Testing web applications using Mocha and WebdriverIO

What you'll learn

- Define tests using Mocha
- Run the test and push the test report to Xray
- Validate that the test results are available in Jira

Source code for this tutorial

- code is available in GiHub

Overview

WebdriverIO is the next-gen browser and mobile automation test framework for Node.js.

Prerequisites

For this example we will use the WebdriverIO framework with the default assertion library provided by WebdriverIO and Junit as the reporter.

In the official documentation we can find the following description:

> WebdriverIO is a progressive automation framework built to automate modern web and mobile applications. It simplifies the interaction with your app and provides a set of plugins that help you create a scalable, robust and flakiness test suite.

If we want, we can use other runners or reporters.

We will need:

- Access to heroku demo site that we aim to test
- Node.js environment with WebDriverIO

To start using WebDriverIO please follow the Get Started documentation.

WebDriverIO provides a client that after being installed will guide you through bootstraping a *Hello World* test suite into your project. For this tutorial we will use the code generated by this tool for simplicity (with page objects).The test consists in validating the login feature (with valid and invalid credentials) of the demo site, for that we have created a base page object that will contain all methods and functionality that is shared across all page objects, a login page, that will extend the base page, that will have all the methods for interacting with the login page and a result page that will have the methods to interact in the page that is loaded after the login operation.

./pageobjects/Page.js

```
/**
* main page object containing all methods, selectors and functionality
* that is shared across all page objects
*/
module.exports = class Page {
    /**
    * Opens a sub page of the page
    * @param path path of the sub page (e.g. /path/to/page.html)
    */
    open (path) {
        return browser.url(`https://the-internet.herokuapp.com/${path}`)
    }
}
```

./pageobjects/login.page.js

```
const Page = require('./page');

/**
 * sub page containing specific selectors and methods for a specific page
 */
class LoginPage extends Page {
    /**
     * define selectors using getter methods
     */
    get inputUsername () { return $('#username') }
    get inputPassword () { return $('#password') }
    get btnSubmit () { return $('button[type="submit"]') }

    /**
     * a method to encapsule automation code to interact with the page
     * e.g. to login using username and password
     */
    async login (username, password) {
        await (await this.inputUsername).setValue(username);
        await (await this.inputPassword).setValue(password);
        await (await this.btnSubmit).click();
    }

    /**
     * overwrite specifc options to adapt it to page object
     */
    open () {
        return super.open('login');
    }
}

module.exports = new LoginPage();
```

./pageobjects/secure.page.js

```
const Page = require('./page');

/**
 * sub page containing specific selectors and methods for a specific page
 */
class SecurePage extends Page {
    /**
     * define selectors using getter methods
     */
    get flashAlert () { return $('#flash') }
}

module.exports = new SecurePage();
```

Define the test that will assert if the operation is successful or not

./example.e2e.js

```
const LoginPage = require('../pageobjects/login.page');
const SecurePage = require('../pageobjects/secure.page');

describe('My Login application', () => {
    it('should login with valid credentials', async () => {
        await LoginPage.open();

        await LoginPage.login('tomsmith', 'SuperSecretPassword!');
        await expect(SecurePage.flashAlert).toBeExisting();
        await expect(SecurePage.flashAlert).toHaveTextContaining(
            'You logged into a secure area!');
    });
});

describe('My Login application', () => {
    it('should not login with invalid credentials', async () => {
        await LoginPage.open();

        await LoginPage.login('tom', 'SuperPassword!');
        await expect(SecurePage.flashAlert).toBeExisting();
        await expect(SecurePage.flashAlert).toHaveTextContaining(
            'Your username is invalid!');
    });
});
```

All of the above were created by the tool provided by WebDriverIO, to create those we followed the documentation and executed first the command to install the WebDriverIO test runner:

```
npm install @wdio/cli
```

Then we answer a series of questions that will define the code to be generated using:

```
npx wdio config
```

The output of the questionnaire will look like this:

```
=======================
WDIO Configuration Helper
=======================

? Where is your automation backend located? On my local machine
? Which framework do you want to use? mocha
? Do you want to use a compiler? No!
? Where are your test specs located? ./test/specs/**/*.js
? Do you want WebdriverIO to autogenerate some test files? Yes
? Do you want to use page objects (https://martinfowler.com/bliki/PageObject.html)? Yes
? Where are your page objects located? ./test/pageobjects/**/*.js
? Which reporter do you want to use? spec
? Do you want to add a service to your test setup? chromedriver
? What is the base url? http://localhost
```

The last two steps to have everything configured is to define that we will use the Junit framework, for that we execute the following command:

```
npm install @wdio/junit-reporter --save-dev
```

In wdio.conf.js , we have added, in the reporters area, the following Junit definition:

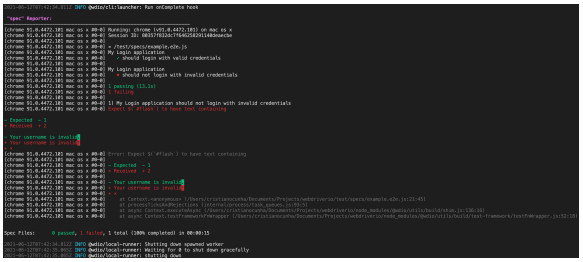
./wdio.conf.js

```
...
   reporters: ['spec',
            ['junit', {
                outputDir: './',
                outputFileFormat: function(options) { // optional
                    return `results.xml`
                }
            }]
        ],
...
```

Once the code is implemented (and we will make it fail on purpose on one test, to show the failure reports), it can be executed with the following command:

```
npx wdio run ./wdio.conf.js
```

The results are immediately available in the terminal.



