Testing mobile apps in the cloud (BrowserStack) using Appium and Cucumber in Ruby

Overview

In this tutorial, we will create a test in Cucumber for Ruby in order to validate a simple mobile application using Appium and BrowserStack for cloud testing.

The test (specification) is initially created in Jira as a Cucumber Test complemented with a Pre-Condition; later on it is exported using the UI or the REST API and run in BrowserStack mobile devices.



Please note

Within this tutorial, only one Test Execution will be used; it will contain one Test Run with all the results for the different used devices. Thus, the overall test run status will be affected by the results made for all the devices.

Instead of this approach, a different one could be creating a Test Execution per each device. The steps would need to be slightly different. namelly the submission process would need to use the standard or multipart Cucumber REST API endpoints, for each result file corresponding to each device. This approach would give the ability to take advantage of Test Environments (more info in Working with Test Environments).

Requirements

- Install Ruby (or JRuby)
- install all dependencies using "bundle install", on the "android" sub-folder

Description

This tutorial is based on BrowserStack's own tutorial for Appium/Cucumber/Ruby.

You may start by cloning the repository https://github.com/browserstack/cucumber-ruby-appium-app-browserstack .

```
git clone https://github.com/browserstack/cucumber-ruby-appium-app-browserstack
cd android
```

We'll use the "android" example folder as basis and the "parallel" task that runs the tests in parallel.

We have to make some changes in order to make Cucumber generate a distinct JSON report per each device.

Rakefile was "hacked" in order to process the devices configured for the "parallel" task and related configuration file (i.e. *config/parallel.config.yml*).

The number of parallel jobs must be equal to the number of configured devices.

Rakefile

```
require 'rake'
require 'parallel'
require 'cucumber/rake/task'
require 'yaml'

Cucumber::Rake::Task.new(:cukesingle) do |task|
  ENV['CONFIG_NAME'] ||= "single"
  task.cucumber_opts = ["--format=pretty -f json -o results.json", 'features/single.feature']
end

task :default => :single

Cucumber::Rake::Task.new(:local) do |task|
  task.cucumber_opts = ["--format=pretty -f json -o results.json", 'features/local.feature',
'CONFIG_NAME=local']
end

task :single, [:device] do |task,args|
        device = (args[:device] || "").gsub(' ','_')
        cuke_task = Cucumber::Rake::Task.new
        cuke_task.cucumber_opts = ["--format=pretty -f json -o device_#{device}.json", 'features/single.
feature']
        cuke_task.runner.run
end

task :parallel do |t, args|
  @num_parallel = 2

  Parallel.map([*1..@num_parallel], :in_processes => @num_parallel) do |task_id|
    ENV["TASK_ID"] = (task_id - 1).to_s
    ENV['name'] = "parallel_test"
    ENV['CONFIG_NAME'] = "parallel"

        TASK_ID = (ENV['TASK_ID'] || 0).to_i
        CONFIG_NAME = ENV['CONFIG_NAME']
        CONFIG = YAML.load(File.read(File.join(File.dirname(__FILE__), "./config/#{CONFIG_NAME}.config.yml")))
        caps = CONFIG['browser_caps'][(task_id-1)]
        ENV['DEVICE'] = caps['device']

    Rake::Task["single"].invoke(caps['device'])
    Rake::Task["single"].reenable
  end
end

task :test do |t, args|
  Rake::Task["single"].invoke
  Rake::Task["single"].reenable
  Rake::Task["local"].invoke
  Rake::Task["parallel"].invoke
end
```

You need to configure the BrowserStack user/key along with desired browser capabilities/devices.

config/parallel.config.yml

```
server: "hub-cloud.browserstack.com"
user: "youruser"
key: "yourkey"

common_caps:
  "build": "cucumber-browserstack"
  "browserstack.debug": true

browser_caps:
  -
    "device": "Google Pixel"
    "app": "bs://6c31566b71e1ee4c5f7f5298c702c0de4c590000"
    "name": "parallel_test"
  -
    "device": "Google Nexus 6"
    "app": "bs://6c31566b71e1ee4c5f7f5298c702c0de4c590000"
    "name": "parallel_test"
```

