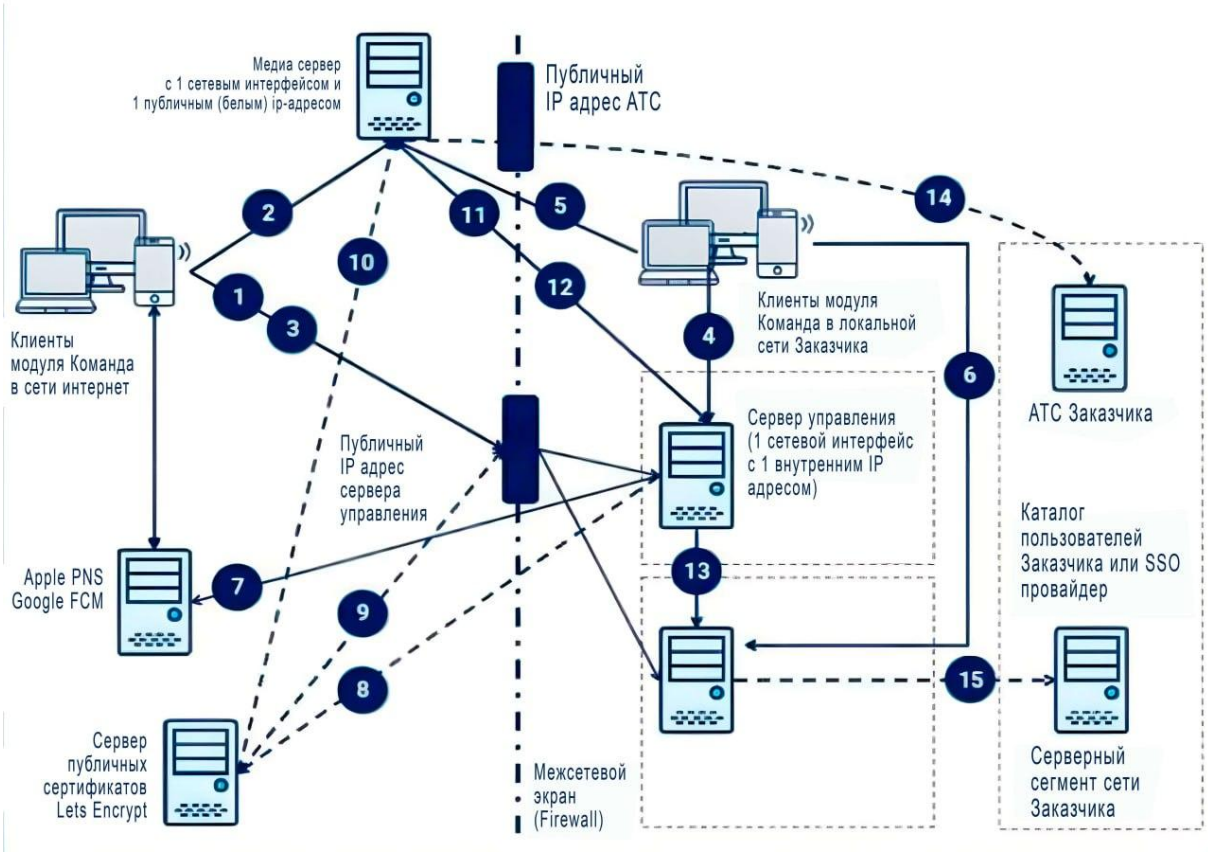


Рисунок 2 - UML-диаграмма сетевого взаимодействия

Описание архитектуры сетевого взаимодействия:

Для работы отправки и чтении сообщений в чат-боте необходимы отдельные публичный и внутренний *IP*-адреса. Если модуль чата опубликован в интернете, пользователи из локальной сети администратора подключаются к публичному *IP*-адресу медиа-сервера (привязанному к порту сервера), установленному администратором чат-бота.

Так же необходимо использовать *SSL* сертификаты. Если организация использует сертификаты, подписанные собственным корневым центром

сертификации, требуется выполнить дополнительную настройку сред исполнения и браузеров, а также установить корневые сертификаты в операционную систему на ПК и мобильные устройства.

Серверы должны быть идентифицированы по доменным именам, указанным в сертификатах.

Для взаимодействия с серверами, в том числе для установки ПО и обмена данными между серверами, необходим канал связи не менее 100 Мбит/с [7].

1.4 Чат-бот

Чат-бот - это программа, способная автоматически отвечать на сообщения в чате на основе заданных алгоритмов и искусственного интеллекта. Он может использоваться для автоматизации общения с клиентами, ответов на часто задаваемые вопросы, организации опросов и многих других задач.

Чат-боты могут быть полезны для бизнеса, поскольку они позволяют автоматизировать рутинные задачи, сократить время ответа на сообщения и улучшить качество обслуживания клиентов. Они также могут помочь в сборе данных и анализе поведения клиентов, а также увеличить число продаж.

Одной из главных проблем, которую решает чат-бот, является устранение необходимости человеческого участия в общении с клиентами. Это позволяет снизить затраты на обслуживание клиентов и повысить качество общения. Также чат-боты способны работать без прерывно, что позволяет быстро и эффективно отвечать на сообщения в любое время дня и ночи [8].

1.4.1 Нагрузочное тестирование чат-бота

Нагрузочное тестирование чат-бота - это процесс, который помогает проверить, как система будет работать в условиях значительной нагрузки. В данном случае под нагрузкой понимается большое количество

одновременных запросов от пользователей.

Нагрузочное тестирование сервера чат бота необходимо, чтобы убедиться, что система сможет обрабатывать большое количество запросов от пользователей и не потерпит сбоя во время использования. Если система не прошла нагрузочное тестирование, то ее производительность может снизиться, а пользователи могут испытать проблемы при общении с чат-ботом.

Одним из распространенных подходов к нагрузочному тестированию является использование инструмента *Gatling*, который позволяет создавать мощные сценарии для тестирования производительности и скорости ответа сервера.

Путем проведения нагрузочного тестирования сервера чат-бота можно выявить проблемы, такие как неправильная конфигурация серверов, недостаточная память и процессор, неправильно настроенный код или проблемы с сетевой инфраструктурой.

В процессе выполнения дипломной работы необходимо будет провести анализ команд и профиля нагрузки сервера, на котором клиенты работают с сервером по двум протоколам *REST API* и *WebSocket*, что поможет определить оптимальные настройки для тестирования. После этого будет разработан скрипт нагрузки, который должен учитывать все возможные сценарии использования чат-бота.

Таким образом, проведение нагрузочного тестирования сервера чат-бота позволяет убедиться, что система работает корректно в условиях значительной нагрузки, избежать сбоев и улучшить общее качество обслуживания пользователей [9].

1.4.2 *REST API*

REST (Representational State Transfer), что расшифровывается как "Передача состояния представления", был придуман Роем Филдингом, который также является одним из создателей протокола *HTTP*. *REST*-сервисы

отличаются тем, что наилучшим образом используют возможности протокола *HTTP* (см. рисунок 3).

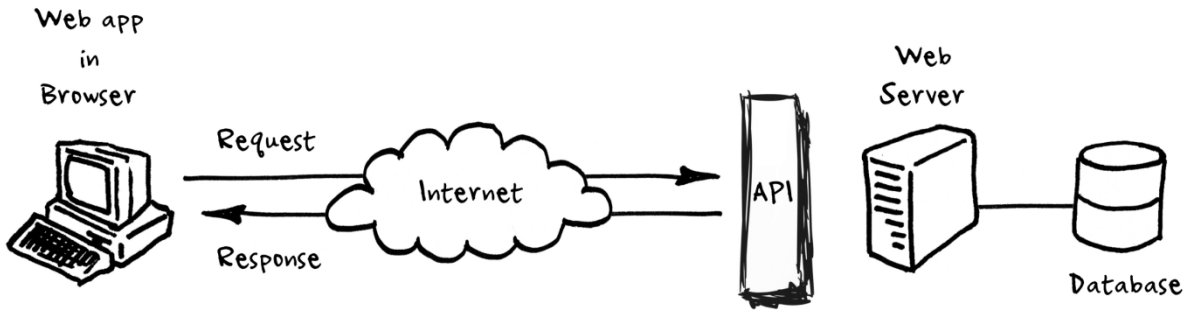


Рисунок 3 – *REST* архитектура.

Рассмотрим данный стиль архитектуры, который имеет особенности характеристики и определенные требования. Рой Фолдинг предложил шесть условий, которые должны быть учтены при проектировании архитектуры в данном стиле.

Концепция клиент-серверной модели заключается в том, что сервер должен скрыть как можно больше деталей своей реализации от клиента. Клиентский интерфейс не должен знать о том, какая СУБД используется на сервере или сколько серверов заняты обработкой запросов и подобных вещей. Использование такого подхода разделяет потребности клиента и сервера, что обеспечивает переносимость кода клиентского интерфейса на другие платформы. Кроме того, упрощение серверной части ведет к улучшению возможностей масштабирования.

Сервер не должен сохранять информацию о клиенте и использовать ее в дальнейшей работе. Зато клиент должен сам отслеживать свою контекстную информацию. Независимость от состояния означает, что результаты вызовов *API* не должны зависеть от предыдущих вызовов.

Возможно кэширование ответа сервера на определенный период времени для повторного использования без запросов к серверу. Если

кэширование используется правильно, оно может уменьшить или даже полностью исключить некоторые клиент-серверные взаимодействия, увеличивая производительность и масштабируемость системы.

Необходимость в единообразии интерфейса является основным требованием при разработке сервисов *REST*. Единообразный интерфейс обеспечивает независимое развитие каждого из сервисов.

Из-за иерархической структуры сетей, клиенты часто не могут точно определить, взаимодействуют ли они напрямую с сервером или через промежуточный узел. Использование промежуточных серверов позволяет повысить масштабируемость путем балансировки нагрузки и распределенного кэширования, а также в соответствии с политикой безопасности, гарантирует конфиденциальность информации.

Код по требованию *REST* позволяет расширить возможности клиента, загружая апплеты или сценарии с сервера. Кроме того, Рой Филдинг описал также свойства, присущие архитектуре *REST*:

-производительность, результативность и эффективность сети, с точки зрения пользователя, могут зависеть от того, как компоненты системы взаимодействуют между собой;

-для обеспечения возможности работы с большим количеством компонентов и их взаимодействий необходима масштабируемость.

Рой Филдинг — один из основных авторов спецификации протокола *HTTP* отмечает, что архитектура *REST* имеет огромное влияние на возможности масштабирования:

-простота унифицированного интерфейса;

-готовность компонентов к принятию изменений для удовлетворения меняющихся потребностей, даже в случае работающего приложения;

-сервисные службы имеют доступ к прозрачной информации обо всех связях между компонентами системы;

-компоненты системы могут быть перемещены вместе с программным

кодом и данными, обеспечивая переносимость;

- надежность системы проявляется в ее стабильности при отказах отдельных компонентов, соединений или данных.

REST представляет собой определенный архитектурный стиль, который воплощается в *RESTful API*. Чем более приложение соответствует принципам *REST*, тем более разработанным оно является. Операции, выполняемые через *RESTful API*, включают получение данных клиентом в удобном для него формате, их изменение, удаление и добавление;

Так же *REST* функционирует на базе протокола *HTTP* и включает в себя операции создания, обновления и удаления данных. Каждая из этих операций представлена соответствующим *HTTP*-методом, таким как *GET* для получения объектов, *POST* для создания, редактирования или удаления объектов, *PUT* для создания или изменения объектов, и *DELETE* для удаления объектов. Все эти методы вместе называются *CRUD* (*create, read, update, delete*) операциями.

REST предлагает универсальный интерфейс для запросов, который не вызывает противоречий, и позволяет взаимодействовать, например, с базами данных. Это одно из главных преимуществ *REST*. Для объяснения связи между *HTTP*-методами, операциями *SQL* и концепцией *CRUD* используется рисунок 4.

Каждый *HTTP* запрос имеет свой собственный статус, необходимый для правильного формирования ответа и передачи его клиенту в соответствии с *REST API*. Коды статуса определены в разделе *10 RFC 2616*:

-100-199 - информационные статусы;

-200-299 - статусы успешных операций;

-300-399 - статусы перенаправления;

-400-499 - статусы ошибок на стороне клиента;

-500-599 - статусы ошибок на стороне сервера.

Рисунок 4 – Соответствие *HTTP* методов *SQL* операциям и концепции *CRUD*.

Таким образом, *REST API* предназначен для создания четко структурированного подхода в разработке прикладных интерфейсов [10].

1.4.3 *WebSocket*

WebSocket API - это передовая технология, которая позволяет установить двунаправленную интерактивную коммуникацию между браузером пользователя и сервером. С помощью этого *API* можно отправлять сообщения на сервер и получать управляемые ответы без явного запроса ответа от сервера. Представьте, что вы встречаете своего друга или коллегу и начинаете с рукопожатия и приветствия, прежде чем перейти к разговору. В компьютерных науках процесс рукопожатия очень важен для синхронизации сервера с его клиентами и является фундаментальным концептом протокола *WebSocket*. Следующая диаграмма иллюстрирует процесс рукопожатия между сервером и различными клиентами (рисунок 5).

