Разработка фреймворка для проведения нагрузочного тестирования

Автор: Ветчинникова Алина Евгеньевна, АВТ-615

Руководитель: Якименко Александр Александрович, к. т. н., доцент

Новосибирск 2020

Актуальность

- Автоматизация тестирования является актуальной задачей для компаний сферы разработки ПО, так как позволяет сократить как время на проведение тестирования, так и количество необходимых для проведения тестирования специалистов.

- Использование инструментов для проведения автоматического тестирования требует от тестировщиков знания языков программирования. Это увеличивает сложность перехода от ручного тестирования к автоматическому.

Постановка задачи

Цель работы: разработать фреймворк, который позволит сократить временные издержки и трудозатраты при проведении нагрузочного тестирования, а также упростить анализ результатов тестирования

Задачи:

- Рассмотреть существующие на рынке решения, используемые для проведения нагрузочного тестирования
- Спроектировать фреймворк, упрощающий взаимодействие с одним из инструментов нагрузочного тестирования
- Разработать модуль генерации тестовых сценариев
- Разработать модуль генерации отчетов
- Провести тестирование разработанного фреймворка

Используемые средства и технологии

- Язык программирования Java
- Система автоматической сборки Maven
- Среда разработки IntelliJ IDEA
- Gatling— среда тестирования нагрузки и производительности с открытым исходным кодом, основанная на Scala, Akka и Netty
- Allure Framework – популярный инструмент построения отчётов автотестов, упрощающий их анализ

Результаты работы

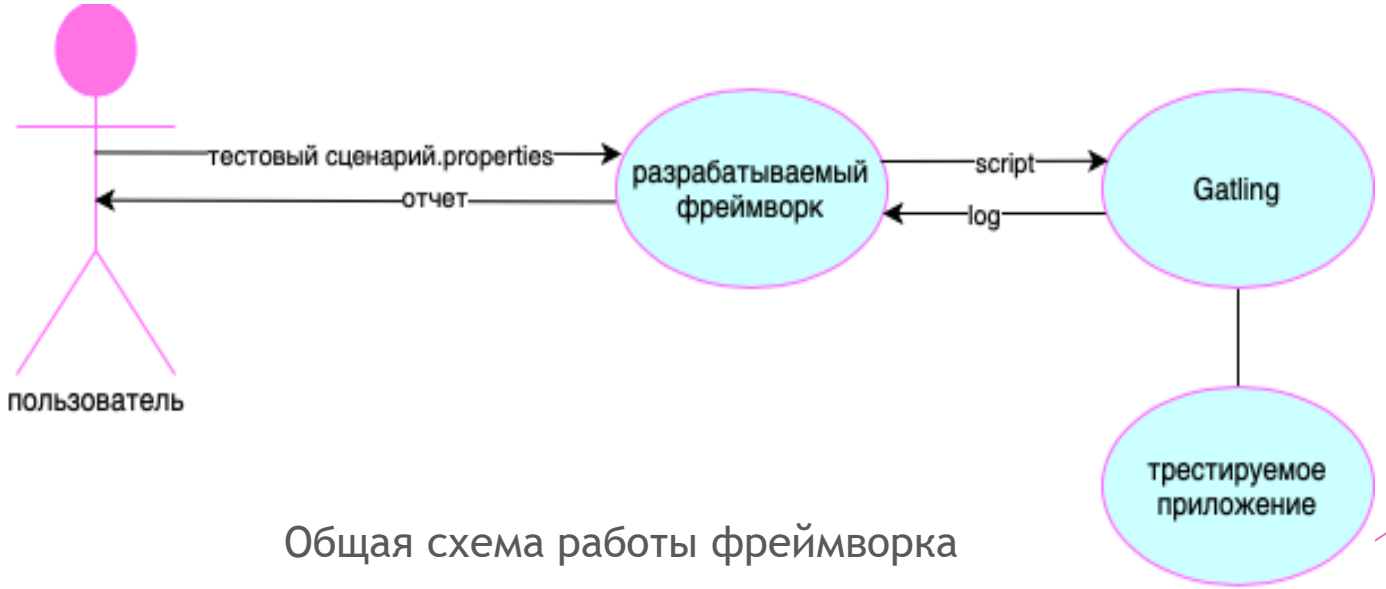
- Спроектирован фреймворк для проведения нагрузочного тестирования веб-приложений
- Разработан модуль генерации скриптов
- Разработан модуль генерации отчетов
- Произведено модульное тестирование

Проектирование

Основная цель данного фреймворка – предоставить пользователям (тестировщикам), в особенности не имеющим знаний в области программирования на скриптовых языках, доступ к инструменту нагрузочного тестирования Gatling, который требует написания скрипта для создания сценариев.