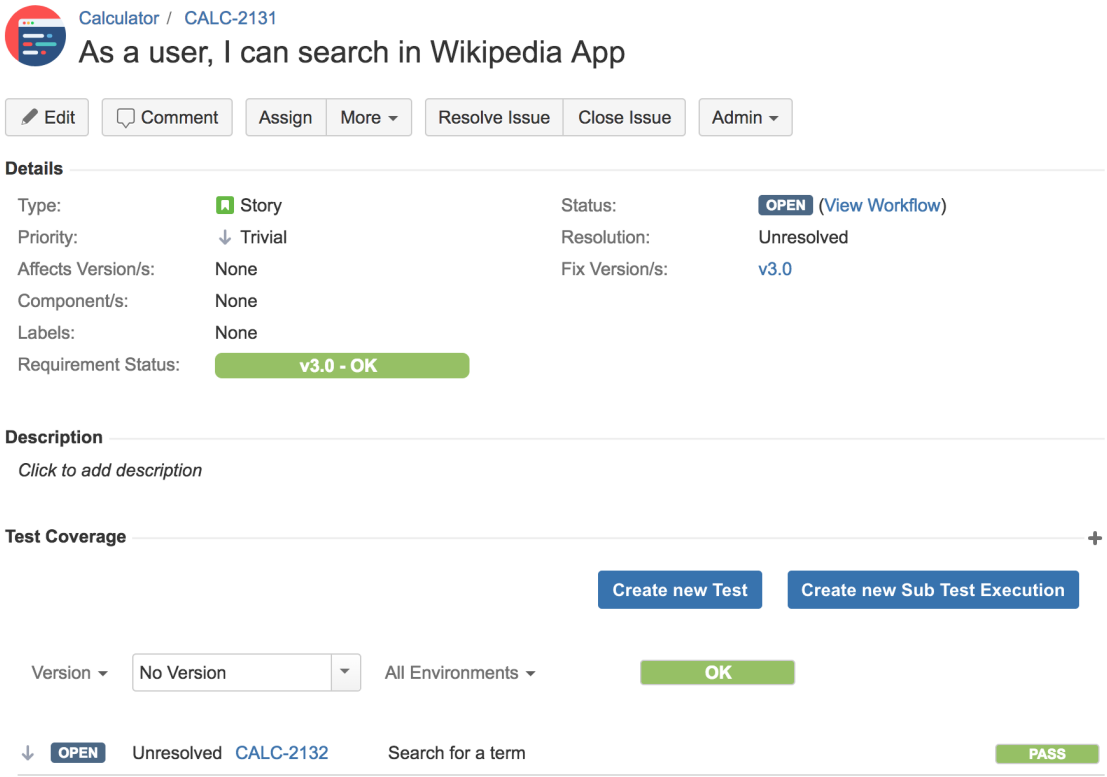
In this tutorial we're using a wikipedia sample application from BrowserStack, that must be uploaded beforehand to BrowserStack. The hashed app id must be configured accordingly on the previous configuration file.

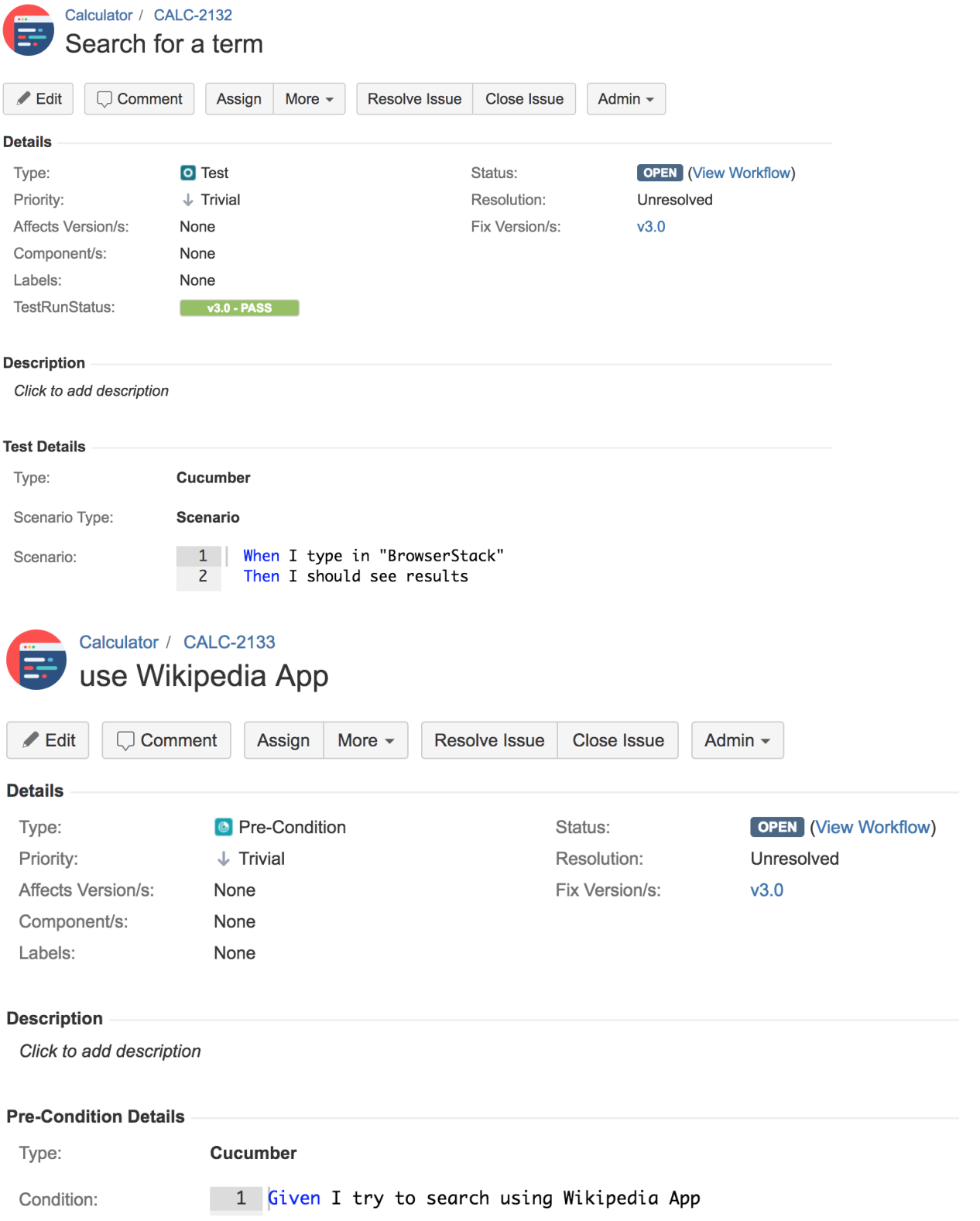
Instead of using the provided "single.feature" file, we'll use JIRA and Xray as master of information.