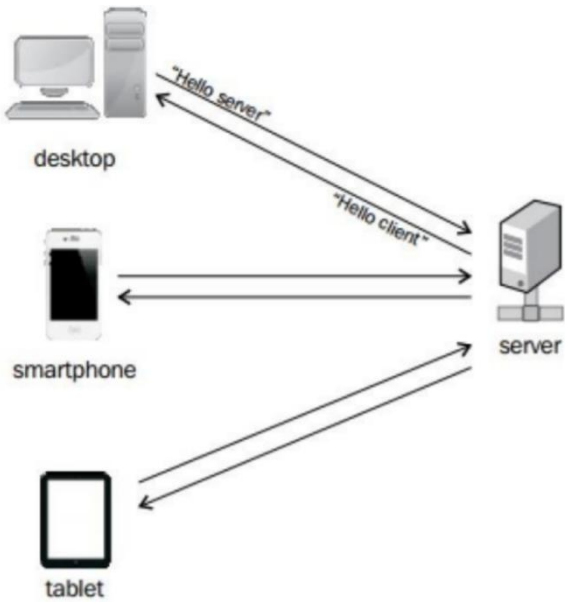


Рисунок 5 – Рукопожатие сервера с клиентами.

Веб-сокеты представляют собой новый шаг в развитии функциональности веб-браузеров в области настольных приложений. Они являются эволюцией, которую долго ждали в технологии клиент-серверных веб-приложений.

Основные особенности веб-сокетов включают стандартизированный протокол, который позволяет связываться между веб-серверами и клиентами в режиме реального времени. Веб-сокеты также представляют кроссплатформенный стандарт для обмена данными между клиентом и сервером в режиме реального времени, что создает новые возможности для предприятий, работающих в режиме реального времени.

Одним из основных преимуществ веб-сокетов является двусторонняя связь (полный дуплекс) по одному *TCP*-соединению. Это добавляет альтернативу взаимодействия сервера и клиента, привнося новые возможности для создания веб-приложений в режиме реального времени.

Основным преимуществом *Web Socket* является возможность обмена данными в обоих направлениях (полный дуплекс) через одно соединение *TCP* [11].

1.5 Постановка задачи

Цель - разработать систему нагрузочного тестирования для чат-бота. Задача заключается в создании системы, которая будет способна обеспечить стабильность, масштабируемость и высокую производительность чат-бота в условиях, когда количество запросов может значительно возрастать, например во время увеличения количества пользователей или отправки большого количества сообщений и запросов.

Кроме того, система нагрузочного тестирования должна быть способна выявлять узкие места, проблемы в производительности и другие технические проблемы, которые могут привести к отказу системы. Разработка подобной системы является сложной и ответственной задачей, требующей высокой

квалификации разработчиков.

Одной из главных проблем, связанных с разработкой системы нагрузочного тестирования для чат-бота, является необходимость в выборе правильного инструмента для тестирования. Кроме того, необходимо проводить тестирование на разных уровнях – от малых объемов данных до больших нагрузок, что необходимо для оценки стабильности, переносимости и масштабируемости системы.

Также одной из проблем является необходимость в проведении комплексного тестирования, включающего в себя не только тестирование общей производительности, но и отдельных функций и компонентов, которые могут оказывать влияние на работу чат-бота.

Цель - составление сценариев нагрузки для тестирования сервера на возможность работы с большим числом пользователей при чтении и отправке сообщений и проверки производительность сервера при одновременном чтении и отправке сообщений.

В каждом сценарии должно быть учтено время ответа сервера на запросы, а также количество запросов, которые сервер сможет обработать в единицу времени.

Полученные результаты тестирования должны быть анализированы с целью выявления проблемных областей для дальнейшей оптимизации сервера, если это необходимо.

Таким образом, задача разработки системы нагрузочного тестирования для чат-бота является сложной и ответственной задачей, требующей высокой квалификации и опыта в сфере тестирования и разработки. Однако, проведение такого тестирования является необходимым для обеспечения качественной и надежной работы чат-бота в условиях повышенной нагрузки.

1.6 Разработка сценариев нагрузки

Основные сценарии нагрузки для чат-бота:

-нагрузочное тестирование на максимальное количество одновременных пользователей - определяется максимальное количество пользователей, одновременно общающихся с ботом, при котором бот продолжает работать без задержек и сбоев;

-тестирование на длительную работу - проверка способности бота продолжать работу без сбоев в течение длительного времени, например, несколько часов или дней;

-тестирование на различные типы запросов - проверка возможности обработки ботом большого количества запросов различных типов, таких как текстовые сообщения, голосовые сообщения и файлы;

-тестирование на различные языки - проверка возможности бота работать на различных языках и обрабатывать запросы на разных языках;

-тестирование на различные платформы - проверка возможности бота работать на различных платформах, таких как *Facebook, Telegram, Viber* и т.д [12].

1.6.1 Сценарии нагрузочного тестирования

В *Gatling* существуют две основные сущности для создания сценариев нагрузочного тестирования - сценарий (*Scenario*) и симуляция (*Simulation*).

Сценарий (*Scenario*) описывает действия пользователя, которые он должен выполнить во время нагрузочного тестирования. Например, это может быть авторизация на сайте, заполнение формы, выбор товара и т.д.

Симуляция (*Simulation*) описывает сценарии, которые будут использоваться в тесте, количество пользователей, которые будут использовать эти сценарии и длительность тестирования.

Для создания сценариев в *Gatling* используется *DSL*-язык на *scala*. Он позволяет описывать действия пользователя в формате цепочек (*chain*). Например:

```
val scn = scenario("Scenario name")
    .exec(http("Request name")
      .get("http://example.com")
```

```
      .check(status.is(200)))
```

В примере выше мы создали сценарий с названием "*Scenario name*", который выполняет *GET*-запрос на адрес "*HTTP://example.com*" и проверяет, что ответ имеет статус 200.

Можно также использовать функции и условия для создания более сложных сценариев. Например:

```
val scn = scenario("Scenario name")
    .exec(http("Request 1")
      .get("http://example.com"))
    .pause(5)
    .doIf(session => session("variable").as [Int] == 1) {//
условие выполнится, если значение переменной "variable" = 1
      exec(http("Request 2")
        .get("http://example.com"))
    }
```

В примере выше мы создали сценарий, который выполняет два *GET*-запроса на "*HTTP://example.com*". Между ними мы добавили паузу в 5 секунд. Затем мы проверяем значение переменной "variable" в сессии пользователя. Если она равна 1, то выполняем второй запрос.

Для создания симуляции мы используем класс *Simulation* и определяем в нем необходимые нам параметры. Например:

```
class MySimulation extends Simulation {
  val scn = scenario("Scenario name")
    .exec(http("Request name")
      .get("http://example.com"))
  setUp(scn.inject(atOnceusers(10))).protocols(http)
}
```

В примере выше мы создали класс *MySimulation*, который наследуется от класса *Simulation*. Здесь мы создали сценарий с названием *"Scenario name"*, который выполнит один *GET*-запрос на "*HTTP://example.com*". Затем мы определили количество пользователей, которые будут использовать этот сценарий (10) и протокол (*HTTP*).

Для запуска симуляции используется команда *"sbt gatling:test"* [13].

1.6.2 Тест сценариев и автоматные модели

Для каждого сценария созданы автоматные модели. Поскольку тестовый сценарий представляет собой последовательность действий, каждый шаг будет соответствовать определенному состоянию в автомате.

Сценарий отправки сообщения пользователю через сокет:

-начальное состояние (*S0*);

-генерация рандомной строки в *localID* (*S1*);

-отправка запроса *sendtyping* с параметрами *roomID*, *userID*, *type* равными *roomId*, *userId* и 1 (*S2*);

-пауза 500 миллисекунд (*S3*);

-отправка запроса *sendtyping* с параметрами *roomID*, *userID*, *type* равными *roomId*, *userId* и 0 (*S4*);

-отправка запроса draft с параметрами *roomID* равным *roomID*, и *text* равным *null* (*S5*);

-генерация рандомной строки длиной 32 символа и сохранение в *localID* (*S6*);

-отправка запроса *sendMessage* с параметрами *localID*, *message*, *roomID*, *userID*, *type*, *attributes*, *user*, *_id*, *created*, *isSending* и *direction* равными соответствующим значениям (*S7*).

Автоматная модель сценария отправки сообщения пользователю через сокет представлен на рисунке 6.

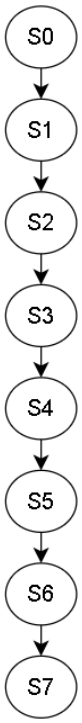


Рисунок 6 - Автоматная модель отправки сообщения пользователю

Сценарий отправки сообщения в группу через сокет:

-начальное состояние (*S0*);

-генерация рандомной строки в *localID* (*S1*);

-отправка запроса *sendtyping* с параметрами *roomID*, *userID*, *type* равными *3-targetGroup*, *userId* и 1 (*S2*);

-отправка запроса *sendtyping* с параметрами *roomID*, *userID*, *type* равными *3-targetGroup*, *userId* и 0 (*S3*);

-отправка запроса draft с параметрами *roomID* равным *3-targetGroup*, и *text* равным *null* (*S4*);

-генерация рандомной строки длиной 32 символа и сохранение в *localID* (*S5*);

-отправка запроса *sendMessage* с параметрами *localID*, *message*, *roomID*, *userID*, *type*, *attributes*, *user*, *_id*, *created*, *isSending* и *direction* равными соответствующим значениям (*S6*);

-пауза 500 миллисекунд (*S7*).

Автоматная модель сценария отправки сообщения в группу через сокет представлен на рисунке 7.

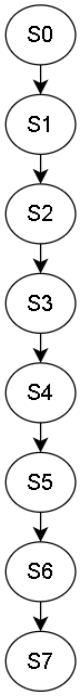


Рисунок 7 - Автоматная модель отправки сообщения в группу

Сценарий чтения сообщений:

-начальное состояние (*S0*);

-отправка запроса на получение списка новых сообщений через *HTTP* (*S1*);

-сохранение последнего идентификатора сообщения (*S2*) ;

-отправка запроса *read_message* с параметрами *operationId* и *maxmessageId* равными соответствующим значениям (*S3*);

-отправка запроса *read_message* с параметрами *roomID*, *maxmessageId*, *unreadCount* и *unreadMentionsCount* равными соответствующим значениям (*S4*) [14].

Автоматная модель сценария чтения сообщений представлен на рисунке 8.

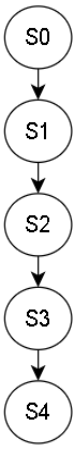


Рисунок 8 - Автоматная модель чтения сообщений

1.6.2.1 Скрипт программы

Листинг:

```
object sendMessage {

//? Генерация рандомной строки в localID
    val random = new Random()
val                              alp                                  =
"1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM"

//? Отправка сообщения по сокету
    val sendmessageuser = repeat(5) {
      exec(ws("WSS sendtyping 1")
        .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"${roomID}","userID":"${userID}","type":
1}]"""))
        .pause(500 milliseconds)
        .exec(ws("WSS sendtyping 0")
          .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"${roomID}","userID":"${userID}","type":
0}]"""))
        .exec(ws("WSS draft")
          .sendText("""42/spikaenterprise,
["draft",{"roomID":"${roomID}","text":null}]"""))
```

```
        .exec(session => {
          session.set("localID",
Iterator.continually(random.nextInt(alp.size)).map(alp).take(32)
.mkString)
        })
        .exec(ws("WSS sendmessage")
          .sendText("""42/spikaenterprise,
["sendmessage",{"localID":"_${localID}}","message":"0301b22f5c3a
448ca986fbc6b6ff097aef824e85d34dbf013cc834c52328613136e7adbf1c2a
85fe661d6b79135ec330a57cf1970535e6040ccb30f60b7f5b762626c3a30eaa
2c6f5b8bffb16e77a8734435","roomID":"${roomID}","userID":"${userI
D}","type":1,"attributes":{"useclient":"web"},"user":${userData}
,"_id":"_${localID}","created":1568970116719,"isSending":true,"d
irection":2}]""")
 )
        .pause(500 milliseconds)
    }
val sendmessageGrup = repeat(5) {
      exec(ws("WSS sendtyping 1")
        .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"3-
${targetGroup}","userID":"${userID}","type":1}]"""))
        //.pause(500 milliseconds)
        .exec(ws("WSS sendtyping 0")
          .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"3-
${targetGroup}","userID":"${userID}","type":0}]"""))
        .exec(ws("WSS draft")
          .sendText("""42/spikaenterprise,
["draft",{"roomID":"3-${targetGroup}","text":null}]"""))
        .exec(session => {
          session.set("localID",
Iterator.continually(random.nextInt(alp.size)).map(alp).take(32)
.mkString)
        })
```

```
          .exec(ws("WSS sendmessage")
          .sendText("""42/spikaenterprise,
["sendmessage",{"localID":"_${localID}","message":"0301cf8f1599c
f1a39fd90cd8ddda6f55af6853ab7e2b7986594f7c9334c6fd62600429f6c1ed
6ff385dddc1d495b08e08cccdd0bef865d06458d2a5d39f1aeb004f1a4de3d2d
0d5d9f3e28e1584c2ba8286","roomID":"3-
${targetGroup}","userID":"${userID}","type":1,"attributes":{"use
client":"web"},"user":${userData},"_id":"_${localID}","created":
1569317150863,"isSending":true,"direction":2}]""")
        ).pause(500 milliseconds)
 }
  }

   object readmessage {
    val   readmessageGrup   =   exec(http("/api/v2/message/list/3-
{targetGroup}/0/new")
      .get("/api/v2/message/list/3-${targetGroup}/0/new")
      .headers(headers_2)
      .queryParam("_", "1677495278092")
      .check(jsonPath("$.data.messages                          [-
1:]._id").saveAs("maxmessageId"))
    )
      .exec(ws("WSS read_message")
        .sendText("""42/spikaenterprise,
["read_message",{"operationId":1676897708185,"maxmessageId":"${m
axmessageId}"}]""")
      )
      .exec(ws("WSS read_message")
        .sendText("""42/spikaenterprise,
["read_message",{"roomID":"3-
${targetGroup}","maxmessageId":"${maxmessageId}","unreadCount":0
,"unreadMentionsCount":0}]""")
      )
  }
```

1.6.2.2 Методы скрипта

Для создания скриптов были использованы следующие методы:

- *repeat()* - цикл, выполняющий указанное количество раз набор действий внутри себя;

-*exec()* - выполнение действия;

-*session.set()* - установка значения в сессионную переменную;

-*http()* - отправка *HTTP*-запроса;

-*get()* - метод *HTTP*-запроса для получения данных;

-*queryParam()* - установка значения параметра запроса;

-*check()* - проверка значения ответа на запрос;

-*JSONPath()* - выборка значения из *JSON*-ответа на запрос;

-*ws()* - создание *WebSocket*-соединения для обмена данными в режиме реального времени;

-*sendText()* - отправка текстового сообщения через *WebSocket*-соединение.

