ТИТУЛЬНЫЙ ЛИСТ

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc10067836)

[ГЛАВА 1. АНАЛИЗ ПРОБЛЕМНОЙ ОБЛАСТИ 7](#_Toc10067837)

[1.1.Постановка задачи в проблемной области 7](#_Toc10067838)

[1.2. Формирование требований к системе 8](#_Toc10067839)

[1.3. Информационная модель 14](#_Toc10067840)

[1.4. Математическая модель 15](#_Toc10067841)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ 16](#_Toc10067842)

[2.1. Проектирование базовой архитектуры 16](#_Toc10067843)

[2.2. Детальное проектирование системы 19](#_Toc10067844)

[2.3. Выбор инструментов 20](#_Toc10067845)

[ГЛАВА 3. РЕАЛИЗАЦИЯ 22](#_Toc10067846)

[3.1. Описание реализации 22](#_Toc10067847)

[3.2. Тестирование 24](#_Toc10067848)

[3.4. Проверка работоспособности 33](#_Toc10067849)

[Заключение 34](#_Toc10067850)

[Приложение 35](#_Toc10067851)

# ВВЕДЕНИЕ

На сегодняшний день много современных и даже высоконагруженных приложений создаются на базе интерфейса прикладного программирования (API[[1]](#footnote-2)). Несмотря на большие достижения в области информационных технологий, создание надежного приложения является сложной задачей. Этому способствует человеческий фактор разработчика, добавление новых функций, исправление участков кода в проекте. Тестирование — очень сложный процесс, который должен проводиться на всех этапах разработки. Данный процесс позволяет не только выявить возможные ошибки в разрабатываемом продукте, но и предоставить о нём актуальную информацию такую как, например, качество разрабатываемого программного обеспечения. Тестирование API является одной из главных задач в процессе контроля качества продукта, с помощью которого можно убедиться в правильности интеграции отдельных нововведений и в корректном соблюдении логики системы в целом. К сожалению, инструменты, предоставляющие возможность автоматизированного тестирования API, требуют наличие навыков программирования, либо большое количество времени на изучение самого инструмента, что негативно сказывается на длительности тестирования в условиях современных методологий разработки программного обеспечения.

Актуальность выбранной темы объясняется тем, что на данный момент большинство специалистов по контролю качества проверяют API сервиса вручную или используют автоматизированную систему тестирования, которая чаще всего настроена другими специалистами и корректно работает только с уникальным API выбранного программного продукта. Таким образом, создание инструмента, в котором будет сочетаться как гибкость и кроссплатформенность программы командной строки, так и легкость использования, положительно повлияет не только на скорость самого процесса тестирования, но и существенно уменьшит количество затрачиваемого времени для выявления изъянов в системе. К популярным аналогам относятся следующие продукты:

cURL. Кроссплатформенная программа командной строки, достоинствами которой является широкий функционал и возможность использования в ручном и автоматизированном тестировании web-сервисов, основанные на стандартных протоколах передачи данных и аутентификации. К серьезным недостаткам относятся высокие требования к знаниям принципов передачи данных, составления запросов и работа с командной строкой, а в случаях проведения автоматизированного тестирования возникает необходимость работы с одним из скриптовых языков.

SoapUI. Инструмент, который позволяет тестировать API, разработанные в соответствии с архитектурным стилем Representational State Transfer или работающие на основе протокола обмена структурированными сообщениям Simple Object Access Protocol. Предоставляет возможности по проектированию полноценной архитектуры сервиса, с заданием всех ресурсов, методов и демонстрационных данных для функционального и нагрузочного тестирования. К недостаткам относятся высокие требования к знаниям пользователя о протоколах и скриптовых языках, а также наличие платной и бесплатной версий с полным и урезанным функционалом соответственно.

Postman. Считается классическим инструментом для тестирования API. Имеет возможности по проектированию дизайна API, обладает набором средств интеграции с системами автодокументирования сервисов. Также присутствует наличие автоматического документирования. В платной версии возможно использование автоматического тестирования. Серьезными недостатками являются требования к системе, а в особенности к объему оперативной памяти, которые не позволяют использовать данный инструмент в бюджетных конфигурациях и специально выделенных для тестирования серверных машинах.

Традиционным и открытым программным обеспечение для нагрузочного и интеграционного тестирования является JMeter. Инструмент позволяет задавать уникальные значения в параметрах, имеет разнообразную визуализацию результатов в виде диаграмм и таблиц. Несмотря на широкий функционал и возможность дополнения инструмента новыми функциями с помощью плагинов, высокие требования к знаниям пользователя для создания параметров и запросов, а также громоздкий графический интерфейс являются серьезными недостатками.

С развитием микросервисных архитектур, выросло количество различных внешних и внутренних API по сравнению с тем, что было, например, 20 лет назад. Следовательно, возросла нагрузка на специалистов по контролю качества, которые тестируют данные интерфейсы, потому что компании могут разрабатывать собственные продукты на основе программного интерфейса или сотрудничать друг с другом, интегрируя множество сервисов, связанных между собой различными видами API. Сотрудники сталкиваются с огромными временными потерями при тестировании, что плохо сказывается на качестве программного обеспечения и влияет на удобство использования продукта конечным потребителем.

С точки зрения практической значимости, проектируемый инструмент для тестирования API благодаря своей гибкости в настройке и простоте использования может быть применен специалистами для тестирования большинства типов сервисов, которые используются в современных сервисах. Следует заметить, что уменьшение временных затрат на такие важные этапы, как настройка и изучение самого инструмента, положительно влияет на общее время тестирования программного интерфейса, а значит поиск и обнаружение дефектов в системе становится менее затратным процессом.