Задачи:

- преобразование тестового сценария из properties файла в scala-скрипт
- преобразование Gatling log в отчет
- отображение отчета



Общая схема работы фреймворка

Модуль генерации скриптов

- Properties файл подается в качестве аргумента при запуске класса *ScriptGenerator* Из этого файла при помощи метода *load* класса *ConfigLoader* загружаются конфигурации и сохраняются в HashMap

- Для каждой конфигурации определен процессор, выбирающий данные из HashMap при помощи метода *findConfiguration* и создающий классы конфигураций, определенные в пакете *object*

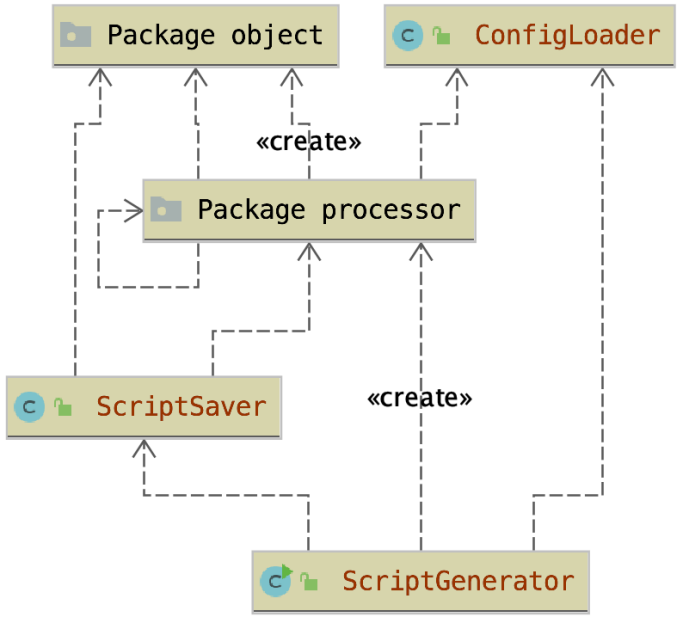


Диаграмма модуля генерации скриптов

- Созданные конфигурации *ScriptBuilder* оформляет в скрипт, который сохраняется с помощью метода *save* класса *ScriptSaver*.

Модуль генерации отчетов

- Лог-файлы подаются в качестве аргумента при запуске класса *GatlingToAllure*.
- Из полученных файлов выделяются запросы, ответы и сессии
- Далее выделенные объекты передаются в *ResponseProcessor* и *SessionProcessor* соответственно
- *JsonFormater* используется для выделения тела запроса или ответа. Выделенные объекты сохраняются в отдельные файлы

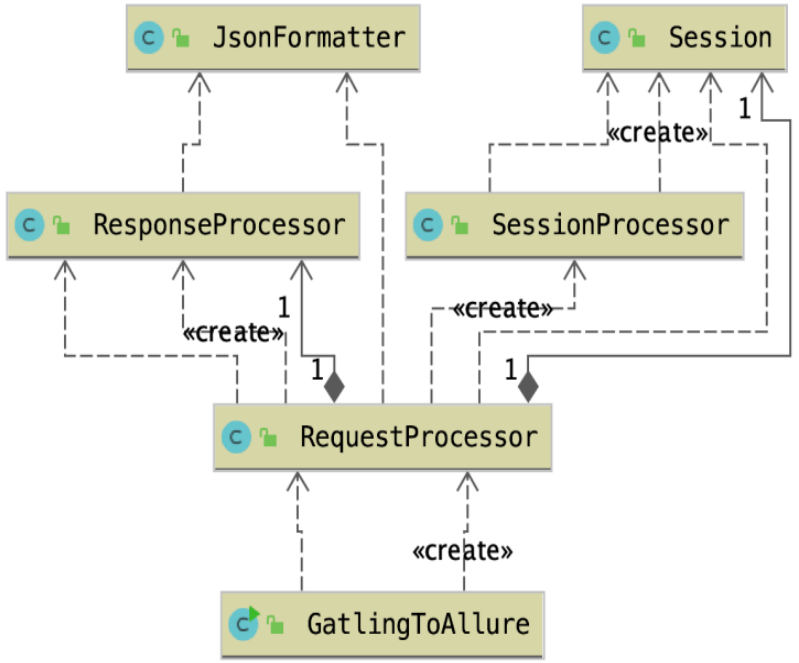


Диаграмма классов модуля генерации отчетов

Тестирование

Проведено модульное тестирование при помощи JUnit5:

