**Задание**

Необходимо с помощью Jmeter покрыть тестами представленные эндпоинты. Распределить нагрузку согласно таблице:

1. Получить токен пользователя - один раз для каждого потока (/api/tester/login)
2. Регистрация игрока - 10% (/api/automationTask/create)
3. Запросить данные профиля игрока - 70% (/api/automationTask/getOne)
4. Запросить данные всех пользователей и отсортировать их по имени (/api/automationTask/getAll) - 30%
5. Нагрузка должна ступенчато увеличиваться - 5 rps, 10 rps, 15 rps. Длительность ступени 3-5 минут.
6. Удалить всех ранее созданных пользователей (/api/automationTask/deleteOne/{id})
7. Запросить список всех пользователей и убедиться что он пустой (/api/automationTask/getAll)

По итогам тестирования должен быть сформирован csv-файл с результатами.

**Отчет о выполнении**

1. Проверяем в Postman работоспособность эндпоинтов согласно тех. Заданию  
     
   Изображение выглядит как текст, снимок экрана, программное обеспечение, число

   Автоматически созданное описание

В ходе проверки обнаруживаем, что запрос данных игрока (п.3) (/api/automationTask/getOne) не работает ни по email (согласно API документации), ни по id.  
  
Регистрация с email=email@email.ru  
  
Изображение выглядит как снимок экрана, текст

Автоматически созданное описание  
  
Запрос на зарегистрированного пользователя:  
  
**GET** <https://testslotegrator.com/api/automationTask/getOne?email=email@mail.ru>  
  
**Ожидаемый результат:** 200 OK + данные зарегистрированного пользователя по email=email@mail.ru  
  
**Фактический:**{

"statusCode": 404,

"message": "Cannot GET /automationTask/getOne?email=email@mail.ru",

"error": "Not Found"

}

Также обнаруживаем, что повторная регистрация доступна на уже существующий email.   
  
Также обнаруживаем, что в пунктах 2-4 указано о распределении нагрузки пользователей в соотношении 10% + 70% + 30%, что не корректно, так как в сумме дает 110%, но всего мы имеем 100% пользователей. Jmeter будет пытаться распределить нагрузку, но мы получим не совсем корректные данные, так как не будем знать как именно он ее распределил.  
  
Также обнаруживаем, что если в задании подразумевается в п.3 запросить данные созданного в п.2 игрока, то мы получим ошибку, так как 10% пользователей только пройдут регистрацию, а 70%-10%=60% будут пытаться запрашивать данные незарегистрированных пользователей. Помимо этого запросы с использованием распределения нагрузки выполняются параллельно, а в задании возможно подразумевается последовательное выполнение действий. Регистрация, а затем только запрос данных зарегистрированного пользователя. Таким образом для теста зафиксируем запрос на 1 определенный email, для проверки нагрузки по этому энд-поинту.  
  
2. Для получения токена для дальнейших запросов (п.1) в Jmeter добавляем “Thread group login” с количеством выполнения 1 пользователь 1 раз как указано в тех задании.  
  
Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание  
  
Так как для ступенчатого распределения нагрузки был установлен плагин “Stepping Thread Group” получение токена было вынесено в отдельную Thread Group, так как требуется выполнение всего 1 раз, а не поступенчатое.   
  
Для сохранения токена из этого запроса и передачи его далее в “Stepping Thread Group” мы используем BeanShell PostProcessor (который запишет токен в переменную скриптом) и далее используем его в “Stepping Thread Group” c помощью BeanShell PreProcessor и скрипта который также добавит сохраненную ранее переменную.

Получаем и сохраняем токен в Thread Group login  
Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание  
  
Устанавливаем скриптом сохраненный ранее токен из Thread Group login  
в “Stepping Thread Group” с помощью BeanShell PreProcessor  
  
  
Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

1. Настраиваем ступени с помощью “Stepping Thread Group” согласно тех. заданию следующим образом:   
     
   Длительность первой ступени 3 минуты, нагрузка 900 пользователей /180 секунд = 5 rps,   
   Далее длительность второй ступени 3 минуты, нагрузка 1800 пользователей /180 секунд = 10 rps,

Далее длительность третьей ступени 3 минуты, нагрузка 2700 пользователей /180 секунд = 15 rps,  
  
  
Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

1. Добавляем запрос на регистрацию  
   HTTP registration, так как нам доступна регистрация по одному email (возвращаются уникальные id пользователей) – то используем тело запроса согласно спецификации, Throughput устанавливаем 10%:  
   Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

   Автоматически созданное описание  
     
   Также в регистрации добавляем скрипт в BeanShell PostProcessor, который запишет все уникально полученные id пользователей и сохранит в property, так как они будут необходимы для удаления всех пользователей по id в п.6 задания   
     
   Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

   Автоматически созданное описание  
     
   5. Устанавливаем запрос GET <https://testslotegrator.com/api/automationTask/getOne?email=lidiiapetrovski@gmail.com> на определенный email (в идеале можно было бы сделать генерацию емейлов при регистрации и подстановку их потом в этот запрос), но так как регистрация 10%, а запрос - 70%, то нагружаем по одному определенному email.  
     