Целью работы является разработка и проведение пробной эксплуатации автоматизированной системы тестирования API, на базе потребностей организации ООО «Кибертоника».

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить внутреннюю документацию организации;
2. Определить функциональные и нефункциональные требования к системе;
3. Выбрать архитектуру и выполнить детальное проектирование системы;
4. Выбрать инструменты и реализовать функционал разрабатываемой системы;
5. Провести тестирование;
6. Написать документацию по системе;
7. Провести пробную эксплуатацию.

# ГЛАВА 1. АНАЛИЗ ПРОБЛЕМНОЙ ОБЛАСТИ

## 1.1.Постановка задачи в проблемной области

Специалисты организации Кибертоника, которая занимается разработкой облачного сервиса для финансовых организаций по предотвращению мошенничества за счёт использования методов машинного обучения, используют в своей работе гибкий подход к разработке программного обеспечения. Согласно этому подходу, после обновлений системы, которые происходят постоянно и с определенной периодичностью в течении недели, необходимо проводить регулярное тестирование системных компонент, нововведений и отдельных модулей. В связи с тем, что сервис, разрабатываемый данной организацией, имеет множество клиентских облачных сред, проведение интеграционного и регрессионного тестирования внутренних API является длительным и трудоёмким процессом. Кроме того, разнообразие и постоянное обновление типов и количества параметров запроса к конечной точке определенного модуля конкретной клиентской среды существенно усложняют процесс генерации тестового потока данных.

Таким образом, на базе потребностей организации, необходимо спроектировать и реализовать систему, которая позволит автоматизировать генерацию тестовых запросов к используемым в системе конечным точкам API и создание отчетов о результатах проведенного тестирования, отображающие актуальную информацию о состоянии продукта, а также упростить проведение специалистами процесса интеграционного и регрессионного тестирования внутренних API, используемых в системе.

## 1.2. Формирование требований к системе

В настоящей выпускной квалификационной работе при разработке требований к системе будем придерживаться классификации FURPS, которая разработана компанией Hewlett-Packard и была предложена в 1992 году. Согласно данной классификации, основными группами требований являются:

1. Функциональность системы (functionality);
2. Удобство использования (usability);
3. Надежность (reliability);
4. Производительность (performance);
5. Поддержка (supportability);

К функциональным требованиям системы относятся:

1. Построение статистической модели. Система должна уметь строить статистическую модель на основе входных данных, которые в свою очередь являются строками в файле. Строка файла является запросом к конкретному типу сервиса и типу API, содержащий конкретные параметры. Модель должна содержать следующие элементы: название модели (по умолчанию является совокупностью строк вида model\_дата запуска системы, например «model\_14-03-2019»), схему обращения к ресурсу, имя пользователя и его пароль, используемые для доступа к ресурсу, полностью прописанное доменное имя хоста в системе Domain Name Service (DNS) или его IP-адрес (представлен в виде четырёх групп десятичных чисел, разделенных точками, числа – целые, в интервале от 0 до 255), номер порта для подключения, уточняющая информация о месте нахождения ресурса (URL-путь). Модель также должна содержать параметры запроса, каждый из которых представляется в виде объекта, который должен включать в себя название параметра, его частоту встречаемости и список значений данного параметра, которые встретились в результате обработки запросов. Если какой-либо из вышеперечисленных элементов запроса не встретился ни разу, он должен быть помечен специальным образом и не участвовать в дальнейшей работе системы.
2. Сохранение и загрузка статистической модели. Система должна иметь возможность сохранять построенную статистическую модель в файл. Формат файла должен являться текстовым форматом обмена данными, основанный на JavaScript (JavaScript Object Notation, JSON). Структура файла должна быть унифицирована и документирована. Поля структуры должны быть логически правильно поименованы и понятны неопытному пользователю со знаниями английского языка и документации по системе. Система должна иметь возможность загрузки статистической модели из файла, описанного выше формата и структуры. Если пользователь, указал формат отличный от корректного или структура файла имеет синтаксические нарушения, система должна оповестить пользователя об ошибке и завершить работу.
3. Визуализация статистической модели. Система должна уметь визуализировать построенную модель и сохранять изображение в векторном формате. Изображение должно содержать все элементы построенной модели. Модель представляется как окружность, внутри которой помещено название модели. Элементы модели представлены в виде трех прямоугольников. В верхнем располагается название элемента, а в нижнем его значение. Изображение должно также содержать гистограмму распределения значений для каждого из параметров. При большом количестве уникальных параметров (больше 10), изображение должно содержать простое перечисление названий параметров в виде таблицы, которая должна состоять из 3 столбцов: название параметра, его тип (число или строка), пример параметра (первое значение из всего списка). Максимальное количество строк для таблицы - 50. При наличии числа параметров больше максимального, необходимо строить дополнительную таблицу, которая будет располагаться правее начальной таблицы.
4. Построение запроса на основе статистической модели. Система должна уметь генерировать запрос к заданной конечной точке API на основе статистической модели. Запрос должен содержать все необходимые элементы для обеспечения доступа к ресурсу, а именно: схему обращения к ресурсу, имя пользователя и его пароль, полностью прописанное доменное имя хоста в системе Domain Name Services (DNS) или его IP-адрес, номер порта для подключения, уточняющая информация о месте нахождения ресурса (URL-путь). После построения основной части запроса, система должна синтаксически корректно и случайно добавлять параметры, тип которых извлекается из статистической модели. Ясно, что если некоторый параметр из модели имеет целый тип значений в 90% случаях из всех обработанных входных данных, то этот тип выбирается как основной тип значений для текущего параметра.
5. Отправка запросов и обработка ответов от API. Система должна осуществлять работу с сервером средствами протокола передачи данных HyperText Transfer Protocol (HTTP/HTTPS). Исходя из спецификации данного протокола, система также должна уметь не только обрабатывать ответ от сервера, но и доступно расшифровывать для пользователя. Таким образом, при отправке запросов на сервер, система должна выводить информацию о текущем процессе работы, в которой должно содержаться следующее: строка сгенерированного запроса к серверу, а именно к конечной точке API и ожидаемый ответ. Следующая строка вывода должна содержать пару, состоящая из краткого ответа и его подробного описания в стандарте HTTP/HTTPS. Успешным запросом является такой запрос, ожидаемый ответ от сервера которого равен фактическому. По умолчанию, время ожидания ответа от сервера равно 10 секундам. Если по истечению этого времени ответ не был получен, то в строку вывода вместо пары значений, необходимо вставить информацию о недоступности сервера. Если сервер не отвечает более чем на 5 запросов, необходимо прекратить отправку запросов и завершить работу системы.
6. Построение статистических отчетов. Система должна уметь строить статистические отчеты и сохранять их в формате JSON, который должен иметь унифицированную структуру и содержать следующую информацию: краткая информация о конечной точки API, дата запуска системы, общее количество отправленных запросов к API за время работы системы, количество успешных и неуспешных запросов, а также их процентное соотношение друг к другу.
7. Логирование запросов к серверу в текстовый файл. При отправке и приёме ответов от сервера система должна хранить лог запросов в текстовом файле, который будет содержать строки, состоящие из локальной даты отправления запроса, тела запроса, ожидаемого ответа, фактического ответа и локальной даты приёма ответа от сервера. Запись в файл должна производиться асинхронно, чтобы не нарушать соединение и логику работы системы с сервером. По умолчанию, при каждом запуске системы лог-файл перезаписывается заново, стирая предыдущие данные, при этом, происходит изменение имени файла.
8. Хранение конфигурационных параметров системы в отдельном файле. Система должна работать на основе некоторых базовых и глобальных параметров, которые в свою очередь должны быть вынесены в файл с форматом Yes Another Markup Language (YAML). Данный файл должен содержать достаточное количество параметров системы, чтобы пользователь мог не только изменить необходимую конфигурацию системы без доступа к исходному коду, но и сохранять текущую конфигурации параметров, для дальнейшего использования и передачи.