- разработаны 5 properties файлов и 5 скриптов, тестовый класс, сравнивающий сгенерированные скрипты с разработанными
- разработан тестовый класс с 5 тестовыми сценариями для всесторонней проверки работы модуля генерации отчетов
- дополнительно разработаны тестовые классы для отдельной проверки корректности работы функции конвертации логов:
  - Подготовлены 2 файла – один содержит удачно прошедший тест, другой – упавший тест. Данные файлы имитируют записи логов. В процессе тестирования проверяется правильность анализа и выделения необходимых частей из лог-файлов

- Все тесты пройдены успешно, что говорит о корректной работе проверяемого функционала разработанного продукта

Заключение

- Изучена тема нагрузочного тестирования веб-приложений. Рассмотрены популярные инструменты для проведения нагрузочного тестирования
- Спроектирован и разработан фреймворк для проведения нагрузочного тестирования веб-приложений
- Фреймворк состоит из 3х модулей – модуля генерации скриптов, модуля генерации отчетов и модуля для создания Maven-плагина
- Произведено модульное тестирование разработанного продукта
- В модуле генерации скриптов реализована только поддержка тестов для http-запросов, когда в Gatling также есть возможность тестировать WebSockets, SSE (Server Sent Events), JMS, MQTT. Реализация конвертации для перечисленных видов запросов возможна в следующих версиях продукта

Использование

Разработанный фреймворк, может быть использован широким кругом как разработчиков и специалистов автоматизированного тестирования, так и тестировщиков, не имеющих навыков программирования, с помощью которого можно ускорить и облегчить проведение нагрузочного тестирования веб-приложений, привести результаты к единому формату и упростить анализ отчетов

Разработка фреймворка для проведения нагрузочного тестирования

Автор: Ветчинникова Алина Евгеньевна, АВТ-615

Руководитель: Якименко Александр Александрович, к. т. н., доцент

Новосибирск 2020

Нагрузочное тестирование

Нагрузочное тестирование — исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов

Задачи нагрузочного тестирования:

- оценка производительности и работоспособности приложения на этапе разработки
- оценка производительности и работоспособности приложения на этапе выпуска новых релизов
- оптимизация производительности приложения
- подбор соответствующей для данного приложения аппаратной платформы и конфигурации сервера

Gatling

Gatling— среда тестирования нагрузки и производительности с открытым исходным кодом, основанная на Scala, Akka и Netty

Достоинства:

- работает с любой операционной системой
- может выполнять свои сценарии в разных облаках для тестирования
- есть плагины для Gradle и Maven
- асинхронная архитектура
- создание информативных отчетов

Недостатки:

- не позволяет равномерно распределить нагрузку между разными машинами
- отсутствует возможность горизонтального масштабирования
- API изменялись радикальным образом
- формат отчетов

Формат данных

- properties — текстовый формат и одноимённое расширение имени файла. Применяется в технологиях, связанных с Java (где имеется класс Properties с методами, позволяющими писать в файл и читать из него), для хранения конфигурационных параметров прикладного ПО (пар «ключ» — «значение»)
- JSON – текстовый формат обмена данными, представляет собой (в закодированном виде) одну из двух структур:
  - Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимая), значением — любая форма.
  - Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность
- Log-файл (протокол, журнал, лог; англ. log) — файл с записями о событиях в хронологическом порядке, простейшее средство обеспечения журналирования. Различают регистрацию внешних событий и протоколирование работы самой программы — источника записей (хотя часто всё записывается в единый файл)

Технические требования

- Программный продукт разрабатывается в виде консольного приложения и для удобства использования оформляется в Maven плагин.

  В плагине пользователь может указать директорию с исходными properties файлами и директорию для сохранения отчетов.