1.6.2.3 Используемые команды

Код написан на языке *Scala* и использует библиотеку *Gatling* для тестирования чат-бота.

В объекте *sendMessage* вызываются следующие команды:

- send*messageuser* - отправка сообщения в личные чаты. Параметры запроса включают *roomID* (идентификатор комнаты), *userId* (идентификатор пользователя) и *userData* (данные пользователя). Ответ от сервера не обрабатывается в данном коде;

- send*messageGrup* - отправка сообщения в групповые чаты. Параметры запроса включают *tarGETGroup* (идентификатор целевой группы), *userId* (идентификатор пользователя) и *userData* (данные пользователя). Ответ от сервера не обрабатывается в данном коде.

В объекте *readmessage* вызывается команда *readmessageGrup. Эта команда отвечает за* чтение сообщений из группового чата. Параметры

запроса включают *tarGETGroup* (идентификатор целевой группы) и заголовки (*(headers_2*). Ответ от сервера обрабатывается в виде извлечения максимального идентификатора сообщения (*maxmessageId*), который затем передается в запросе на прочтение сообщений через *WebSocket* [15].

1.6.3 Результаты работы программы

Создание программы для нагрузочного тестирования состоит из нескольких этапов.

1.6.3.1 Определение требований к программе нагрузочного тестирования

Первым шагом необходимо определить, какие задачи должна выполнять программа нагрузочного тестирования чат-бота. В частности, нужно определить:

-какое количество запросов и какой тип запросов должны быть отправлены в чат-бот;

-какое количество пользователей и какое количество запросов каждый из них должен отправить;

-какой верхний предел нагрузки может выдержать чат-бот;

-какая частота запросов должна быть установлена и какой должен быть интервал между запросами;

-какие метрики необходимо учитывать при проведении тестирования и какой должен быть пороговый уровень для каждой метрики.

1.6.3.2 Автоматная модель

Затем создать модель автомата, которая будет соответствовать поведению пользователя.

1.6.3.3 Создание скриптов

Следующим шагом в процессе создания программы нагрузочного тестирования является создание скриптов, которые будут использоваться для тестирования. Скрипты должны отражать типичные ситуации, с которыми сталкиваются пользователи при работе с чат-ботом. Например, это может быть отправка и чтение сообщений, и т.д.

1.6.3.4 Настройка среды тестирования

Некоторые программы нагрузочного тестирования могут требовать настройки среды тестирования. Например, настройка виртуальных машин для имитации реальных пользователей. Для генерации большого количества запросов ведется работа с базами данных бота, подготовка статических ресурсов (картинки, текстовые файлы итд)Конфигурация среды тестирования должна быть выполнена с учетом требований к тестируемому приложению;

1.6.3.5 Запуск теста и анализ результатов

Наконец, после настройки среды тестирования и создания скриптов, можно перейти к последнему этапу - запуску теста и анализу результатов. В тесте должно участвовать достаточное количество пользователей для достижения заданных пороговых значений. Результаты тестирования должны быть записаны и проанализированы с целью выявления более слабых мест в приложении и внесения корректировок.

1.6.4 Разработка программы

Разработка программы для нагрузочного тестирования состоит из следующих этапов:

-использование *HTTP*-сниффера для определения структуры *REST*-методов, которые должны быть протестированы;

-создание модели автомата, которая соответствует поведению пользователя при использовании чат-бота;

-разработка программы для нагрузочного тестирования на основе модели автомата;

-создание модуля отладки для выявления проблем и устранения ошибок в программе для нагрузочного тестирования;

-описание нагрузки, которая должна быть протестирована с использованием программы для нагрузочного тестирования.

В результате выполнения этих этапов создана программа для нагрузочного тестирования, которая будет использоваться для оценки

производительности сервера чат-бота [16].

На рисунке 9 представлена диаграмма развертывания, на которой изображены основные узлы выполнения программных компонентов в реальном времени, а также процессы и объекты.

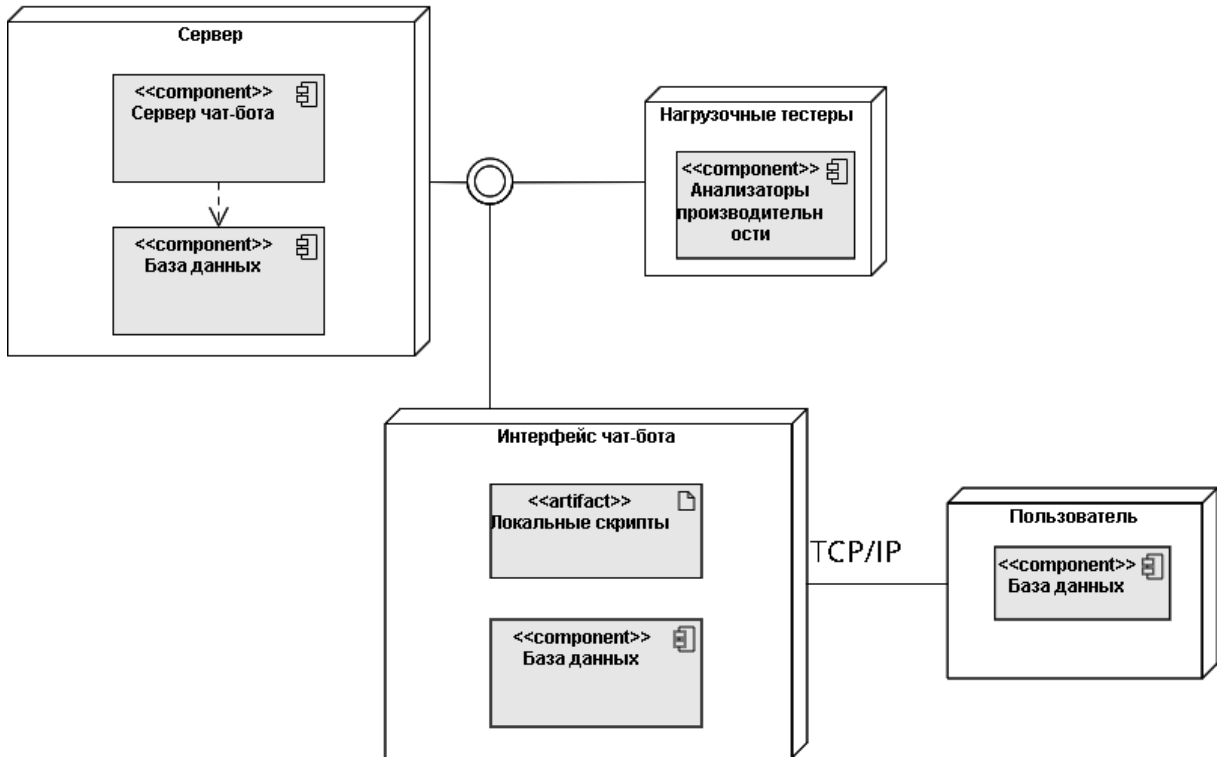


Рисунок 9 – UML-диаграмма развертывания чат-бота.

1.7 Разработка сценария вGatling

1.7.1 Создание *HAR*-файла

Для использования *Gatling* необходим *HAR*-файл, который можно сгенерировать при помощи *Gatling Recorder* путём записи пользовательского пути в *Google Chrome*. Это наиболее удобный способ, позволяющий обойти проблемы при записи *HTTP*S. Чтобы получить *HAR*-файл, следуйте этим инструкциям:

-открыть тестовый сайт *Gatling* с базой данных компьютеров;

-открыть консоль используя горячие клавиши *ctrl+shift+I;*

-нажать на “*Network*/Сеть”;

-нажать на кнопку “*Clear*”(рисунок 10);

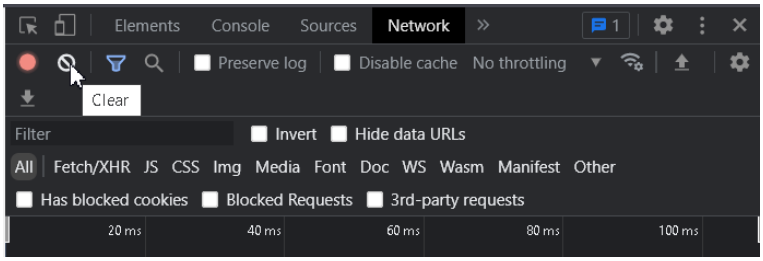


Рисунок 10 - Вкладка “*Network*/Сеть” очищена

-нажать на любое название компьютера;

-нажать на строку “*Choose a company* / Выберите компанию”;

-в открывшемся списке нажать на название любой компании;

-нажать на кнопку “*Save this company*/Сохранить этот компьютер”;

-нажать правой кнопкой мыши в любом месте раздела "Сеть" и выберите опцию "Сохранить все как файл *HAR* с содержимым". Сохранить файл в удобном для вас месте на компьютере(рисунок 11);

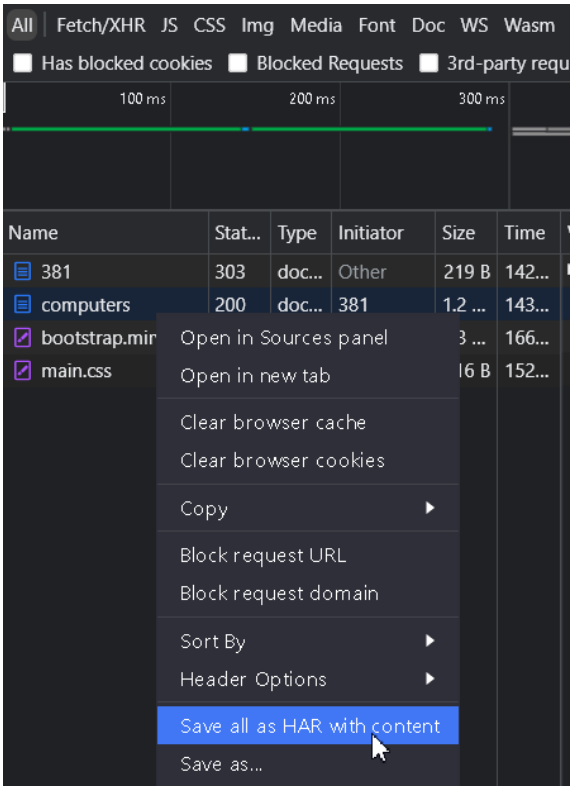


Рисунок 11 - Сохранение *HAR*-файла

-перейти в папку *Gatling bin* (где впервые был запущен *Gatling*) и запустили файл *recorder.bat*. Будет загружен рекордер *Gatling*(рисунки 12,13);