Далее были описаны требования к удобству использования системы:

1. Пользовательский интерфейс. Взаимодействие пользователя с системой должно обеспечиваться средствами текстового интерфейса командной строки, который в свою очередь должен соответствовать задачам пользователя и технологии, быть понятным, логичным и защищенным от воздействия от человеческих ошибок при вводе данных с клавиатуры. Выводимая информация является протоколом работы автоматизированной системы, степень детализации которой регулируется глобальным параметром из конфигурационного файла и представляет из себя двухуровневую иерархию:

1 уровень. Глобальный параметр имеет значение «system». На этом уровне система должна выводить только основную информацию о процессе работы, к которой относится: идентификатор уровня (является строкой с значением «[SYSTEM]»), сгенерированный запрос к API и результат тестирования (строка, значениями которой могут быть либо «OK» – при совпадении ожидаемого результат с фактическим, либо «FAILED» – при несовпадении);

2 уровень. Глобальный параметр имеет значение «all». На текущем уровне, добавляются дополнительные выводы информации: дата и время отправки сгенерированного запроса, сам запрос к API и подробный ответ от сервера (код ошибки и его расшифровка). На любом уровне в конце работы системы необходимо выводить общую статистику по тестированию: общее количество запросов, количество успешных и неуспешных запросов к серверу, ответ от сервера, ошибки (если таковые имеются) и процентное соотношение успешных запросов к общему количеству.

1. Справочная информация в системе. Автоматизированная система должна содержать внутреннюю справочную информацию. Доступ к ней осуществляется через ключ, передаваемый в команду запуска, значение которого равно «--help». При запуске с текущим ключом, система должна вывести следующую краткую информацию: основное назначение системы, версию системы, имя разработчика, названия дополнительных файлов, прилагаемые к системе (конфигурационный, лог-файл и др.) и их краткое описание, а также ссылка на эксплуатационную документацию.
2. Наличие эксплуатационной документации. Система должна содержать руководство пользователя, которое располагается в открытом доступе сети Интернет, а также находится рядом с файлами исходного кода программы. Руководство должно содержать информацию о системе (версия, лицензия, имя разработчика), краткий принцип работы программы(допускается использование диаграмм), пошаговую инструкцию по установке и запуску системы, подробную информацию о всех функциях, которые представлены в системе (краткое описание, тип входных параметров, тип выходных параметров), описание структуры конфигурационных файлов, возможные ошибки и способы их решения.