In other words, in JIRA we'll:

1. create a story
2. create a Test for it
3. create a Pre-Condition and associate it to the previous Test

Although it's not needed, we will also create a blank Test Execution with the Test and we'll use it as basis in order to run and report our test results.





After creating a Cucumber Test, of Cucumber Type "Scenario", in Jira, you can export the specification of the test to a Cucumber .feature file via the REST API or the **Export to Cucumber** UI action from within the Test Execution issue.

The created file will be similar to the following:

features/single.feature

```
@CALC-2130
@REQ_CALC-2131
Feature: As a user, I can search in Wikipedia App

        Background:
                #@CALC-2133
                Given I try to search using Wikipedia App


        @TEST_CALC-2132
        Scenario: Search for a term
                  When I type in "BrowserStack"
                  Then I should see results
```

Tests can be run by execution rake's "parallel" task.

```
 bundle exec rake parallel
```

The previous task will generate a Cucumber JSON report per each target device.

```
zip browserstack.zip device_*.json
```

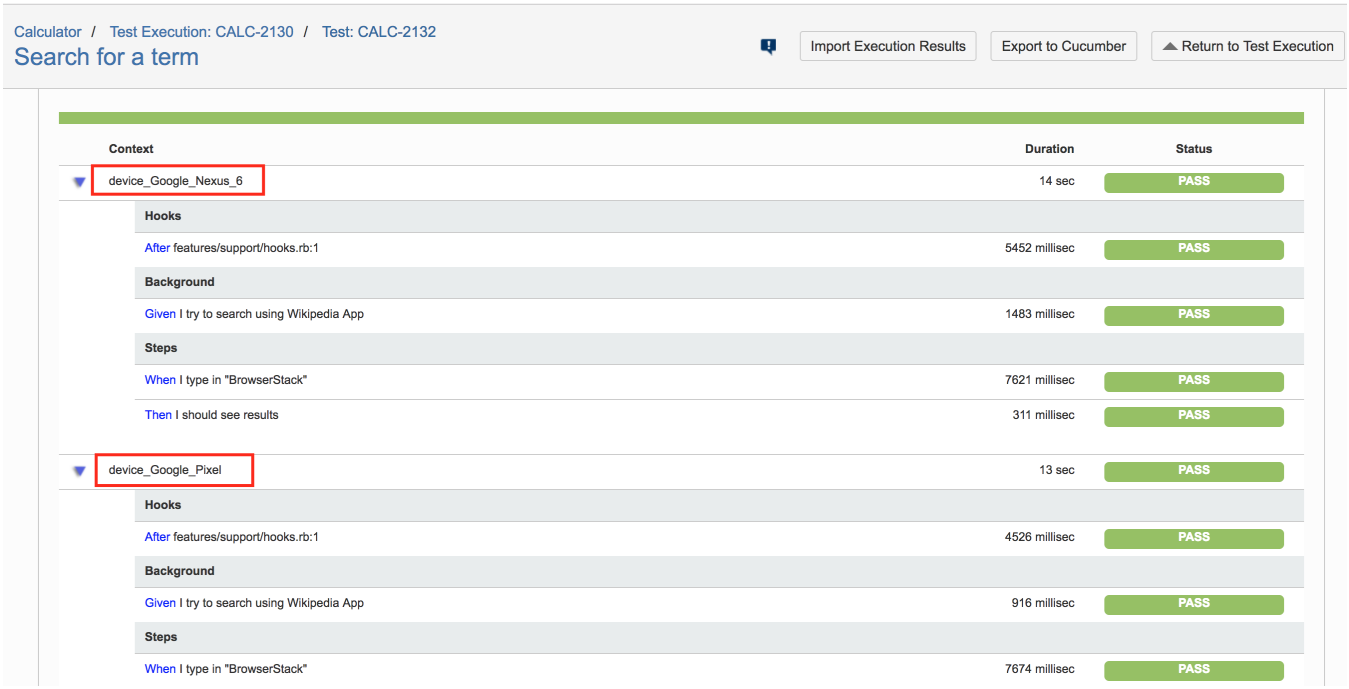
These files can be bundled in ZIP file and submitted to Xray using the "bundle" REST API endpoint (either by invoking the REST API directly or by using one of the free add-ons for Jenkins/Bamboo).