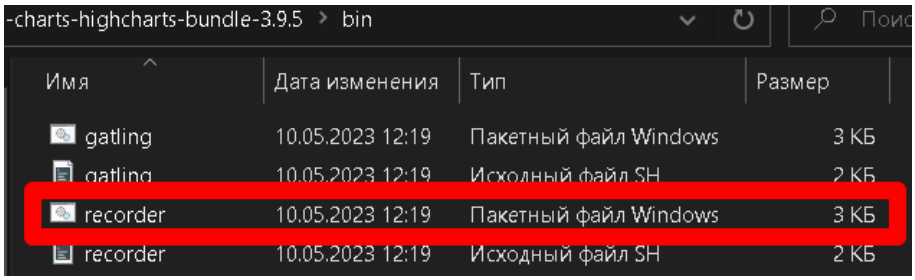


Рисунок 12 - Пакетный файл *recorder.bat*

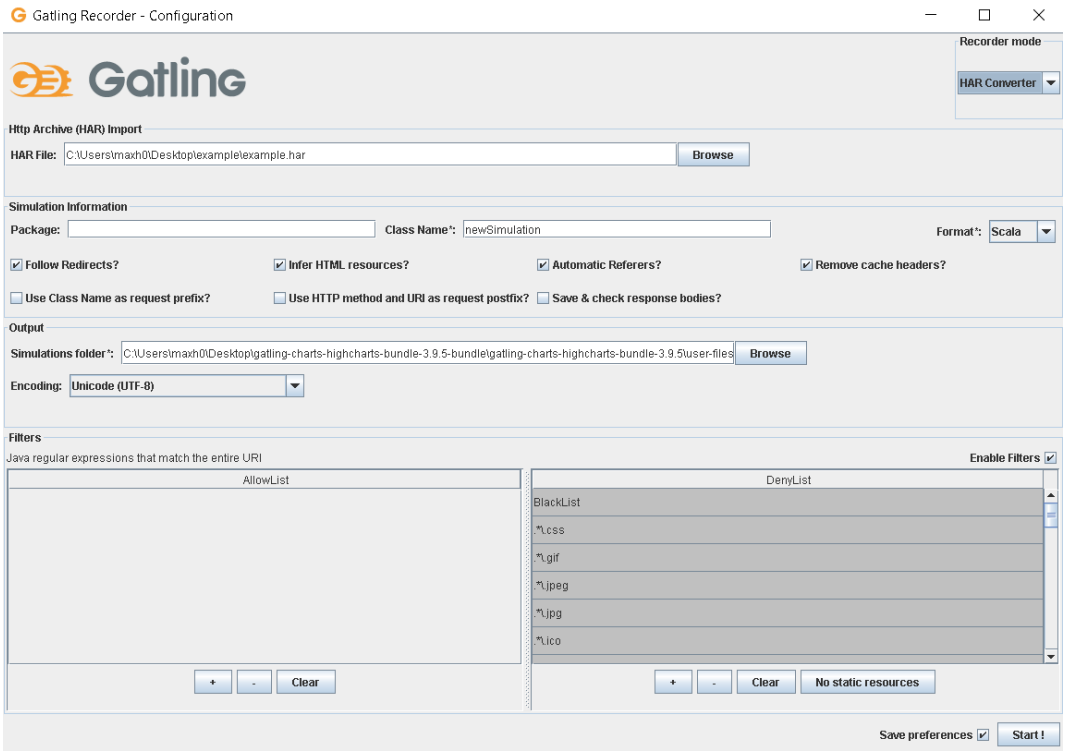


Рисунок 13 - Начальный экран *Gatling Recorder*

-изменить *Recorder Mode* в правом верхнем углу на *HAR Converter*;

-в разделе *HAR File* перейти к местоположению созданного *HAR* -файла;

-написать название скрипт, изменив *Class Name* на, например, *newSimulation*;

-нажать *Start!;*

- когда все работает корректно, отображается сообщение, что все прошло успешно(рисунок 14).

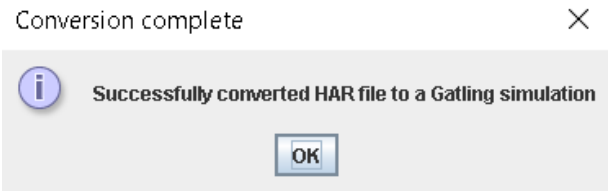


Рисунок 14 - Сообщение, когда нет ошибок

Для запуска скрипта необходимо вернуться в папку *bin Gatling* и повторно запустить *Gatling.bat*. После загрузки *Gatling*, вы сможете выбрать только что созданный скрипт.

Только что созданный скрипт, можно найти в папке user-files/*Simulation*s в каталоге *Gatling*(рисунок 15).

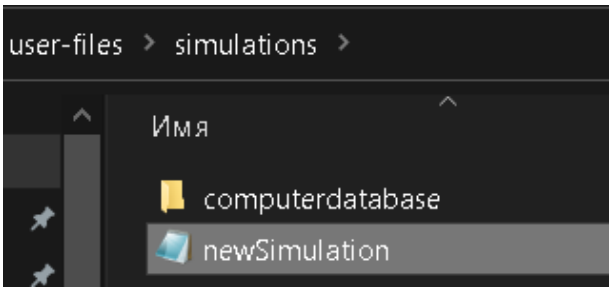


Рисунок 15 - Файл созданного скрипта

Открыть скрипт *MyComputertest*, в текстовом редакторе. Ниже представлен листинг этого скрипта:

```
import scala.concurrent.duration._
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class newSimulation extends Simulation {
  private val httpProtocol = http
    .baseUrl("http://computer-database.gatling.io")
```

```
    .inferHtmlResources(AllowList(),       DenyList("BlackList",
"""".*\.css""",   """.*\.gif""",   """.*\.jpeg""",   """.*\.jpg""",
"""".*\.ico""",          """.*\.woff""",          """.*\.woff2""",
"""".*\.(t|o)tf""",         """.*\.png""",         """.*\.svg""",
""".*detectportal\.firefox\.com.*""",               """.*\.js""",
"""".*\.css""",   """.*\.gif""",   """.*\.jpeg""",   """.*\.jpg""",
"""".*\.ico""",          """.*\.woff""",          """.*\.woff2""",
"""".*\.(t|o)tf""",         """.*\.png""",         """.*\.svg""",
""".*detectportal\.firefox\.com.*""",               """.*\.js""",
"""".*\.css""",   """.*\.gif""",   """.*\.jpeg""",   """.*\.jpg""",
"""".*\.ico""",          """.*\.woff""",          """.*\.woff2""",
"""".*\.(t|o)tf""",         """.*\.png""",         """.*\.svg""",
""".*detectportal\.firefox\.com.*"""))
    .acceptHeader("text/html,application/xhtml+xml,application/x
ml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
signed-exchange;v=b3;q=0.7")
    .acceptEncod ingHeader("gzip, deflate")
    .acceptLanguageHeader("ru,en;q=0.9,ko;q=0.8")
    .contenttypeHeader("application/x-www-form-urlencoded")
    .originHeader("http://computer-database.gatl ing.io")
    .upgradeInsecureRequestsHeader("1")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36   (KHTML,   like   Gecko)   Chrome/110.0.0.0
YaBrowser/23.3.3.719 Yowser/2.5 Safari/537.36")
 private val headers_0 = Map("Cache-Control" -> "max-age=0")
  private val scn = scenario("newSimulation")
    .exec(
      http("request_0")
        .post("/computers")
        .headers(headers_0)
        .formParam("name", "test")
        .formParam("introduced", "")
        .formParam("discontinued", "")
        .formParam("company", "1")
    )
```

```
setUp(scn.inject(atOnceusers(1))).protocols(httpProtocol)
}
```

1.7.2 Создание проекта *Gatling* из архетипа *Maven*

-открыть терминал или командную строку и ввести: *mvn archetype:generate;*

-в конце будет этот запрос: *Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains):* Нужно ввести *gatling*;

-затем будет: *1: remote -> io.gatling.highcharts:gatling-highcharts-Maven-archetype (gatling-highcharts-Maven-archetype);*

-ввести 1 для выбора архетипа *Gatling*.

На следующем экране выбрать последнюю версию(рисунок 16)

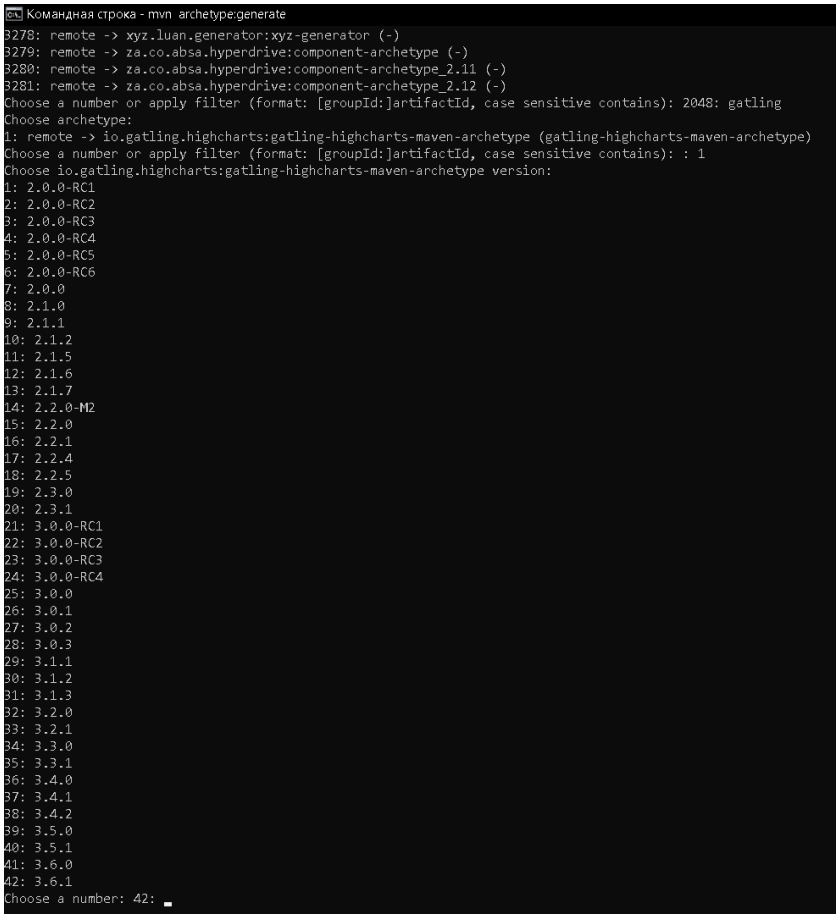


Рисунок 16 - Выбор версии *Gatling*

-далее для *groupId* ввести *com.gatlingTest;*

-для *artifactId* ввести *myGatlingTest;*

-для *version* просто нажать *ENTER*, чтобы принять *1.0-SNAPSHOT;*

-для *package* нажать *ENTER* еще раз, чтобы принять в качестве имени пакета *com.gatlingTest;*

-в конце ввести *Y*, чтобы подтвердить все настройки, и *Maven* создаст для вас новый проект(рисунки 17,18).

```
Define value for property 'groupId': com.gatlingTest
Define value for property 'artifactId': myGatlingTest
Define value for property 'version' 1.0-SNAPSHOT: : 1.0-SNAPSHOT
Define value for property 'package' com.gatlingTest: : com.gatlingTest
Confirm properties configuration:
groupId: com.gatlingTest
artifactId: myGatlingTest
version: 1.0-SNAPSHOT
package: com.gatlingTest
 Y: :
```

Рисунок 17 - Настройка проекта *Gatling* из архетипа *Maven*

```
Y: : y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: gatling-highcharts-maven-archetype:3.6.1
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.gatlingTest
[INFO] Parameter: artifactId, Value: myTest
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.gatlingTest
[INFO] Parameter: packageInPathFormat, Value: com/gatlingTest
[INFO] Parameter: package, Value: com.gatlingTest
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: com.gatlingTest
[INFO] Parameter: artifactId, Value: myTest
[INFO] Project created from Archetype in dir: C:\Users\maxh0\myTest
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:05 min
[INFO] Finished at: 2023-05-12T20:35:11+03:00
[INFO] ------------------------------------------------------------------------

C:\Users\maxh0>
```

Рисунок 18 - Итог проведения шагов, представленных выше

Далее импортируем этот проект в *IntelliJ*:

-на стартовой странице *IntelliJ* выбрать *Import Project*.

-перейти в папку проекта, и выбрать файл *pom.xml*.

-нажать open (рисунок 19).

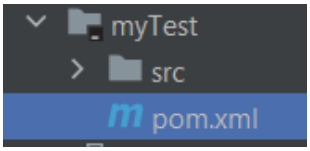


Рисунок 19 - Файл *pom.xml*

Структура проекта:

-pom.xml – это файл *Maven*, используемый для конфигурирования проекта;

-src/Gatling/Scala – это место, где находятся сценарии *Gatling*;

-src/Gatling/resources – это место, где находятся ресурсы, используемые в тестах *Gatling*, например, конфигурационные файлы.