Надежность автоматизированной системы определяется надежностью функциональных модулей. Система должна обеспечивать cохранение работоспособности при отказе или выходе из строя по любым причинам одного из модулей системы, а также при отсутствии или разрыве Интернет-соединения. В обоих случаях, пользователю необходимо выводить информацию о некорректной работе системы и завершить работу. Система должна иметь следующие средства восстановления при ошибках во входных данных: возможности обработки ошибочных ситуаций, наличие тестов для проверки допустимых значений входных данных, наличие системы контроля полноты и корректности входных данных, наличие проверки параметров и адресов по диапазону их значений, наличие обработки граничных результатов, наличие обработки неопределенностей(например, деление на 0 и др.).

Требования к производительности тесно связаны с требованиями в архитектуре, так как производительность сильно зависит от конфигурации системы и мощности аппаратного обеспечения. В связи с тем, что автоматизированная система будет использоваться на разных аппаратных платформах, необходимо, чтобы система корректно работала на следующем минимальном аппаратном обеспечении:

Операционная система: unix-подобные системы (Ubuntu 14.07 LTS, macOS Sierra 10.12.6 и выше)

Оперативная память: 1Gb, 1600MHz, LPDDR3

Сетевое соединение: 128 Kb/s и больше

## 1.3. Информационная модель

Информационная модель должна включать такое формализованное описание предметной области, которое будет понятно любому специалисту.

На основании выделенных сущностей и результатов анализа проблемной области была спроектирована ER-диаграмма в нотации Crow’s Foot (рисунок 1).

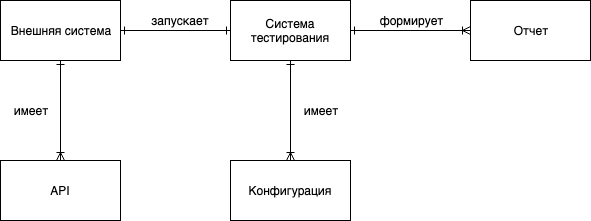


Рис. 1 Информационная модель

## 1.4. Математическая модель

Что делает система? – тестирует;

Тестирует что? – API web-сервиса;

Тестирует чем? – множеством запросов;

Множество запросов состоит из отдельных запросов. Следовательно сам процесс тестирования, сводится к процессу построения запросов.

Что такое запрос?

|  |  |
| --- | --- |
| С точки зрения IT | С точки зрения математики |
| https://host.com/a?data=test  Очевидно, что  https – протокол;  host.com – хост;  a – узел;  data=test – наименование параметра и его значение;  Т.е. запрос – это строка, состоящая из протокола,хоста, узлов, параметров и их значений.  Есть множество разнообразных запросов из входных данных, значит мы можем собрать статистику по элементам.  Есть статистика – есть распределение элементов.  Создание запроса – это выбор элементов относительно их распределения с учетом зависимостей.  Для простоты разобьем создание на два этапа:   1. Создание пути 2. Выбор параметров и значений |  |

С точки зрения математики

https://host.com/a?data=test

Очевидно, что https,host.com,a,data=test = это какие-то значения.

У нас есть статистика, следовательно значения представляются в виде случайных величин.

Вероятность величины P(x) = значение из подсчитанной статистики с помощью формулы A/B, где А – количество вхождений в множество запросов, B – общее количество запросов.

Будем рассматривать следующую ситуацию:

Основной путь уже сгенерирован: https://host.com/a/b.../f/g

Осталось присоединить к этому пути параметр и значение. Будем считать(для простоты), что параметр зависит только от последнего узла(в нашем случае g), а значение параметра зависит от самого имени параметра.

Получаем 3 случайных величины: название узла(*x*), название параметра(y) и значение параметра(*z*).

*p(x,z) = p(x)(z)* так как вы упростили зависимость: считаем, что значение параметра(*z*) НЕ зависит от значения узла(x). Величины *x* и *y* – независимы

Следовательно, мы можем описать распределение величин с помощью формулы условной вероятности:

- означает, что название узла влияет на распределение параметра

– аналогично, название параметра влияет на распределения значения

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ

## 2.1. Проектирование базовой архитектуры

Проектирование архитектурного решения системы является одним из самых серьезных и ответственных этапов на протяжении всего жизненного цикла программного обеспечения. Это обуславливается возможностью появления скрытых ошибок в системе и проблем масштабирования, которые могут проявится только на самых поздних этапах жизненного цикла. При выборе архитектуры мы будем опираться на следующее приоритетные характеристики: относительно простое расширение и изменение разрабатываемой системы, а также простота и понимание самой архитектуры. Перед проектированием уникальной базовой архитектуры системы необходимо провести анализ уже существующих стандартных архитектур по классификации Гарлана и Шоу. При анализе были выделены подходящие на первый взгляд стандартные решения для разрабатываемой системы:

1. Архитектура каналов и фильтров;
2. Образец проектирование Interpreter;
3. Уровневая архитектура;
4. Клиент-серверная архитектура.

Очевидно, что при детальном рассмотрении решений, образец проектирования Interpreter, который относится к категории виртуальных машин, не может использоваться в разрабатываемой системе из-за низкой эффективности данного решения: во внутренних системах организации ООО «Кибертоника» не существует и не будет существовать в ближайшее время систем, использующих данное архитектурное решение. Клиент-серверная архитектура несмотря на неоспоримое преимущество (низкое сцепление между модулями системы) относительно перечисленных выше решений, также не является подходящим выбором: сервер представляется как единый модуль и любое добавление или изменение серверных функций влечет за собой полное изменение клиентских частей, а также появление скрытых ошибок при комплексном взаимодействии системы, что негативно будет сказываться на работе системы. Также, данное архитектурное решение является наиболее избыточным для разрабатываемой системы, которая подразумевается, как легкое и консольное приложение. При рассмотрении архитектуры каналов и фильтров, которая основывается на потоках данных, было получено, что текущий вариант является наилучшим архитектурным решением для разрабатываемой системы, так как каждая функция может быть представлена отдельным фильтром, тем самым обеспечивается быстрая замена функций без существенных изменений в самой системе.

