**Отчет по выполнению задания по тестированию.**

**Блок 1: Подготовка и выполнение начальных задач.**

1. Создано тестовое окружение на базе предоставленного сервера.
2. Логи production\_log.csv были загружены и сохранены для анализа.
3. Был написан парсер для логов на Java, который считывает данные из файла logs/production\_log.csv и извлекает необходимую информацию о запросах.

* **Класс Main** - класс является точкой входа в программу. Его задача - инициализация и запуск процесса парсинга. Используемый метод: **main(String[] args):** Создает массив релевантных методов, инициализирует объект парсера, запускает парсинг лог-файла и выводит статистику.
* **Класс Parser -** класс отвечает за чтение и парсинг лог-файлов.
  + RequestStatistics stats: Объект для хранения статистики запросов.
  + Parser(String[] relevantMethods): Конструктор, инициализирующий объект статистики с релевантными методами.
  + void parseLogFile(String filePath): Читает лог-файл построчно, извлекает информацию о запросах и добавляет её в статистику.
  + RequestStatistics getStatistics(): Возвращает объект статистики.
* **Класс RequestStatistics -** класс хранит статистику запросов и предоставляет методы для работы с ней.
  + int totalCalls: Общее количество вызовов.
  + Map<String, Integer> methodCounts: Количество вызовов для каждого метода.
  + String[] relevantMethods: Массив релевантных методов.
  + RequestStatistics(String[] relevantMethods): Конструктор, инициализирующий объект статистики.
  + void addRequest(String methodPath): Добавляет информацию о запросе в статистику.
  + int getTotalCalls(): Возвращает общее количество вызовов.
  + Map<String, Integer> getMethodCounts(): Возвращает количество вызовов для каждого метода.
  + String[] getRelevantMethods(): Возвращает массив релевантных методов.
  + boolean isRelevantRequest(String methodPath): Проверяет, является ли запрос релевантным.
  + void printStatistics(): Выводит статистику запросов.
  + void printRPS(): Выводит расчетную интенсивность запросов (RPS).

### Результат:

* **Количество вызовов:**
  1. POST /api/signDoc: 37289 (10.02%)
  2. GET /api/sendMessage: 93114 (25.02%)
  3. GET /api/getMessage: 93114 (25.02%)
  4. POST /api/addDoc: 74385 (19.98%)
  5. GET /api/getDocByName: 74329 (19.97%)
* **RPS:**
  1. POST /api/signDoc: 10.36
  2. GET /api/sendMessage: 25.87
  3. GET /api/getMessage: 25.87
  4. POST /api/addDoc: 20.66
  5. GET /api/getDocByName: 20.65

1. Были записаны скрипты:

### SignDoc Requests - скрипт выполняет запрос на подписание документа. Он отправляет файл на сервер с использованием метода POST. Заголовки и параметры запроса настроены для корректной отправки файла в формате multipart/form-data.

### Message Requests

### SendMessage Transaction - скрипт выполняет запрос на отправку сообщения. Сообщение генерируется динамически с использованием JSR223 PreProcessor (Generate Random Message) и передается в качестве параметра запроса. Заголовки запроса для принятия ответа в формате JSON.

#### **GetMessage Transaction** - скрипт выполняет запрос на получение сообщения. Запрос направляется на сервер для получения последнего отправленного сообщения. Заголовки настроены для принятия любого типа данных.

### AddDoc Requests - скрипт выполняет запрос на добавление документа. Документ создается динамически с использованием JSR223 PreProcessor (UniqueFileName Generator), который генерирует файл с уникальным именем и сохраняет его в указанной директории. Запрос отправляет файл на сервер.

### GetDocByName Requests - скрипт выполняет запрос на получение документа по имени. Имя документа генерируется и сохраняется в переменной с использованием JSR223 PreProcessor (UniqueFileName Generator). Запрос отправляется на сервер для получения информации о документе по его имени.

1. Базовый профиль нагрузки:

На основе парсинга логов был создан базовый профиль нагрузки. Рассчитаны значения RPS, пейсинг и количество пользователей для каждого метода:

Формулы для расчета:

* Пейсинг (сек) = 1 / RPS
* VUS = Количество вызовов / (3600 \* RPS)