     
     
   6. Создаем запрос на данные всех пользователей и сортировку по имени  
     
   GET <https://testslotegrator.com/api/automationTask/getAll?sort=name>  
     
   Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

   Автоматически созданное описание  
     
   Throuput устанавливаем согласно заданию = 70%  
     
   7. Общее количество запросов исходя из задания:  
     
   общее количество запросов в секунду (R): R = RPS \* количество пользователей  
     
   На первой ступени (RPS = 5, 3 минуты): 900 пользователей \* (3 минуты \* 60 секунд/минуту) = 810,000 запросов  
     
   На второй ступени (10 RPS, 3 минуты): 1800 пользователей \* 10 RPS \* (3 минуты \* 60 секунд/минуту) = 1,080,000 запросов  
     
   На третьей ступени (15 RPS, 3 минуты): 2700 пользователей \* 15 RPS \* (3 минуты \* 60 секунд/минуту) = 1,215,000 запросов  
     
   Общее количество запросов, которое будет выполнено в ходе тестирования, составляет 3,105,000 запросов. Из них 10% на регистрацию – значит 310500 пользователей должно быть зарегистрировано.  
     
   Создаем новый отдельный Thread group с количеством потоков 310500 для удаления ранее созданных пользователей. (будем считать, что у нас база изначально пустая, но фактически пункт 7 задания (удаление всех пользователей) выполнить будет проблематично, так как у нас удаление только по одному id, а ранее в базе уже существовали пользователи с другим id, которые нам неизвестны)  
     
   И добавляем запрос на удаление  
   DELETE https://testslotegrator.com/api/automationTask/deleteOne/{id}  
     
   Также добавляем к этому запросу BeanShell PreProcessor со скриптом импорта ранее сохраненных при запросе регистрации id пользователей записанных в properties  
     
   Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, компьютер

   Автоматически созданное описание  
     
   8. Добавляем запрос на всех пользователей и проверку Response Assertion что ответ пустой и юзеров нет.   
     
   GET <https://testslotegrator.com/api/automationTask/getAll?sort=name>  
   Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

   Автоматически созданное описание  
   Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

   Автоматически созданное описание

9. Для сохранения результатов в csv файл добавляем View Results in Table и указываем файл csv для сохранения /Users/lidiakarandasova/Downloads/apache-jmeter-5.5/results/testResults.csv  
  
10. Важный нюанс для настройки – так Thread Group с http запросами в jmeter выполняются параллельно, а нам необходим последовательный их запуск, поэтому добавим delay таймеры задержки запуска 10 секунд перед Stepping Thread Group для того, чтобы получить успеть получить токен в Thread Group login (первой группе), а также паузу 10 секунд + 9 минут = 9 минут и 10 секунд на запуск deleteAll, который будет длится в зависимости от пропускной способности какое-то время и 9 минут 10 секунл + время на удаление всех пользователей на паузу до запуска последней группы запроса пустого списка, таким образом мы получим последовательный запуск thread group.

Constant Timer задержка 10 секунд чтобы успеть получить токен:  
Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, компьютер

Автоматически созданное описание  
Установим таймер 9 минут 10 секунд = 550000 мс перед запуском удаления всех пользователей по id  
Изображение выглядит как текст, программное обеспечение, компьютер, Мультимедийное программное обеспечение

Автоматически созданное описание  
  
Установим таймер 11 минут 20 секунд = 680000 мс перед запуском последнего пункта проверки на пустой список. Тут берем около 2 минут на удаление 310500 пользователей, учитывая что пропускная способность у нас не менее 2500 пользователей в секунду.   
  
  
Изображение выглядит как программное обеспечение, текст, Мультимедийное программное обеспечение, компьютер

Автоматически созданное описание  
  
  
Результаты: на основании тестов замечено, что при данных количествах пользователей и rps, пропускная способность ниже, чем подаваемая нагрузка