После завершения импорта проекта нужно открыть панель "*Project Directory*" в левой части экрана и раскрыть папку "*src>test>scala*". Дважды нажать на класс "*Engine.scala*". Если в верхней части экрана появится сообщение "*No Scala SDK in module*", то нажать кнопку "*Setup Scala SDK*"(рисунки 20,21):

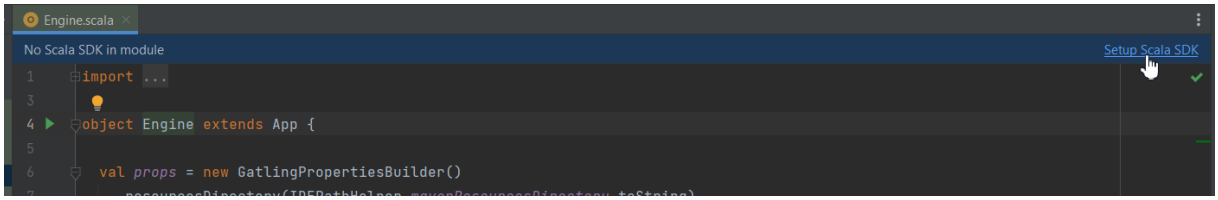


Рисунок 20 - "*No Scala SDK in module*"

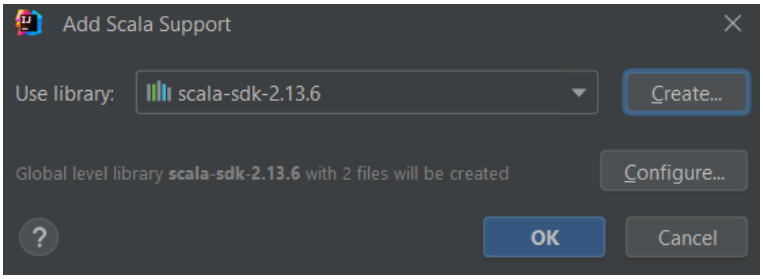


Рисунок 21 - Загрузка версии библиотеки *scala*

1.7.3 Начало работы с тестом производительности *Gatling*

Для успешной компиляции и работы с проектом на языке *Scala* в среде разработки *IntelliJ* необходимо отметить папку *Scala* в качестве исходной. Это можно сделать, нажав правой кнопкой мыши на папку *Scala* и выбрав опцию *Mark Directory As -> test Sources Root*(рисунок 22):

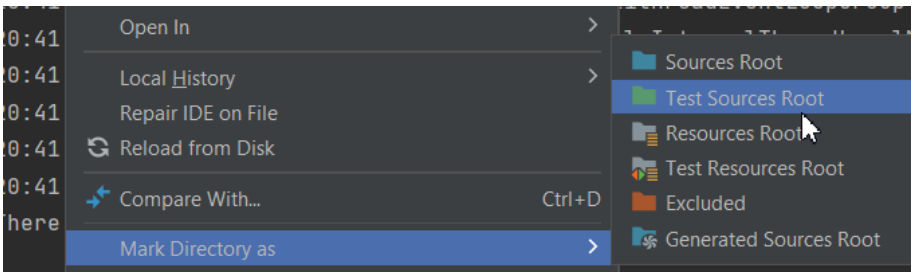


Рисунок 22 - Отметили папку *Scala* в качестве исходной

Отметить всю папку *src* как *source root*, как показано на рисунке 23:

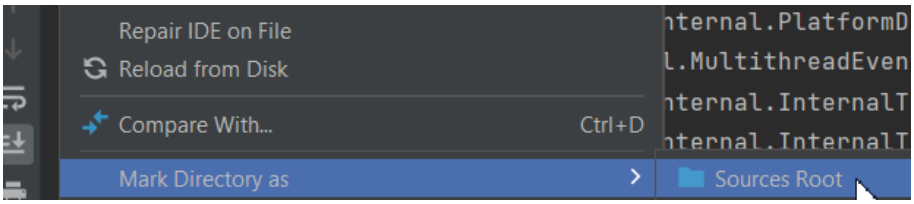


Рисунок 23 - Отметили папку *src* как *source root*

1.7.4 Добавление скрипта *Gatling*

Для добавления скрипта нужно открыть папку "*scala*". Нажать правую кнопку мыши на этой папке. Выбрать пункт "New > *Scala* Package". Назвать создаваемый пакет "*computerData*base". Нажать правую кнопку мыши на созданный пакет "*computerData*base". Выбрать пункт "*New > Scala Class*". Назвать создаваемый класс "*BasicSimulation*".

Для выполнения примера моделирования, включенного в проект, нужно сделать следующее:

-открыть папку *src*-> *test*-> *scala*;

-найти файл класса *Engine* и открыть его;

-запустить класс *Engine* для выполнения предварительно настроенного сценария моделирования, который поставляется в комплекте с проектом(рисунок 24).

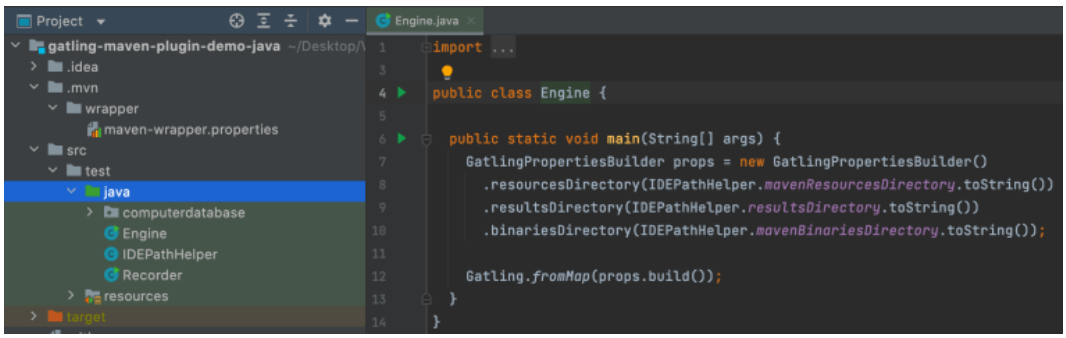


Рисунок 24 - Исходный класс Engine

Далее для формирования отчета нужно запустить *Gatling* тест напрямую через *Maven* из командной строки:

-открыть терминал в каталоге вашего проекта;

-ввести *mvn Gatling:test.* Эта команда выполнит *Gatling* тест с помощью *плагина Maven для Gatling*(рисунок 25) [17].

```
C:\Users\maxh0\myGatlingTest>mvn gatling:test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.gatlingTest:myGatlingTest >-------------------
[INFO] Building myGatlingTest 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- gatling:3.1.2:test (default-cli) @ myGatlingTest ---
Simulation computerdatabase.BasicSimulation started...

================================================================================
2023-05-12 21:11:30                                           5s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=2      KO=0     )
> request_1                                                (OK=1      KO=0     )
> request_1 Redirect 1                                     (OK=1      KO=0     )

---- Scenario Name -------------------------------------------------------------
[------------------------------------------------------------------------]  0%
          waiting: 0      / active: 1      / done: 0
================================================================================



================================================================================
2023-05-12 21:11:33                                           7s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=2      KO=0     )
> request_1                                                (OK=1      KO=0     )
> request_1 Redirect 1                                     (OK=1      KO=0     )

---- Scenario Name -------------------------------------------------------------
[########################################################################]100%
          waiting: 0      / active: 0      / done: 1
================================================================================

Simulation computerdatabase.BasicSimulation completed in 7 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
```