Таким образом, была спроектирована высокоуровневая модель системы, на основе архитектуры каналов и фильтров, которая изображена на рисунке 1 в виде диаграмм потоков данных в нотации Гейна-Сарсона.

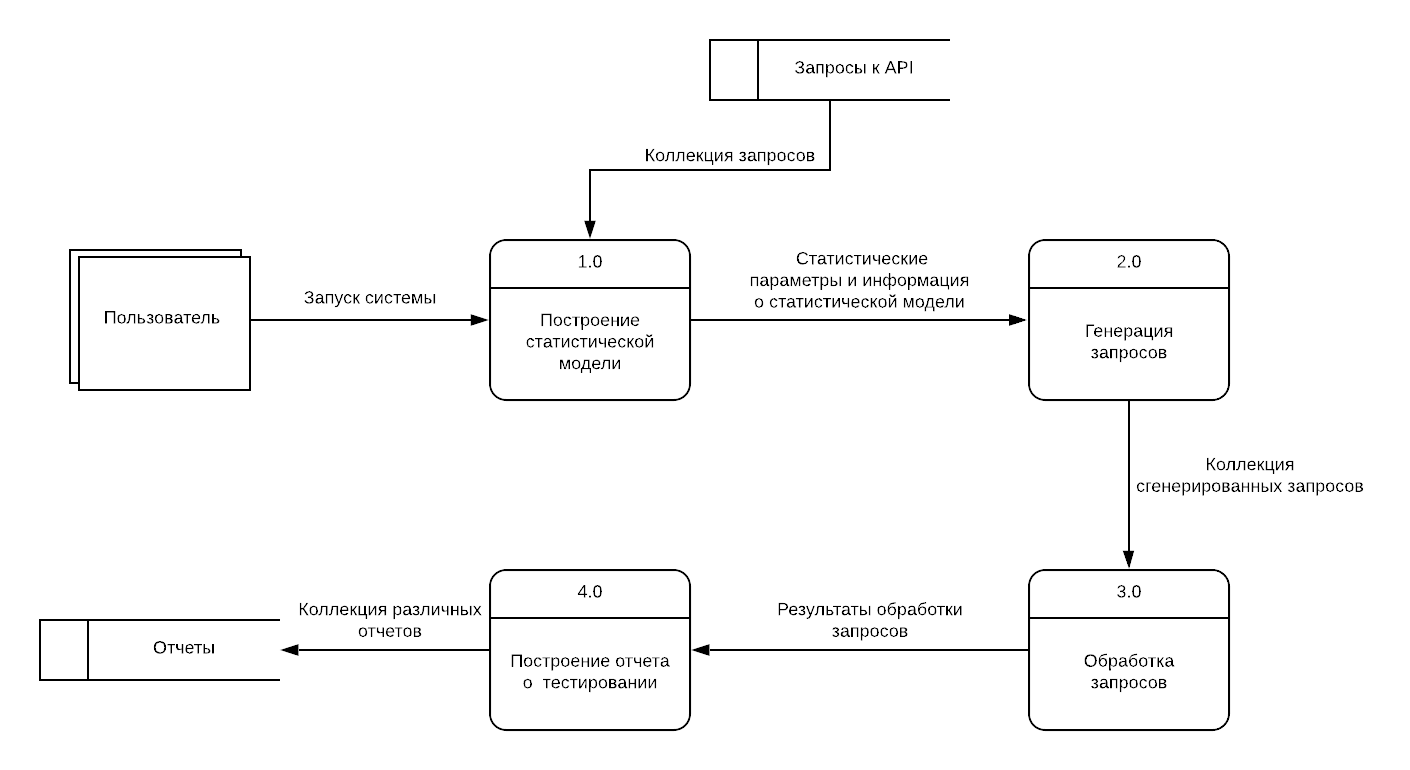


Рис. 2 Диаграмма потоков данных системы

После проектирования, обязательным условием архитектуры является удовлетворение заданным функциональным требованиям. Анализ показал, что выбранное архитектурное решение полностью обеспечивает выполнение функциональных требований, изображенных на рисунке 2 в виде диаграммы вариантов использования.



Рис. 3 Диаграмма вариантов использования

## 2.2. Детальное проектирование системы

(черновик)

Система разрабатывается как открытое программное обеспечение на базе потребностей организации ООО «Кибертоника». Следовательно, исходный код и справочная информация системы должны быть размещены на какой-либо распределенной системе управления версий под лицензией MIT2.

В связи с тем, что этап проектирования базовой архитектуры завершился выбором уже существующего архитектурного решения типа «канал-фильтр», было принято решение не использовать шаблоны проектирования и объектно-ориентированный подход, так как описанные решения лишь усложняют реализацию и поддержку всей системы в целом.

Система представляет собой набор фильтров, каждый из которых является отдельным программным модулем. Реализация фильтра-модуля, отвечающий за обработку ответов от сервера зависит от выбранного языка программирования и не состовляет трудности. При анализе проектирования модуля построения статистической модели, была найдена функция, отвечающая за построение дерева структуры web-сервиса, которая может существенно усложнить этап реализации системы. Следовательно, было принято решение реализовать данный функционал на основе языка описания алгоритмов – псведокоде.