- После подключения плагина пользователю доступны 2 аннотации:
  - @generateScript
  - @createReport

- **Основные функции программы:**
  - принять на вход тестовый сценарий в виде properties файла
  - преобразовать входной файл в тестовый скрипт
  - сохранить скрипт в директорию simulations
  - получить результаты тестирования в виде log файлов
  - преобразовать log в файлы для allure
  - передать файлы в allure
  - сохранить allure report

Разработка фреймворка

Разрабатываемый фреймворк состоит из 3х частей:

- модуль генерации скриптов – отвечает за создание scala-скрипта с параметрами запуска тестов, определенными в properties файле. Сгенерированный скрипт сохраняется и в дальнейшем может быть использован для запуска тестов при помощи Gatling

- модуль генерации отчетов – отвечает за анализ логов с результатами тестирования Gatling, их конвертацию в файлы для allure и генерацию allure report

- модуль оформления фреймворка в плагин Maven

Модуль генерации отчетов

- При проведении тестирования, все результаты записываются в логи Gatling. Для дальнейшего анализа логи должны быть определенного формата. При подключении плагина в директории resources генерируется файл logback.xml, в котором описан паттерн записи логов и расположение лог-файлов

- Лог-файлы подаются в качестве аргумента при запуске класса *GatlingToAllure*. Лог-файл имеет простую для разбора структуру – время запроса, уровень и источник логирования, соответствующие паттерну описанному на рисунке 7; информация о сценарии, имя запроса, статус; информация о сессии; сам HTTP запрос, его заголовок; HTTP ответ, статус, заголовок и тело ответа

Модуль оформления фреймворка в плагин Maven

- Модуль содержит 2 класса *ConvertPropertiesToScript* и *ConvertLogsToAllureData*, которые расширяют класс *AbstractMojo*, что позволяет с помощью аннотаций Mojo и переопределения метода *execute* обозначить классы целью Maven и сгенерировать зависимости для плагина

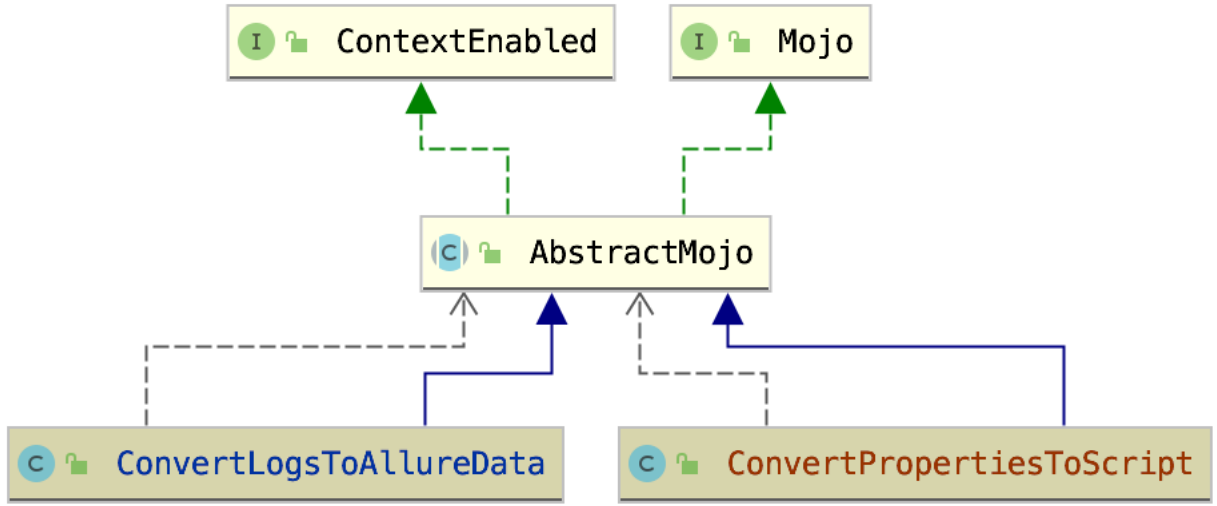


Диаграмма классов модуля оформления фреймворка в плагин Maven

Модуль оформления фреймворка в плагин Maven

- В каждом классе переопределен метод *execute*, который использует StreamAPI для выбора properties и log файлов

- Аннотация @*generateScript* вызывает метод *execute* объекта класса *ConvertPropertiesToScript*, где генерируются скрипты, а аннотация @*createReport* вызывает метод *execute* объекта класса *ConvertLogsToAllureData*, где генерируются JSON и txt файлы для создания Allure report