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                          2 (OK=2      KO=0     )
> min response time                                    148 (OK=148    KO=-     )
> max response time                                    692 (OK=692    KO=-     )
> mean response time                                   420 (OK=420    KO=-     )
> std deviation                                        272 (OK=272    KO=-     )
> response time 50th percentile                        420 (OK=420    KO=-     )
> response time 75th percentile                        556 (OK=556    KO=-     )
> response time 95th percentile                        665 (OK=665    KO=-     )
> response time 99th percentile                        687 (OK=687    KO=-     )
> mean requests/sec                                 0.25 (OK=0.25   KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                             2 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 0s.
Please open the following file: C:\Users\maxh0\myGatlingTest\target\gatling\basicsimulatio
n-20230512181122048\index.html[INFO] --------------------------------------------------
------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.860 s
[INFO] Finished at: 2023-05-12T21:11:33+03:00
[INFO] ------------------------------------------------------------------------
```

Рисунок 25 - Результаты тестирования

Отчет храниться в папке *Gatling* в выходном каталоге инструмента сборки (в папке *tarGET*). Откройте отчет в доступном браузере и проанализируйте результат выполнения. Отчет можно открыть в доступном браузере, т к поверх текстового результата внутри консоли *Gatling* создает отчет на основе HTML для дальнейшего изучения. Отчет представлен на рисунках 26-.

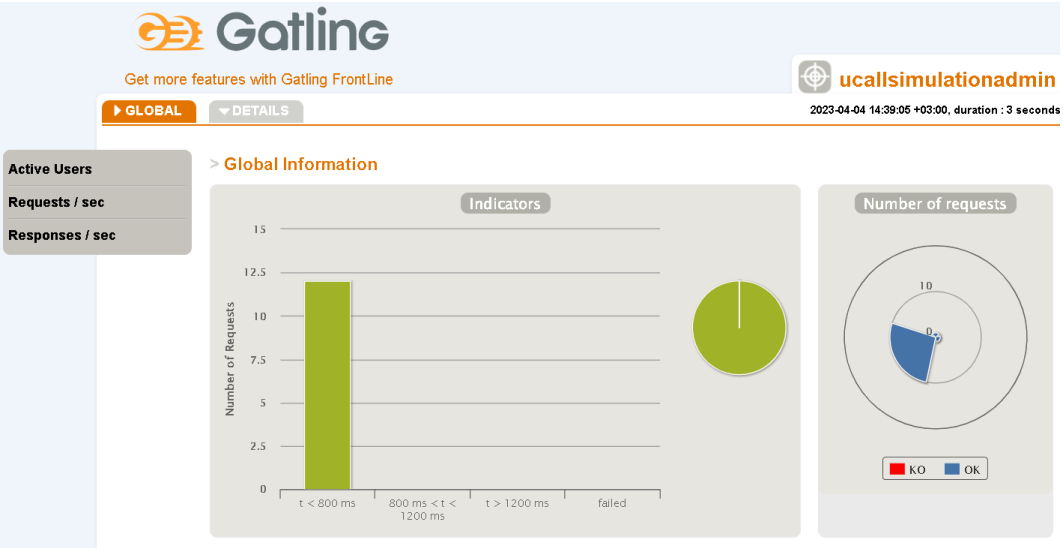


Рисунок 26 - Общая информация

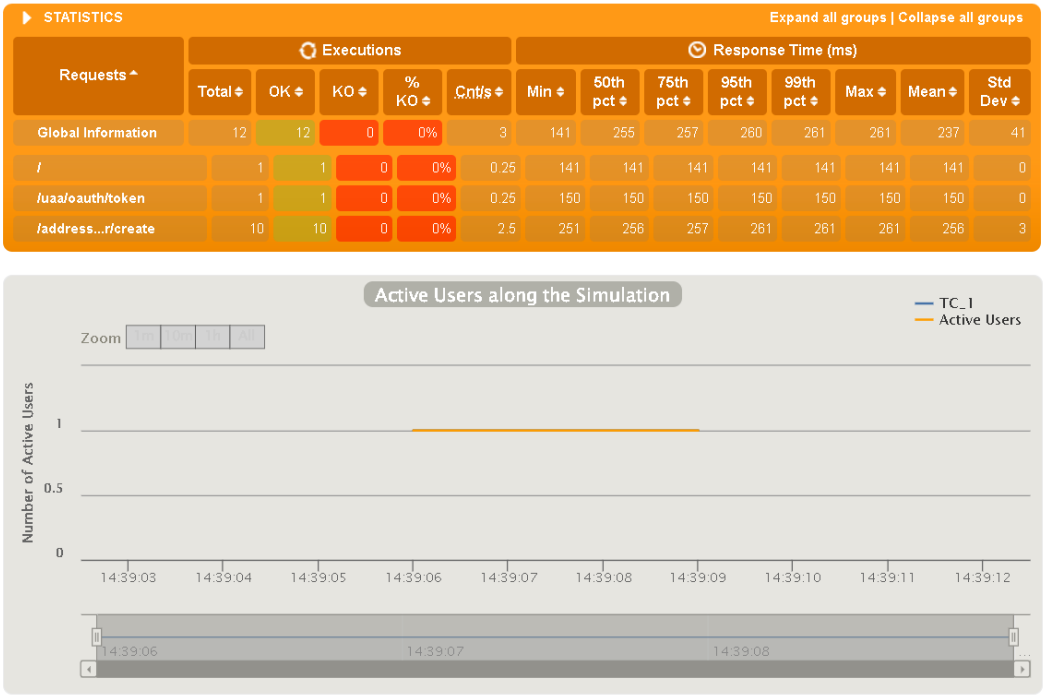


Рисунок 27 - Активные пользователи симуляции

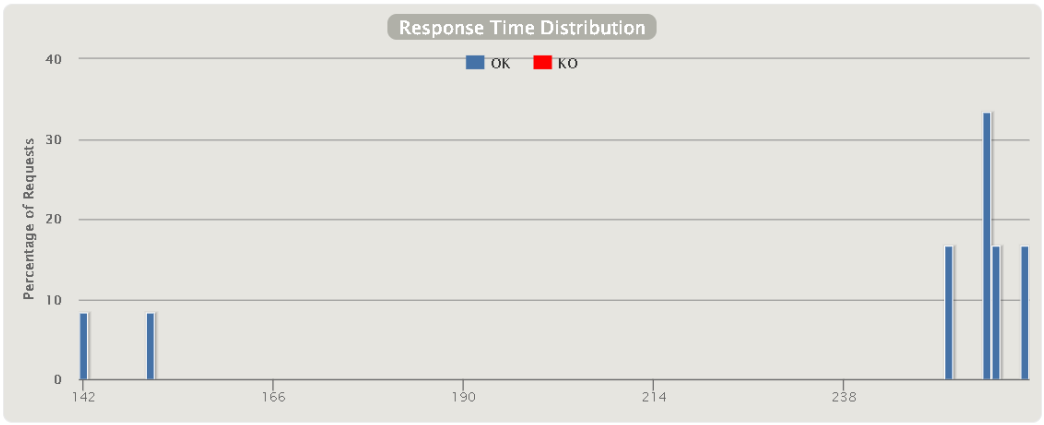


Рисунок 28 - Распределение времени отклика

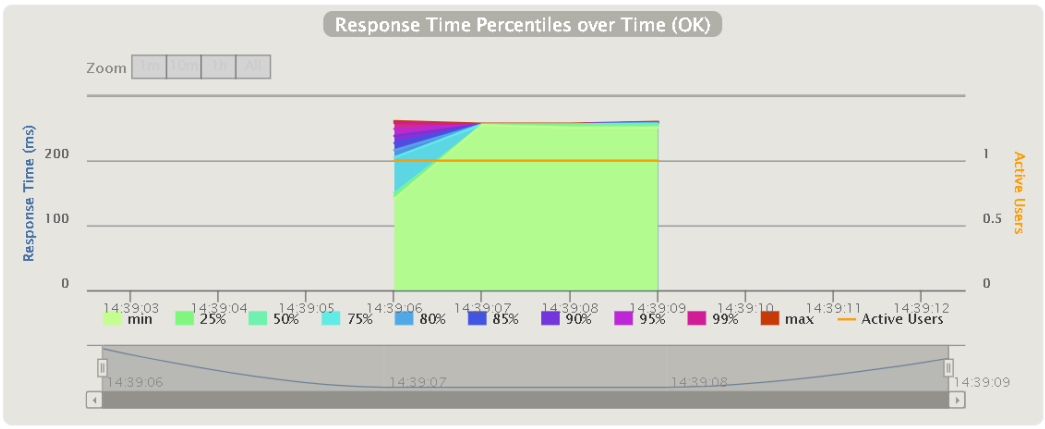


Рисунок 29 - Процент времени отклика с течением времени

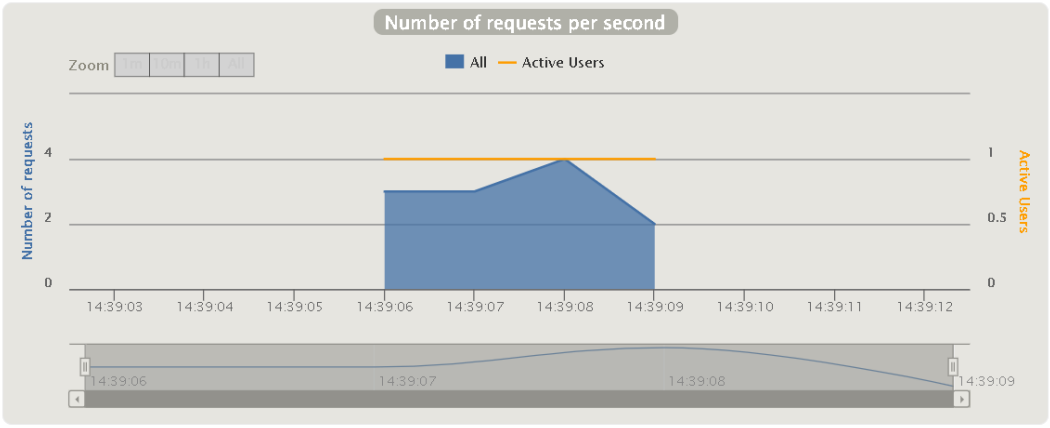


Рисунок 30 - Количество запросов в секунду

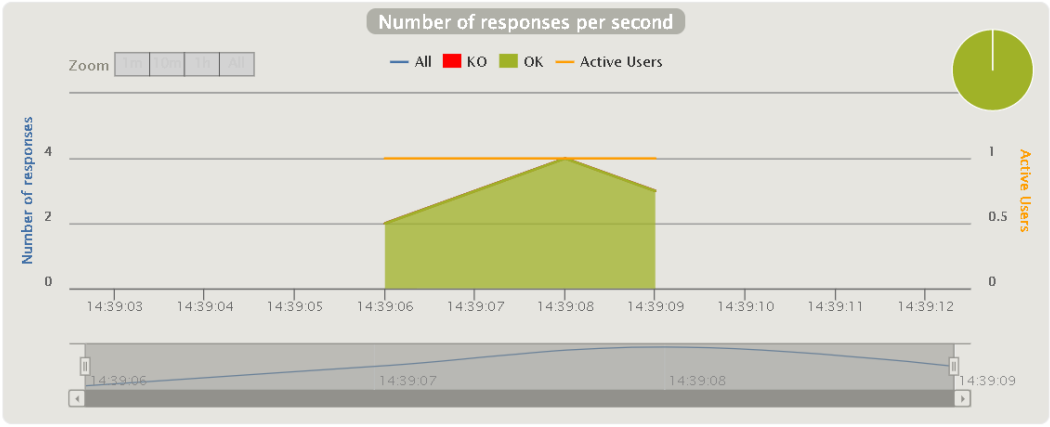


Рисунок 31 - Количество ответов в секунду

1.7.5 Анализ результата

По результатам проверки в условиях нагрузки не было выявлено никаких ошибок в системе, поэтому тестирование на выдержку было выполнено удачно.

2 Охрана труда

Данный раздел посвящен вопросам охраны труда и безопасности жизнедеятельности пользователей ПЭВМ в процессе разработки и эксплуатации автоматизированной системы нагрузочного тестирования чат-бота. Основная цель - минимизация пагубных факторов на здоровье специалистов, работающих с ПЭВМ, и обеспечение безопасности процесса разработки и эксплуатации системы.

Таким образом, в рамках данной главы необходимо рассмотреть основные меры и требования, которые должны быть выполнены для обеспечения безопасности работников при проведении нагрузочного тестирования чат-бота.

2.1 Возможные риски и опасности при проведении нагрузочного тестирования

Перед началом работы необходимо провести анализ возможных рисков и опасностей при проведении нагрузочного тестирования чат-бота. К таким рискам могут относиться несколько аспектов:

-некорректное использование оборудования и инструментов во время проведения тестирования;

-несоблюдение мер безопасности при обслуживании и техническом обслуживании оборудования;

-риск эксплуатации безопасных и неисправных устройств;

-несанкционированный доступ к приложениям и данным, которые используются при проведении тестирования;

-работа с интернет-ресурсами, содержание которых может быть небезопасно.

2.2 Меры по обеспечению безопасности

Для обеспечения безопасности при проведении нагрузочного тестирования чат-бота рекомендуется принимать следующие меры:

2.2.1. Организация рабочего места

Перед началом работы необходимо следить за правильной организацией рабочего места. Все рабочие места должны быть оборудованы эргономичной мебелью, а также быть достаточно просторными для работы. Также необходимо обеспечить качественный освещение, вентиляцию и микроклимат на рабочих местах.

2.2.2. Использование средств индивидуальной защиты

При проведении тестирования, необходимо обеспечивать работникам средства индивидуальной защиты, такие как перчатки, маски, очки и другие средства. Это поможет предотвратить возможные травмы и пожары, а также защитить работников от шума и электромагнитного воздействия.

2.2.3. Предупреждение аварийных ситуаций

Для предотвращения возможных аварий на рабочих местах необходимо обеспечить безопасное использование оборудования и инструментов, а также правильный подход к обслуживанию и техническому обслуживанию оборудования. Все технические работы должны проводиться только в соответствии с мерами безопасности и профессиональными навыками личного состава.

2.2.4. Обучение и тренировка персонала

Важным аспектом в обеспечении безопасности на рабочих местах является обучение и тренировка персонала. Работники должны быть проинформированы о возможных рисках и опасностях, а также о мерах защиты и безопасности при проведении работ. Также необходимо обеспечить персональные тренинги и прокладывать организационный путь для повышения профессиональной квалификации.

2.2.5. Контроль соблюдения мер безопасности

Для обеспечения безопасности на рабочих местах необходимо проводить контроль за соблюдением мер безопасности на рабочих местах. Контроль возможен как отдельно по каждому работнику, так и групповыми методами. Важно настроить систему контроля за соблюдением мер безопасности и регулярно модернизировать её для решения возможных проблем.

Все эти меры помогут обеспечить безопасность при проведении нагрузочного тестирования чат-бота, защитить персонал и оборудование от возможных рисков и опасностей. Кроме того, следует обеспечить взаимодействие с соответствующими органами стандартизации для получения рекомендаций и соответствующих обновлений.

Заключение

В рамках дипломной работы были рассмотрены основные проблемы, которые решает тестирование, а также протоколы современных чат-ботов, и тестирование под нагрузкой: *REST API* и технология *WebSocket*. Были проанализированы инструменты нагрузочного тестирования и их преимущества и недостатки.

Так же проведено нагрузочное тестирование чат-бота с использованием разработанной программы. Были получены результаты, которые позволили сделать выводы о производительности системы.

Более того рассмотрены преимущества и недостатки различных инструментов, используемых в нагрузочном тестировании. Была разработана структура и написан скрипт для нагрузочного тестирования чат-бота с использованием инструмента *Gatling*.

В ходе проведения нагрузочного тестирования была выявлена максимальная пропускная способность системы и определены возможные узкие места, которые необходимо устранить для повышения производительности.

Результаты данной работы могут быть использованы для улучшения качества чат-ботов и повышения уровня их производительности.

Таким образом, данная дипломная работа раскрывает способы выполнения нагрузочного тестирования и предлагает эффективное решение с использованием инструмента *Gatling*.

Список используемых источников

1. Мартинез М., Мартинез Ф., Рубио О. Тестирование программного обеспечения: Практический подход.
2. "*testing computer* Software" by Cem Kaner, Jack Falk, and Hung Q. Nguyen.
3. Бондаренко, В. Требования к тестированию ПО // Открытые системы. ПОЛИТЕХ-2015: Труды XX Международной научно-технической конференции. – СПб.: Изд-во Политехн. ун-та.
4. "Performance *testing* Gu*Id*ance for Web Applications" by Microsoft.
5. "*Gatling*: High Performance *load testing*", by Alan Mark Berg.
6. "Performance *testing* with *Gatling*", by Ric*HAR* d Bradley.
7. "*computer Networking*: A Top-Down Approach" by James F. Kurose and Keith W. Ross.
8. "Chatbot Development: A Beginner's Gu*Id*e to Build*ing* AI Robots" by Murtaza Rizvi.
9. "*load testing* Us*ing Gatling*: From Beginner To Advanced" by Deepak Kumar Mishra.
10. Aniche, M., Klinkenberg, S., & Bacchelli, A. (2018). The impact of *REST API* evolution on client applications. Empirical Software Engineer*ing*.
11. "*WebSocket*: Lightweight Client-Server Communications" by Andrew Lombardi.
12. "Тестирование программного обеспечения. Практика автоматизации" Д.Аграманов, В.Евтушенко, С.Петухов, М.Шишкулов.
13. Семёнов, А. М. Создание сценариев нагрузочного тестирования А. М. Семёнов.
14. "Автоматизация тестирования программного обеспечения" В.С. Буханов, В.В. Киркин.
15. "Технологии автоматизированного тестирования" Алексея Баранова.

16. "Разработка тестовых сценариев для нагрузочного тестирования веб-приложений на языке программирования Python с использованием фреймворка *Gatling*" Владимира Гаджиханова и Ивана Субботина.

17. "Нагрузочное тестирование в *Gatling*" Павел Симаков.

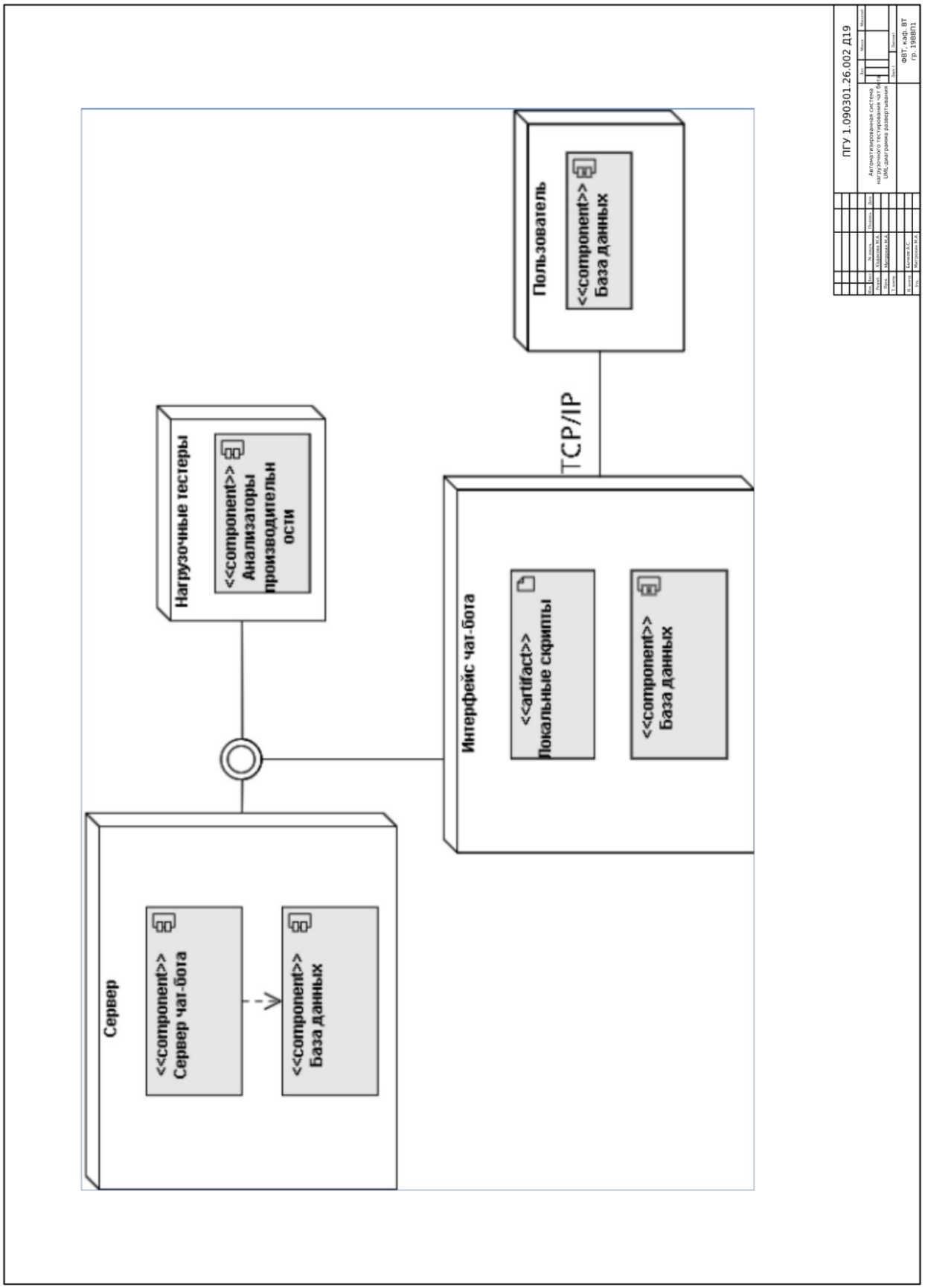
Приложение А

(Обязательное)

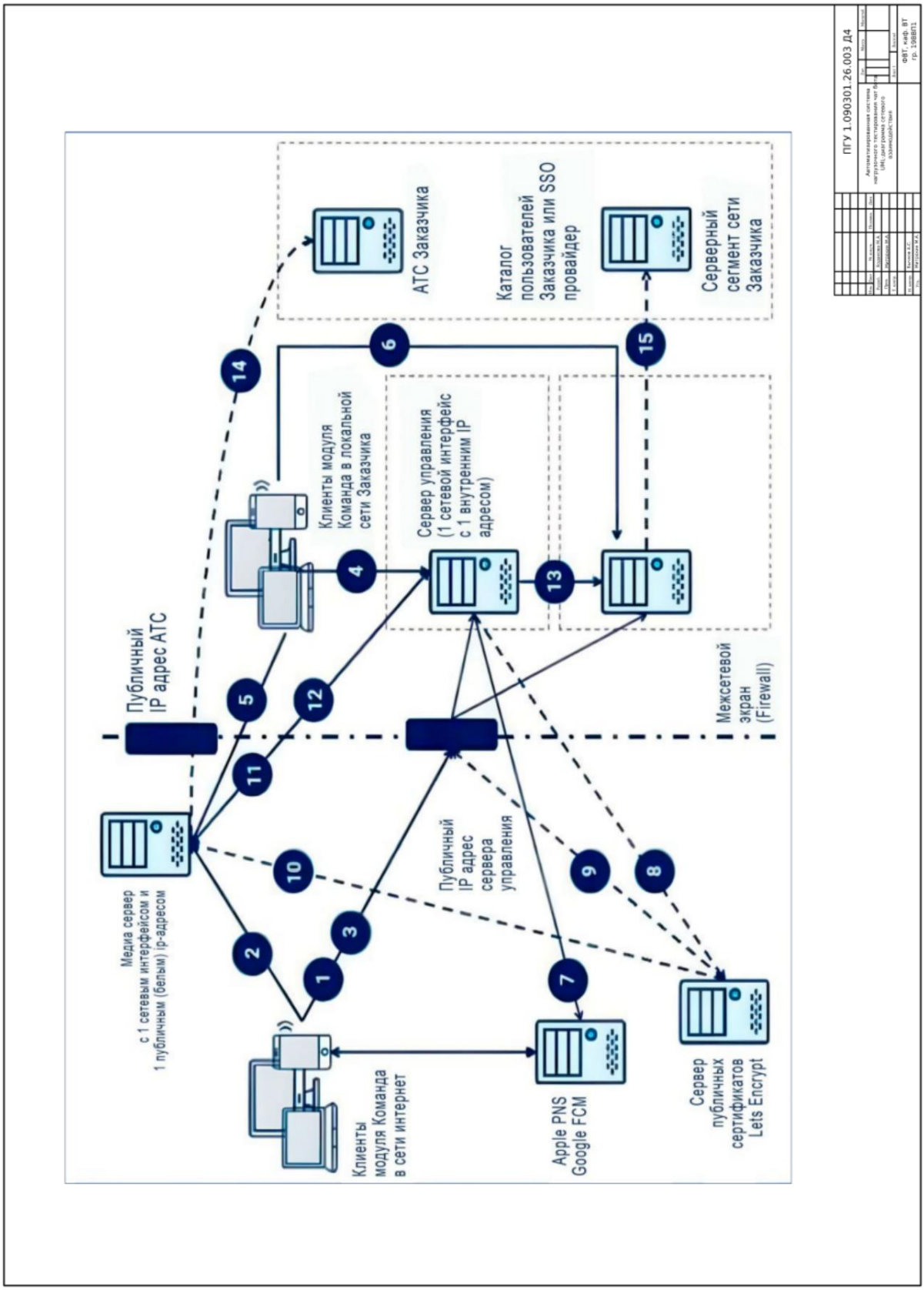
Автоматизированная система нагрузочного тестирования чат-бота

UML-диаграммы

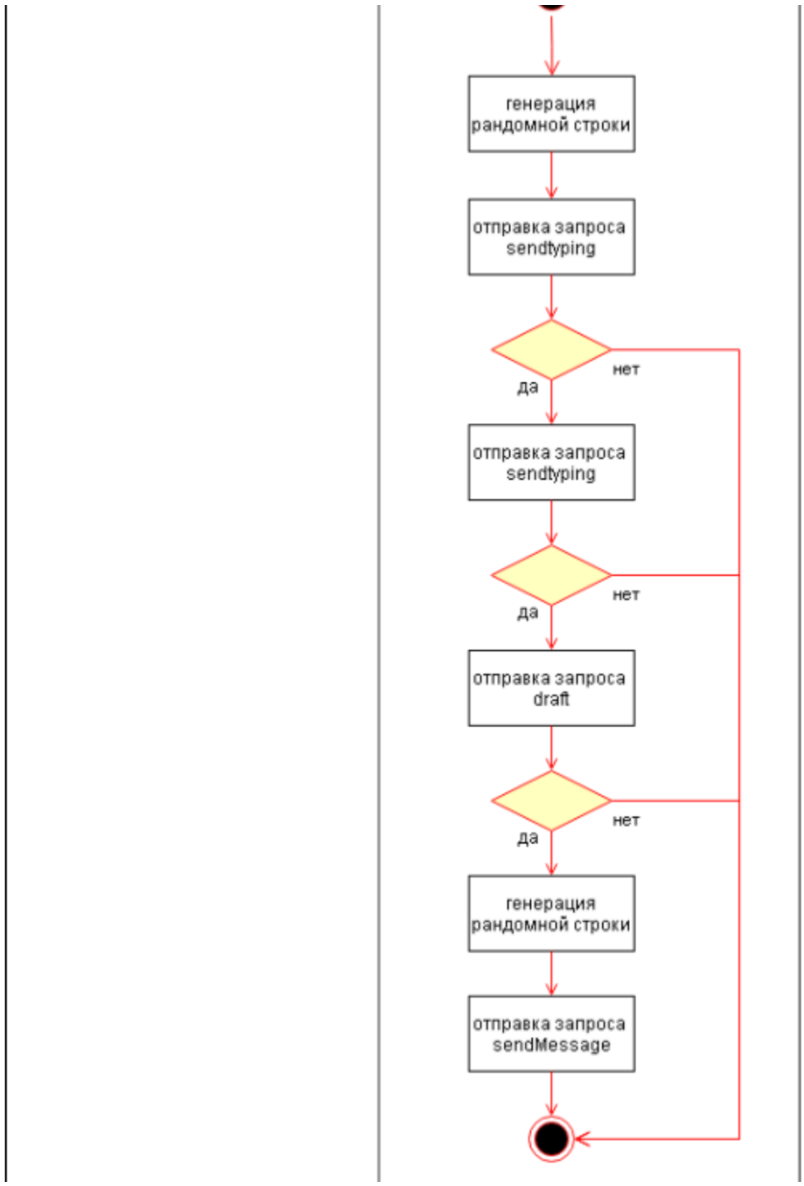
Приложение А.1 - *UML*-диаграмма развертывания



Приложение А.2 - *UML*-диаграмма сетевого взаимодействия



Приложение А.3 - UML-диаграмма активности отправки сообщений



Приложение А.4 - UML-диаграмма активности чтения сообщений

Приложение Б

(Обязательное)

Автоматизированная система нагрузочного тестирования чат-бота

Листинг программы

Файл *newSimulation.scala*