[псевдокод]

На данном этапе проектирования системы наибольший интерес будут представлять модули, отвечающие за построение статистической модели и генерацию отчетов по тестированию указанного web-сервиса. Рассмотрим проектирование модуля для построения статистической модели. МодульПроцесс детального проектирования не только описывает технические детали и нюансы, но и может предоставить уже готовые спроектированые решения алгоритмов в виде блок-схем.

[блок-схема]

## 2.3. Выбор инструментов

При выборе языка программирования для реализации описанной системы мы будем учитывать следующие факторы: существующий опыт разработки на языке, удобство работы с сложными структурами данных, разнообразное количество сторонних библиотек для визуализации данных, кроссплатформенность и возможность интеграции в уже существующие системы организации. При анализе существующих языков программирования, были выбраны следующие кандидаты: Java, Go и Python, версии 3.7.

Java. Язык программирования общего назначения, который используется для настольных, сетевых, мобильных и корпоративных приложений. Вокруг языка существует не только огромное сообщество, но и целая экосистема инструментов для программирования на Java и выполнения абсолютно разных по своему типу задач. Однако, несмотря на удовлетворение многим вышеперечисленным факторам, данный язык программирования не подходит для реализации, так как систему, написанную на этом языке будет сложно интегрировать в уже существующие подсистемы организации, что является решающим фактором для принятия решения об отказе от этого инструмента.

Go. Компилируемый многопоточный язык программирования. Высокая производительность, простота синтаксиса и широкий выбор библиотек являются серьезными преимуществами над остальными кандидатами. Несмотря на внушительный набор достоинств, Golang имеет и недостатки: неудобное управление не только зависимостями, но и возникающих ошибок, не свойственный тип задачи для языка, так как в первую очередь, Golang создавался для написания серверной части сервиса.

Python. Высокоуровневый язык программирования общего назначения, имеющий огромное количество математических библиотек, а так же библиотек для работы с данными и их визуализации. Обладает минималистичным синтаксисом и динамической типизацией, что является преимуществом, для работы с необработанными массивами данных. Однако требует написания модульных тестов и повышенного внимания к процессу разработки из-за возможности появления скрытых ошибок.

После тщательного анализа кандидатов, было принято решение использовать язык программирования Python, версии 3.7. Решающими факторами в пользу этого языка стали: существенный опыт разработки относительно опыта работы на других кандидатах и возможность быстрой интеграции с другими системами организации.

Для визуализации статистической модели и других данных системы было решено использовать самую популярную библиотеку, которая подходит для описанных задач – Matplotlib, версии 3.0.3, в совокупности с библиотекой NumPy, версии 1.11.2, которая добавляет поддержку больших многомерных массивов и матриц, вместе с больной библиотекой высокоурвневых математических функций для операций над этими массивами и матрицами.

В связи с тем, что система будет работать с такой структурой данных, как дерево, то для ускорения разработки базового функционала системы, было принято использовать уже описанную реализацию деревьев, которую предоставляет библиотека AnyTree, версии 2.6.0.

# ГЛАВА 3. РЕАЛИЗАЦИЯ

## 3.1. Описание реализации

(черновик)

На текущем этапе, была реализована структура данных, в которой хранится вся информация о работе системы и ее данных. А именно: параметры конфигурации, входные данные, пути, очищенные от массива входных данных, техническая информация о запросах(протокол, хост и т.д.), корень дерева структуры API сервиса и данные о статистическом модели. Заполнение такой структуры данных происходит в модуле model.py, и реализуется следующими функциями:

load\_configuration() – функция, которая производит чтение файла конфигураций. Если процесс чтения прервался, или файла не оказалось, то система будет использовать настройки по умолчанию;

load\_data() – функция, которая производит чтение основного файла входных данных, который содержит список запросов к API;

parse\_data() – производит обработку и очистку входных данных;

build\_tree() – функция построения дерева структуры API сервиса;

calc\_statistic() – расчет статистических данных и распределения для узлов, параметров и их значений;

Все функции реализованы с обработчиками исключений таким образом, что при возникшем сбое, система подскажет пользователю текст ошибки и название функций, в которой эта ошибка появилась.

Вторым по важности модуем является stochastic\_process.py, реализующий построение запросов, которые в свою очередь, подчиняются некоторому распределению, вычисленному на этапе работы модуля model.py

{описание функций модуля}

Заключающим модулем в описаниии реализации является модуль report.py, который производит создание статистических отчетов в текстовом и графическом формате. Каждая функция имеет 2 типа реализации: для создания текстового(имеет приставку «txt») или для графического(имеет приставку «pic») отчетов.

Модуль имеет следующие функции( знак «\*» - заменяет приставку):

\*\_methods() – генерирует круговую диаграмму, которая отображает распределение методов запроса(GET,POST,HEAD и т.д.)

\*\_params\_histogram() – генерирует гистограмму параметров, которые используются в запросах

\*\_tree\_visualize() – выполняет визуализацию дерева структуры API

\*\_level\_stats\_histogram() – генерирует гистограмму, которая отображает распределение конечных точек на каждом уровне API.