Использование фреймворка

- Подключение – в файле pom.xml указывается плагин FrameForGatling

```
<plugin>
    <groupId>com.vetchinnikova</groupId>
    <artifactId>FrameForGatling </artifactId>
    <version>1.0-SNAPSHOT</version>
    <configuration>
        <pathToLogs>/gatling-demo-project/target</pathToLogs>
        <pathToResults>target/</pathToResults>
        <multipleSimulationLogs>true</multipleSimulationLogs>
    </configuration>
</plugin>
```

- Тестовый сценарий

Для примера использования разработанного фреймворка был протестирован сайт НГТУ. Разработан тестовый сценарий «Просмотр расписания занятий», содержащий шаги:

1. Открыть домашнюю страницу сайта НГТУ
2. Открыть вкладку «Обучающимся»
3. Нажать «Расписание занятий»
4. Навести на «Факультет автоматики и вычислительной техники»
5. Нажать «АВТ-615»

Тестовый сценарий

Данный сценарий, отображенный в виде REST-запросов:

1. GET https://www.nstu.ru
2. GET https://www.nstu.ru/studies
3. GET https://www.nstu.ru/studies/schedule/schedule_classes
4. GET https://www.nstu.ru/studies/schedule/schedule_classes/schedule?group=%D0%90%D0%92%D0%A2-615

config.properties

```
url=baseURL("https://www.nstu.ru")
header1=acceptHeader("text/html, */*")
header2=acceptCharsetHeader("UTF-8")
header3=acceptEncodingHeader("gzip, deflate")
options=disableFollowRedirect
status=200
scenarioName=scenarioName("ScenarioViewSchedule")
http1=http("MainPage")
http1Verb=get("/")
http2=http("StudiesPage")
http2Verb=get("/studies/")
http3=http("ScheduleClassesPage")
http3Verb=get("/studies/schedule/schedule_classes/")
http4=http("GroupPage")
http4Verb=get("/studies/schedule/schedule_classes/schedule/")
http4Options=queryParam("group", "%D0%90%D0%92%D0%A2-615")
openInjection=atOnceUsers(50)
```

Конфигурации в файле config.properties

- базовый URL(https://www.nstu.ru), на котором будут строиться все запросы
- данные для проверки заголовков – тип данных ответа (text/html, */*), кодировка (UTF-8), метод кодирования, примененный к телу объекта (gzip, deflate)
- код статуса для проверки ответа (200)
- 4 http запроса с указанием названий запросов, методов, относительных адресов, передаваемых параметров
- параметр запуска – количество пользователей (50)

Генерация скриптов

- Для запуска генерации скриптов необходимо указать аннотацию *@generateScript*, после выполнения которой скрипты появятся в директории simulations

```
import io.gatling.app.Gatling
import io.gatling.core.config.GatlingPropertiesBuilder

object Engine extends App {

 @generateScript

 Gatling.fromMap(new GatlingPropertiesBuilder().build)
}
```

Использование аннотации generateScript

```
class nstuLoadTest extends Simulation {
  val protocol: HttpProtocolBuilder = http
    .baseURL( url = "https://www.nstu.ru")
    .acceptHeader( value = "text/html, */*")
    .acceptCharsetHeader( value = "UTF-8")
    .acceptEncodingHeader( value = "gzip, deflate")
    .check(status is 200)

  val scn = scenario( scenarioName = "ScenarioViewSchedule")
    .exec(
      http( requestName = "MainPage")
        .get("/")
    )
    .exec(
      http( requestName = "StudiesPage")
        .get("/studies/")
    )
    .exec(
      http( requestName = "ScheduleClassesPage")
        .get("/studies/schedule/schedule_classes/")
    )
    .exec(
      http( requestName = "GroupPage")
        .get("/studies/schedule/schedule_classes/schedule/")
        .queryParam("group", "%D0%90%D0%92%D0%A2-615")
    )

  setUp(
    scn.inject(atOnceUsers( users = 1
  ).protocols(protocol)
}
```