```
object sendMessage {

    //? Генерация рандомной строки в localID
    val random = new Random()
    val                              alp                                    =
"1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM"

    //? Отправка сообщения по сокету
    val sendmessageuser = repeat(5) {
      exec(ws("WSS sendtyping 1")
        .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"${roomID}","userID":"${userID}","type":
1}]"""))

        .pause(500 milliseconds)

        .exec(ws("WSS sendtyping 0")
          .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"${roomID}","userID":"${userID}","type":
0}]"""))

        .exec(ws("WSS draft")
          .sendText("""42/spikaenterprise,
["draft",{"roomID":"${roomID}","text":null}]"""))

        .exec(session => {
          session.set("localID",
Iterator.continually(random.nextInt(alp.size)).map(alp).take(32)
.mkString)
        })

        .exec(ws("WSS sendmessage")
          .sendText("""42/spikaenterprise,
```

```
["sendmessage",{"localID":"_${localID}}","message":"0301b22f5c3a
448ca986fbc6b6ff097aef824e85d34dbf013cc834c52328613136e7adbf1c2a
85fe661d6b79135ec330a5
7cf1970535e6040ccb30f60b7f5b762626c3a30eaa2c6f5b8bffb16e77a87344
35","roomID":"${roomID}","userID":"${userID}","type":1,"attribut
es":{"useclient":"web"},"user":${userData},"_id":"_${localID}","
created":1568970116719,"isSending":true,"direction":2}]""")
        )

        .pause(500 milliseconds)
    }

    val sendmessageGrup = repeat(5) {

      exec(ws("WSS sendtyping 1")
        .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"3-
${targetGroup}","userID":"${userID}","type":1}]"""))
        //.pause(500 milliseconds)
        .exec(ws("WSS sendtyping 0")
          .sendText("""42/spikaenterprise,
["sendtyping",{"roomID":"3-
${targetGroup}","userID":"${userID}","type":0}]"""))

        .exec(ws("WSS draft")
          .sendText("""42/spikaenterprise,
["draft",{"roomID":"3-${targetGroup}","text":null}]"""))

        .exec(session => {
          session.set("localID",
Iterator.continually(random.nextInt(alp.size)).map(alp).take(32)
.mkString)
        })
        .exec(ws("WSS sendmessage")
```

```
          .sendText("""42/spikaenterprise,
["sendmessage",{"localID":"_${localID}","message":"0301cf8f1599c
f1a39fd90cd8ddda6f55af
6853ab7e2b7986594f7c9334c6fd62600429f6c1ed6ff385dddc1d495b08e08c
ccdd0bef865d06458d2a5d39f1aeb004f1a4de3d2d0d5d9f3e28e1584c2ba828
6","roomID":"3-
${targetGroup}","userID":"${userID}","type":1,"attributes":{"use
client":"web"},"user":${userData},"_id":"_${localID}","created":
1569317150863,"isSend ing":true,"direction":2}]""")
        ).pause(500 milliseconds)
    }
  }