Example for submission of results using "curl"

```
curl -H "Content-Type: multipart/form-data" -u user:password -F "file=@browserstack.zip" https://sandbox.xpand-addons.com/rest/raven/1.0/import/execution/bundle
```

The execution screen details will not only provide information on the test run result, but also for each step.

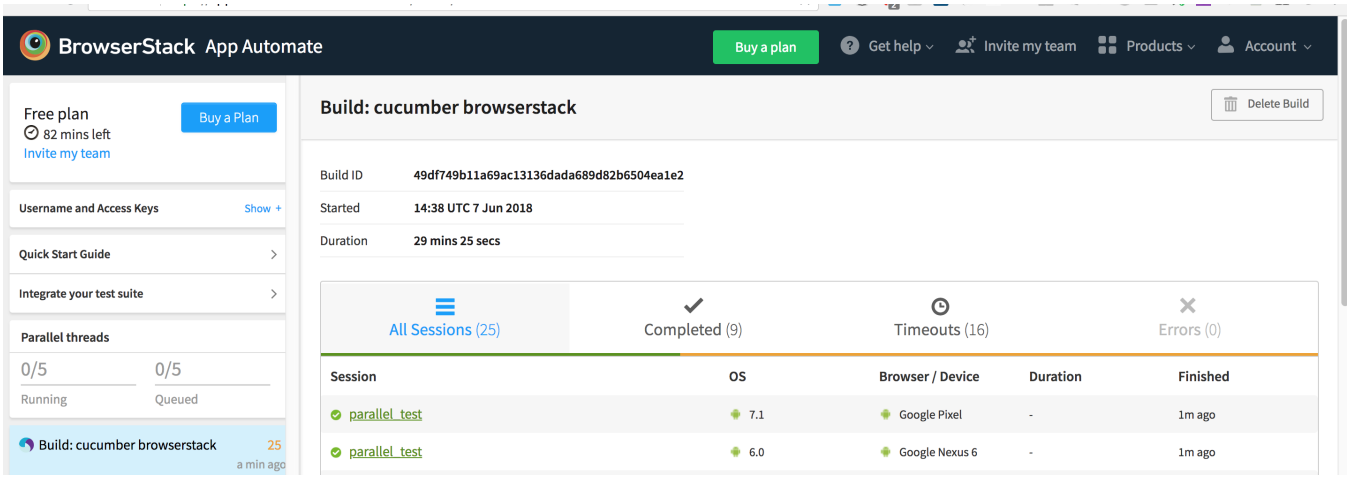
For each device, a different "context" will appear along with the respective step results.



Learn more

Please see Testing in BDD with Gherkin based frameworks (e.g. Cucumber) for an overview on how to use Cucumber Tests with Xray.

In BrowserStack you can see some info about it.



References

- https://docs.cucumber.io/tools/ruby/
- https://www.browserstack.com/app-automate/appium-cucumber
- Testing in BDD with Gherkin based frameworks (e.g. Cucumber)
- Exporting Cucumber Tests - REST
- Import Execution Results - REST