Сгенерированный скрипт

Создание отчета

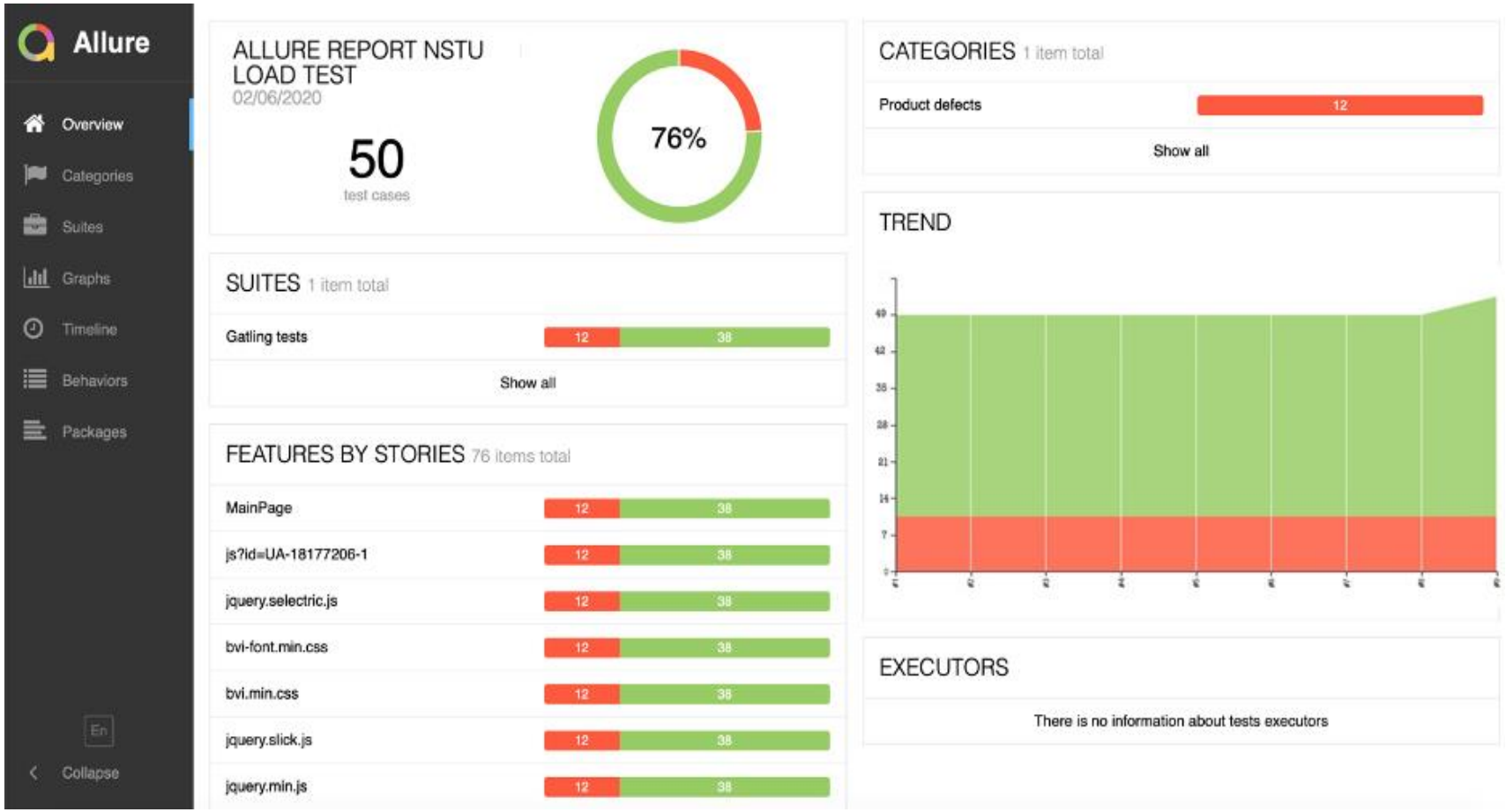
- Для запуска генерации отчета необходимо указать аннотацию @*createReport*, после выполнения которой данные для отчета появится в директории *allure-results*

```
import io.gatling.app.Gatling
import io.gatling.core.config.GatlingPropertiesBuilder

object Engine extends App {

 @generateScript

 Gatling.fromMap(new GatlingPropertiesBuilder().build)

 @createReport

}
```