   object readmessage {
    val   readmessageGrup   =   exec(http("/api/v2/message/list/3-
{targetGroup}/0/new")
      .get("/api/v2/message/list/3-${targetGroup}/0/new")
      .headers(headers_2)
      .queryParam("_", "1677495278092")
      .check(jsonPath("$.data.messages                          [-
1:]._id").saveAs("maxmessageId"))
    )
      .exec(ws("WSS read_message")
        .sendText("""42/spikaenterprise,
["read_message",{"operationId":1676897708185,"maxmessageId":"${m
axmessageId}"}]""")
      )
      .exec(ws("WSS read_message")
        .sendText("""42/spikaenterprise,
["read_message",{"roomID":"3-
${targetGroup}","maxmessageId":"${maxmessageId}","unreadCount":0
,"unreadMentionsCount":0}]""")
      )
  }
```

Приложение В

(Обязательное)

Автоматизированная система нагрузочного тестирования

чат-бота

Презентация

Автоматизированная система нагрузочного тестирования чат-бота

Автор работы:
студент гр. 19ВВП1 Ходакова Мария Алексеевна

Научный руководитель :
к.т.н. Синев Михаил Петрович

Пенза, 2023

Цель и объект работы

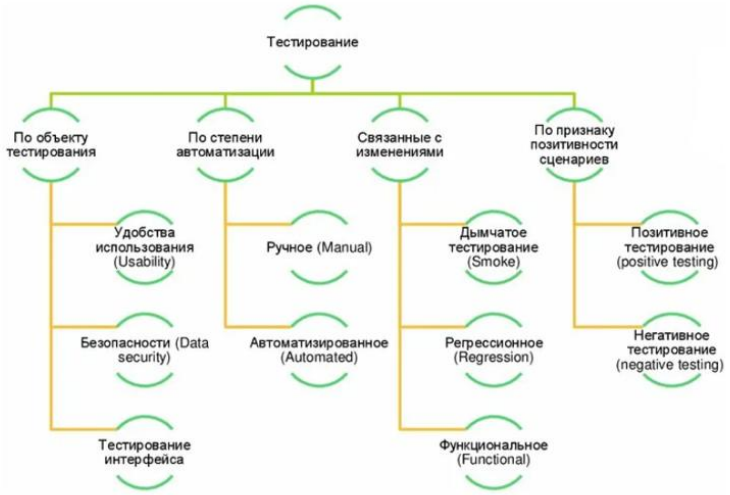
Цель работы - разработать систему нагрузочного тестирования сервера.

Объекты работы - программа нагрузочного тестирования сервера, для создания октрой нужны:

1. Сценарии нагрузки, в которых производится нагрузочное тестирование на максимальное количество одновременных пользователей - определяется максимальное количество пользователей, одновременно общающихся с ботом, при котором бот продолжает работать без задержек и сбоев;
2. Модель автомата, которая соответствует поведению пользователя в созданных сценариях;
3. Программа для нагрузочного тестирования на основе модели автомата;

В результате будет создана программа для нагрузочного тестирования, которая будет использоваться для оценки производительности сервера чат-бота.

Направления тестирования



Этапы нагрузочного тестирования

1. Анализ требований и сбор информации о тестируемой системе
2. Разработка модели нагрузки
3. Выбор инструмента для нагрузочного тестирования
4. Проведение тестирования
5. Анализ результатов

1. Анализ требований и сбор информации о тестируемой системе и выбор инструментов для нагрузочного тестирования

Используемые инструменты и технологии

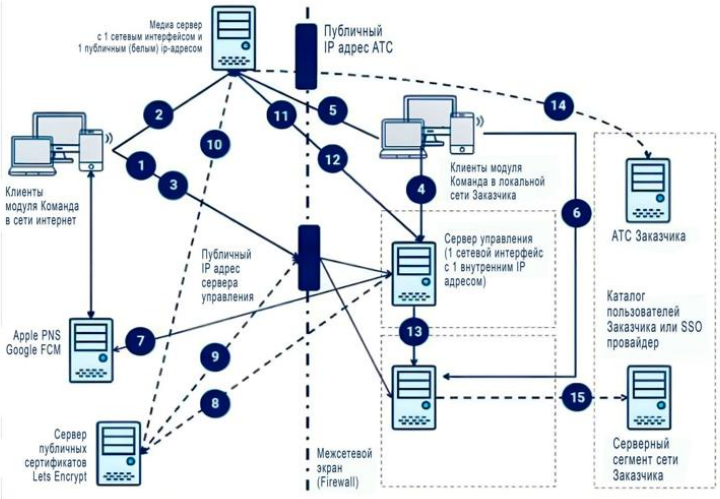
Gatling

Scala

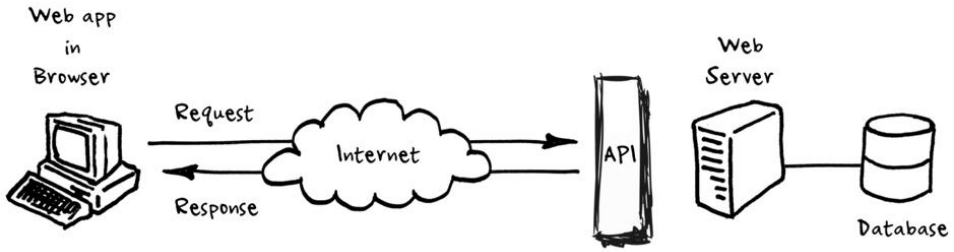
IJ

Maven™

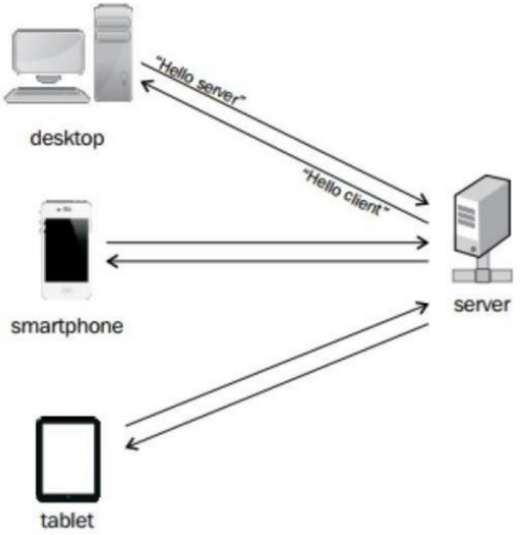
Архитектура сетевого взаимодействия



REST API, что расшифровывается как "Передача состояния представления", был придуман Роем Филдингом, который также является одним из создателей протокола HTTP. REST-сервисы отличаются тем, что наилучшим образом используют возможности протокола HTTP.



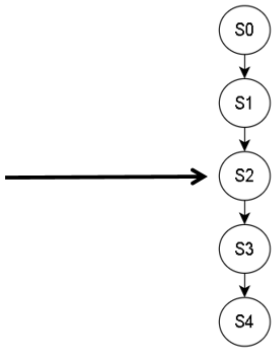
WebSocket - это передовая технология, которая позволяет установить двунаправленную интерактивную коммуникацию между браузером пользователя и сервером. С помощью этого API можно отправлять сообщения на сервер и получать управляемые ответы без явного запроса ответа от сервера.



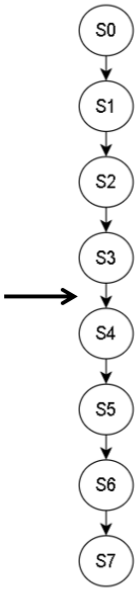
2. Разработка модели нагрузки

Разработка сценариев и автоматных моделей

1. Чтение сообщений:
   - Начальное состояние (S0)
   - Отправка запроса на получение списка новых сообщений через HTTP (S1)
   - Сохранение последнего идентификатора сообщения (S2)
   - Отправка запроса read_message с параметрами operationId и maxMessageId равными соответствующим значениям (S3)
   - Отправка запроса read_message с параметрами roomID, maxMessageId, unreadCount и unreadMentionsCount равными соответствующим значениям (S4)

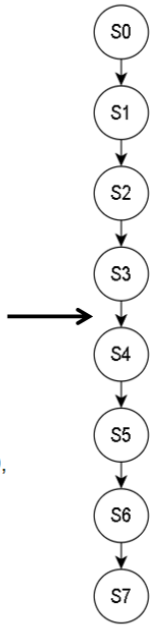


2.Отправка сообщения пользователю через сокет:
   - Начальное состояние (S0)
   - Генерация рандомной строки в localID (S1)
   - Отправка запроса sendtyping с параметрами roomID, userID, type равными roomId, userId и 1 (S2)
   - Пауза 500 миллисекунд (S3)
   - Отправка запроса sendtyping с параметрами roomID, userID, type равными roomId, userId и 0 (S4)
   - Отправка запроса draft с параметрами roomID равным roomId, и text равным null (S5)
   - Генерация рандомной строки длиной 32 символа и сохранение в localID (S6)
   - Отправка запроса sendMessage с параметрами localID, message, roomID, userID, type, attributes, user, _id, created, isSending и direction равными соответствующим значениям (S7)

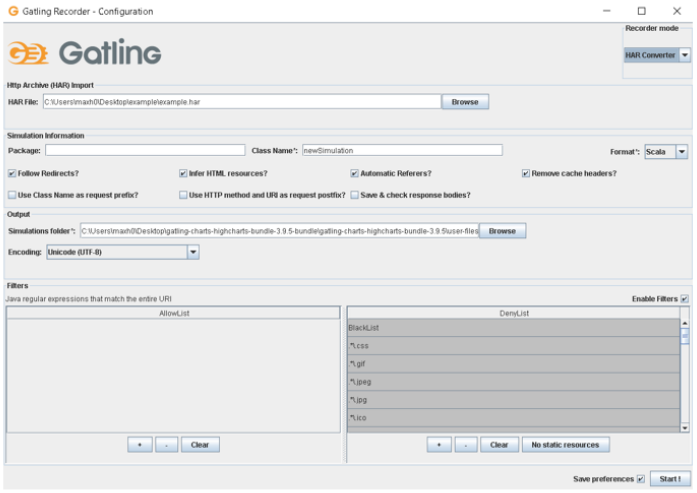


3.Отправка сообщения в группу через сокет:

- Начальное состояние (S0)
- Генерация рандомной строки в localID (S1)
- Отправка запроса sendtyping с параметрами roomID, userID, type равными 3-targetGroup, userId и 1 (S2)
- Отправка запроса sendtyping с параметрами roomID, userID, type равными 3-targetGroup, userId и 0 (S3)
- Отправка запроса draft с параметрами roomID равным 3-targetGroup, и text равным null (S4)
- Генерация рандомной строки длиной 32 символа и сохранение в localID (S5)
- Отправка запроса sendMessage с параметрами localID, message, roomID, userID, type, attributes, user, _id, created, isSending и direction равными соответствующим значениям (S6)
- Пауза 500 миллисекунд (S7)

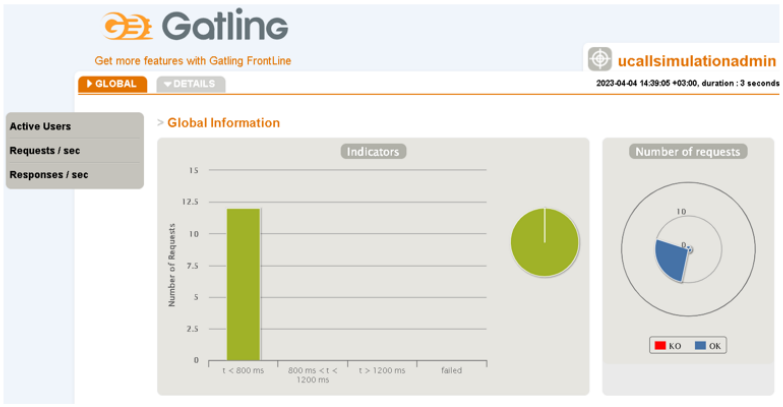


4. Проведение тестирования



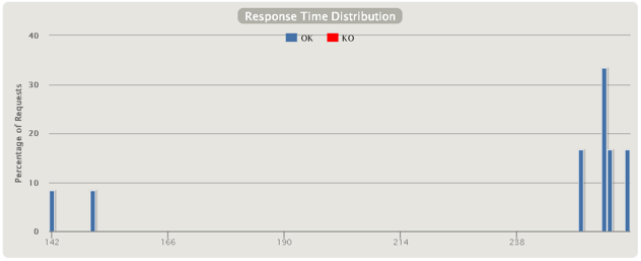
5. Результаты и анализ тестирования

Gatling создает отчет на основе HTML для дальнейшего изучения. Отчет представлен на следующих рисунках: на этом рисунке представлены показатели и количество запросов

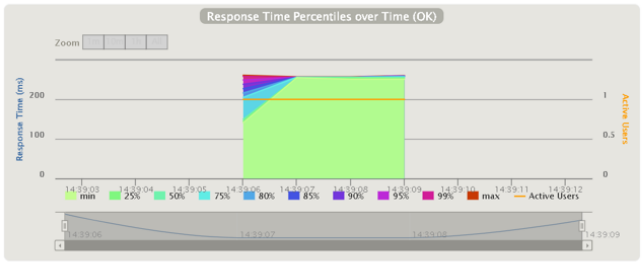


Анализ тестирования

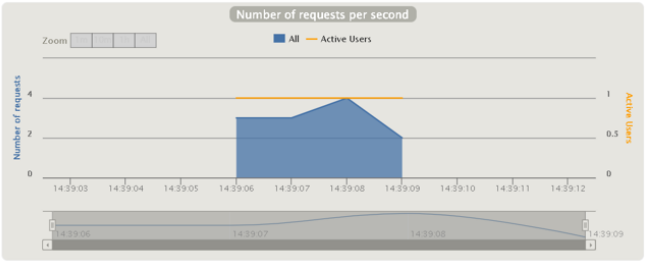
Распределение времени отклика:



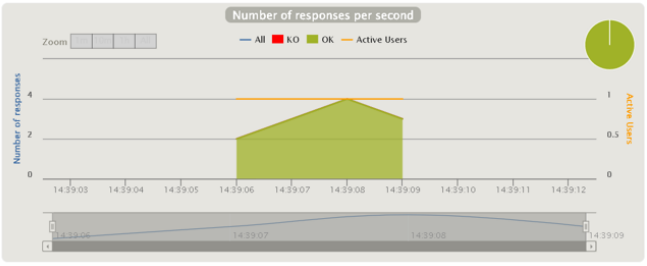
Процент времени отклика с течением времени:



Количество запросов в секунду:



Количество ответов в секунду:



По результатам проверки в условиях нагрузки не было выявлено никаких ошибок в системе, поэтому тестирование на выдержку было выполнено удачно.

Вывод

В рамках дипломной работы были рассмотрены основные проблемы, которые решает тестирование, а также протоколы современных web-приложений, которые необходимо тестировать под нагрузкой: REST API и технология WebSocket. Были проанализированы инструменты нагрузочного тестирования и их преимущества и недостатки.

Так же проведено нагрузочное тестирование чат-бота с использованием разработанной программы. Были получены результаты, которые позволили сделать выводы о производительности системы