## 3.2. Тестирование

Процесс тестирования программного обеспечения заключается в исследовании программного продукта, в результате которого можно установить степень соответствия реального поведения программного обеспечения и его требуемого поведения. Следовательно, существуют разнообразные уровни и виды тестирования. Во время проведения процесса тестирования разрабатываемой системы, необходимо использовать следующие уровни тестирования:

1. Модульное;
2. Интеграционное;
3. Системное;
4. Приёмочное;

Разработка программы велась с помощью техники Test Driven Development, которая основана на простом цикле: написать тестовый случай для участка кода, написать сам код, реализующий функционал тестового случая и провести улучшение кода при необходимости. Данная техника была выбрана из-за того, что безопасность написания программ на выбранном языке программирования для реализации системы требует тщательного прокрытия большого количества участков кода из-за появления скрытых ошибок, которые связаны, в большинстве случаев, с динамической типизацией языка. Таким образом, каждый модуль уже имеет готовый набор тестовых случаев, который обеспечивает минимальное покрытие имеющихся функций. При написании тестов для всех модулей, расчет значения среднего коэффицента покрытия кода, показал, что значение примерно равно 71.3%. В связи с достаточно высоким уровнем этого значения, было принято провести повторный анализ тестовых случаев, их улучшение. При проведении заключительного расчета значения тестового покрытия кода, было отмечено, что при изменении старых тестовых случаев на более детальные, значение коэффицента снизилось до 54.1%. Что является достаточно среднем показателем для выбранного языка программирования и текущего уровня тестирования.

На следующем уровне тестирования происходит проверка связи между модулями, а также взаимодействие с различными частями операционной системы. В связи с тем, что модули уже были разработаны, во время проведения интеграционного тестирования, будем использовать подход «снизу вверх», в соответсвии с которым, все модули, процедуры и функции собираются воедино и тестируются. Таким образом, были выявлены следующие критические для системы дефекты:

1. Некорректное обращение к полям структуры переданного параметра при передачи данных между модулем работы с сервером и модулем, для создания отчетов;
2. При окончании работы модуля подсчета статистической модели, данные не передавались в следующий модуль генерации запросов;
3. Запись неверного формата данных (целое число, отображающее количество, вместо ожидаемого коэффицента соотношения) в модуле расчета статистической модели;
4. Неверная передача конфигурационных параметров в модуль для создания статистических отчетов;
5. Недостаток переданных параметров между модулем генерации запросов и модулем работы с сервером;
6. Ошибка в названии параметра. Модуль генерации запросов запрашивает несуществующее поле;
7. Некорректная работа с файловой системой в модулей создания отчетов;
8. При создании статистических отчетов, файлы создаются в неверной директории.

После устранения вышеперечисленных дефектов, был неоднократно проведен процесс тестирования на текущем уровне. Таким образом, во время интеграционного тестирования был выявлен примерно 31 дефект, 10 из которых, являлись критическими. Было решено закончить этап интеграционного тестирования в связи с тем, что с проведением каждой разнообразной проверкой стало существенно падать количество обнаруженных дефектов.

Основной задачей системного тестирования является проверка всех требований к системе, при этом на этом уровне можно обнаружить дефекты, связанные с непредусмотренными сценариями использования. При выполнении системного тестирования был использован подход на базе требований, в соответствии с котором, для каждого требования пишутся тестовые случаи. Основные тестовые случаи, для проверки некоторых функциональных требований перечислены в таблицах 1-5.

|  |  |  |
| --- | --- | --- |
| Проверка построения статистической модели | | |
| Действие | Ожидаемый результат | Результат теста |
| Запустить систему с подготовленными данными | Был создан не пустой файл с именем «model» и форматом JSON | Пройден |
| Поля структуры из файла model.json не пустые | Пройден |
| Имена полей структуры файла model.json имеют не пустое значение | Пройден |
| Структура файла синтаксически не нарушена(может прочитать любой текстовый редактор JSON форматов) | Пройден |
| Существуют специально помеченные поля, которые не учавствуют в обработке | Пройден |

Таблица 1. Тестовые случаи для проверки построения статистической модели

|  |  |  |
| --- | --- | --- |
| Проверка сохранения и загрузки статистической модели | | |
| Действие | Ожидаемый результат | Результат теста |
| Запустить систему с подготовленными данными | Создан не пустой файл | Пройден |
| Запустить систему с флагом «загрузить модель». Файл существует. | Отчеты системы созданы на основе данных сохраненного файла | Пройден |
| Запустить систему с флагом «загрузить модель». Файла не существует. | Система корректно завершила свою работу и вывела сообщение и проблеме с загрузкой файла. | Пройден |
| Запустить систему. Во время процесса расчета статистической модели аварийно завершить работу | Система не создала файл. Не перезаписала уже существующий. | Не пройден |

Таблица 2. Тестовые случаи для проверки сохранения и загрузки статистической модели

|  |  |  |
| --- | --- | --- |
| Проверка визуализации статистической модели (построение статистических отчетов) | | |
| Действие | Ожидаемый результат | Результат теста |
| Запустить систему с подготовленными данными | Были созданы отчеты в векторном формате | Не пройден |
| Визуализация отчетов корректна, фигуры и графики отображаются нормально | Не Пройден |
| Данных отчетов совпадают с данными в файле model.json | Пройден |
| Помеченные поля не учавствуют в визуализаци | Пройден |