Использование аннотации createReport

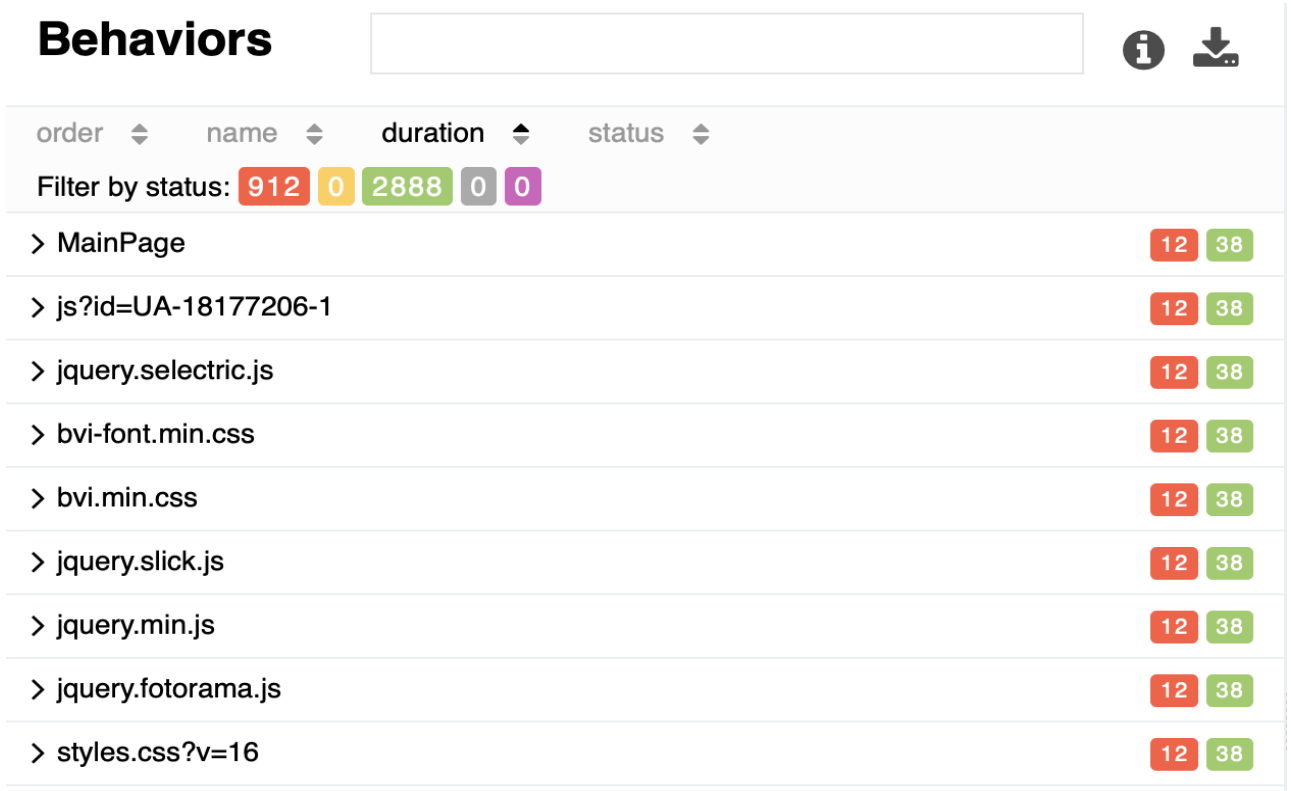
Результаты тестирования

При заданных конфигурациях лишь 38 из 50 тестов прошли успешно, что составляет 76% от общего количества



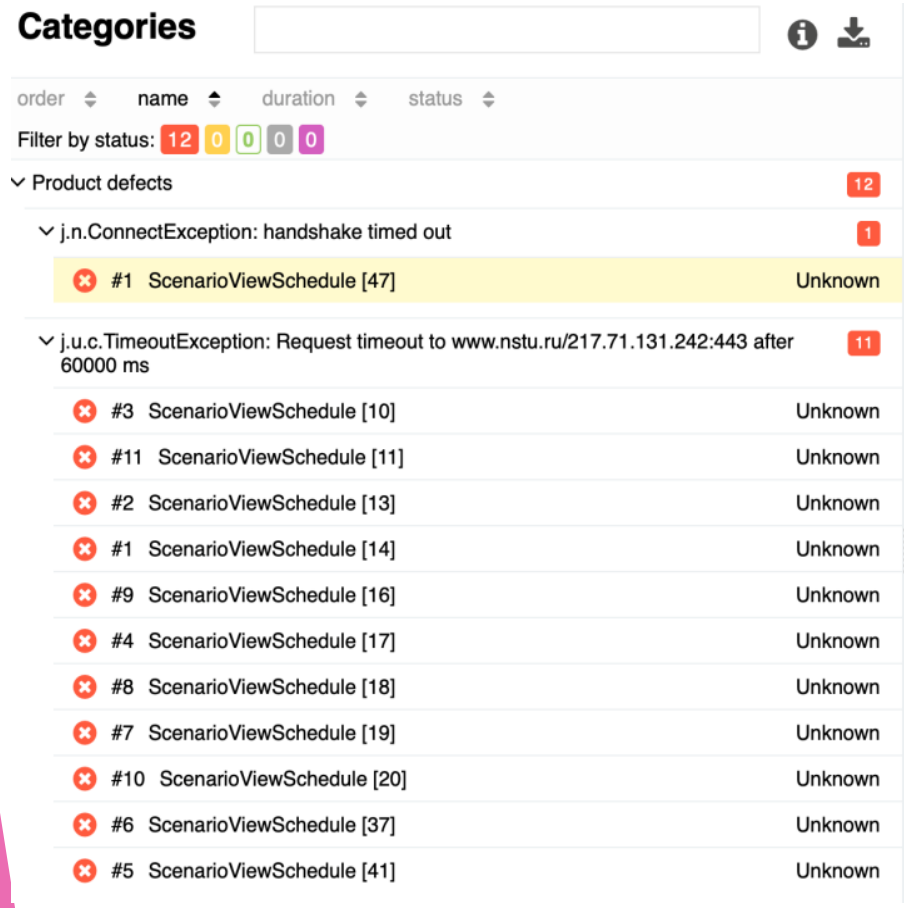
Информация по запросам

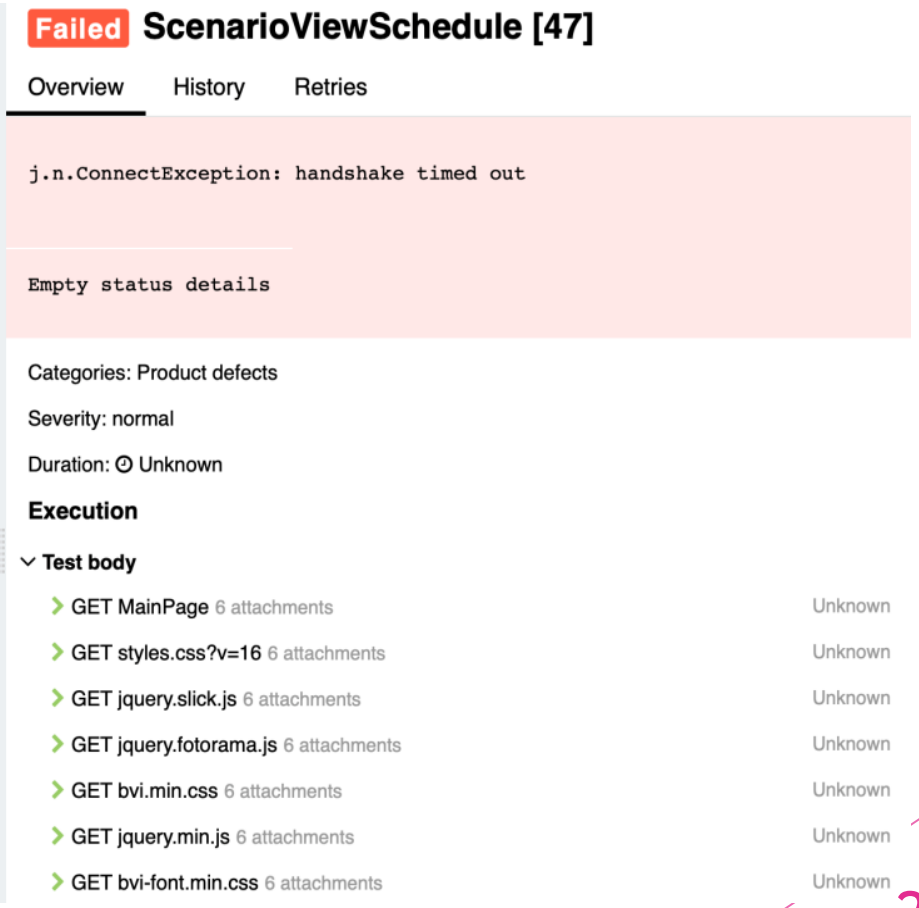
Из 3800 запросов 2888 прошли успешно, 912 завершились с ошибкой



Информация об ошибках

Для просмотра причин ошибок, необходимо открыть подробную информацию об упавшем запросе. В данном случае причиной всех ошибок является слишком долгое время ответа





Результаты тестирования

Также можно получить информацию о тестовом прогоне в графическом виде: статус прогона, распределение тестов по их критичности, длительности прохождения, перезапусках, категориях дефектов