Таблица 3. Тестовые случаи для проверки визуализации статистической модели

|  |  |  |
| --- | --- | --- |
| Проверка генерации запросов на основе статистической модели | | |
| Действие | Ожидаемый результат | Результат теста |
| Запустить систему с подготовленными данными | В файле model.json поле «synthetic\_req» должно быть не пустое | Пройден |
| Структура поля «synthetic\_req» - словарь с вложенными полями «reqs» и «count» | Пройден |
| Поле «reqs» не пустое и представляет из себя список элементов(запросов), поле «count» отображает количество элементов | Пройден |
| Элементы в списке поля «reqs» представляют из себя декартово произведение всех узлов на всех уровней | Пройден |
|  | Элементы в списке поля «reqs» синтаксически правильно построены, без нарушения целостности URL-запроса | Пройден |

Таблица 4. Тестовые случае для проверки генерации запросов

|  |  |  |
| --- | --- | --- |
| Проверка отправки запросов и обработки ответов от сервера | | |
| Действие | Ожидаемый результат | Результат теста |
| Запустить систему с подготовленными данными | Система успешно взаимодействует с сервером на основе протокола HTTP/HTTPS | Пройден |
| Обработка ответов от сервера корректна и соответствует списку кодов состояния протокола HTTP | Пройден |
| Корректное завершение соединения с сервером | Не пройден |
| Запустить систему с подготовлеными данными. Локальный сервер выключить. | Система через 10 секунд после отправки запроса объявила о истекшем времени | Пройден |
| После 5 запросов, на которых сервер не ответил, система завершила работу с сообщением для пользователя | Пройден |
| Запустить систему с подготовленными данными. Локальный сервер включен. | Система успешно соединяется с сервером | Пройден |
| Все запросы из поля «synthetic\_reqs» были отправлены на сервер | Пройден |
| Количество обработанных ответов равно количеству запросов. Значения не искажены | Пройден |

Таблица 5. Тестовые случаи для проверки отправки запросов и обработки ответов от сервера

Таким образом, при проведении системного тестирования, было обнаружено 4 дефекта, которые делятся по серьезности следующим образом: 3 критических и 1 незначительный. Все дефекты исправлены и проведено неоднократное тестирование системы на соответствие к предъявленным требованиям.

Важным этапом при проведении системного тестирования является проверка на соответствие не только функциональным, но и нефункциональным требованиям. Одним из таких является требование к производительности. В связи с тем, что система не подразумевается как инструмент для стрессового тестирования API, наибольший интерес представляет анализ поведения системы при увеличении объема входных данных. Следовательно, необходимо провести объемное тестирование системы.

Для получения результатов по тестированию, анализ которых поможет определить примерное поведение системы, необходимо решить следующие задачи:

1. Разработать модель нагрузки;
2. Сконфигурировать тестовый стенд;
3. Разработать вспомогательные системы или скрипты;
4. Провести процесс тестирования.

Проведенный анализ сформированных требований показал, что к системе не предъявлены жесткие рамки ограничений на производительность, но в то же время ясно, что система будет использоваться специалистами по тестированию, которым необходимо получить результаты работы системы за адекватное время.

Моделью объемного тестирования будет считать постепенное увеличение количество уникальных запросов к одноуровневому API с уникальными узлами при неизменной конфигурации параметров системы. Время работы модуля, отвечающий за отправку запросов на сервер учитываться не будет. Создание статистических отчетов настроено на графический режим без создания HTML-страницы.

Тестовый стенд представляет из себя машину со следующими аппаратными и программными характеристиками: процессор Inter Core i5 с частотой 2,3 ГГц, 2 ядра, размер кэша 2 уровня для каждого ядра равен 256 кБайтам, кэша 3 уровня равен 4 мБайтам, объем оперативной памяти 8 гБайт с частотой 2133 МГц формата LPDDR3, видеочип Intel Iris Plus Grpahics 640 интегрированный в процессор, твердотельный накопитель с файловой системой APFS, операционная система macOS Mojave, версии 10.14.4.

После проведения процесса тестирования, был построен график (рисунок 3) отображающий зависимость времени работы системы с модулем генерации статистических отчетов и без него, от количества входных уникальных запросов. Анализ показал, что основную часть времени работы занимает визуализация статистических данных. Таким образом, при работе со сложными структурами сервиса имеет смысл разграничить работу модулей на разные по мощности машины, а благодаря удачно выбранной архитектуре, требуется минимальное внесение изменений в исходной код для реализации такого разграничения.

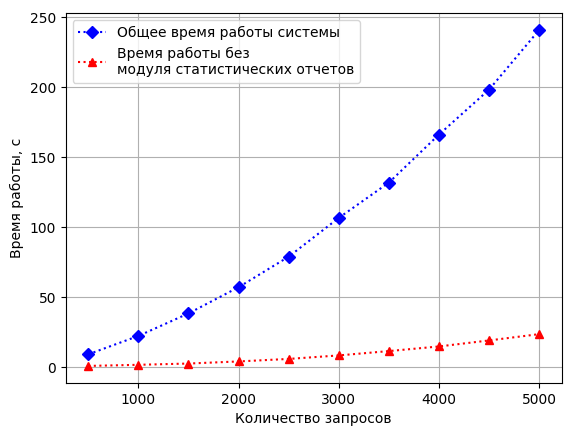


Рис. 4 График зависимости времени работы системы от количества входных уникальных запросов с модулем генерации статистических отчетов и без.

## 3.4. Проверка работоспособности

# Заключение

# Приложение

1. Интерфейс прикладного программирования (Application Programming Interface, API) - набор готовых процедур и функций, классов и структур. [↑](#footnote-ref-2)