In this example, one test has failed and the other one has succeed, the output generated in the terminal is the above one and the correspondent Junit report is as below:

Junit Report

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
  <testsuite name="My Login application" timestamp="2021-06-12T08:42:21"
time="2.37" tests="1" failures="0" errors="0" skipped="0">
    <properties>
```

```
      <property name="specId" value="0"/>
      <property name="suiteName" value="My Login application"/>
      <property name="capabilities" value="chrome.91_0_4472_101.macosx"/>
      <property name="file" value="./test/specs/example.e2e.js"/>
    </properties>
    <testcase classname="chrome.91_0_4472_101.macosx.My Login application"
name="should login with valid credentials" time="2.369">
      <system-out><![CDATA[
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/url - {"url":"
https://the-internet.herokuapp.com/login"}
RESULT: {"url":"https://the-internet.herokuapp.com/login"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#username"}
RESULT: {"using":"css selector","value":"#username"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/bdbe5639-
b66b-4f3e-9551-d701b1040909/clear - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/bdbe5639-
b66b-4f3e-9551-d701b1040909/value - {"text":"tomsmith"}
RESULT: {"text":"tomsmith"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#password"}
RESULT: {"using":"css selector","value":"#password"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/f9b56cb7-
f274-4370-889d-044db3b07ecb/clear - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/f9b56cb7-
f274-4370-889d-044db3b07ecb/value - {"text":"SuperSecretPassword!"}
RESULT: {"text":"SuperSecretPassword!"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"button[type=\"submit\"]"}
RESULT: {"using":"css selector","value":"button[type=\"submit\"]"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/3f43e518-
e698-4bff-b647-6255d6fbeab5/click - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/elements -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/7e87ed7b-
be88-4023-bb59-b3439d06d53f/text - {}
RESULT: {}
]]></system-out>
    </testcase>
  </testsuite>
  <testsuite name="My Login application" timestamp="2021-06-12T08:42:23"
time="10.709" tests="1" failures="1" errors="1" skipped="0">
    <properties>
      <property name="specId" value="0"/>
      <property name="suiteName" value="My Login application"/>
      <property name="capabilities" value="chrome.91_0_4472_101.macosx"/>
      <property name="file" value="./test/specs/example.e2e.js"/>
    </properties>
    <testcase classname="chrome.91_0_4472_101.macosx.My Login application"
name="should not login with invalid credentials" time="10.707">
      <failure/>
      <error message="Expect $(`#flash`) to have text containing

[32m- Expected  - 1[39m
[31m+ Received  + 2[39m

[32m- Your username is invalid[7m.[27m[39m
[31m+ Your username is invalid[7m![27m[39m
[31m+ ×[39m"/>
      <system-out><![CDATA[
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/url - {"url":"
```

```
https://the-internet.herokuapp.com/login"}
RESULT: {"url":"https://the-internet.herokuapp.com/login"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#username"}
RESULT: {"using":"css selector","value":"#username"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/af5062d3-
a7f3-41f2-9fb1-9d7db3641681/clear - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/af5062d3-
a7f3-41f2-9fb1-9d7db3641681/value - {"text":"tom"}
RESULT: {"text":"tom"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#password"}
RESULT: {"using":"css selector","value":"#password"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/b4df7026-
0164-4c38-93ee-419a9b963d67/clear - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/b4df7026-
0164-4c38-93ee-419a9b963d67/value - {"text":"SuperPassword!"}
RESULT: {"text":"SuperPassword!"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"button[type=\"submit\"]"}
RESULT: {"using":"css selector","value":"button[type=\"submit\"]"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element/457ae344-
2625-495d-a17c-fa1189ceb0c8/click - {}
RESULT: {}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/elements -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: POST /session/80357f832dc7f646258291140deaecbe/element -
{"using":"css selector","value":"#flash"}
RESULT: {"using":"css selector","value":"#flash"}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
```

```
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
RESULT: {}
COMMAND: GET /session/80357f832dc7f646258291140deaecbe/element/f73dfed1-
9b8b-4dd0-86c2-ca9e2dea499f/text - {}
COMMAND: DELETE /session/80357f832dc7f646258291140deaecbe - {}
RESULT: {}
RESULT: {}
]]></system-out>
      <system-err><![CDATA[
Error: Expect $(`#flash`) to have text containing

[32m- Expected  - 1[39m
[31m+ Received  + 2[39m

[32m- Your username is invalid[7m.[27m[39m
[31m+ Your username is invalid[7m![27m[39m
[31m+ ×[39m
    at Context.<anonymous> (/Users/cristianocunha/Documents/Projects
/webdriverio/test/specs/example.e2e.js:21:45)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
    at async Context.executeAsync (/Users/cristianocunha/Documents/Projects
/webdriverio/node_modules/@wdio/utils/build/shim.js:136:16)
    at async Context.testFrameworkFnWrapper (/Users/cristianocunha
/Documents/Projects/webdriverio/node_modules/@wdio/utils/build/test-
framework/testFnWrapper.js:52:18)
]]></system-err>
    </testcase>
  </testsuite>
</testsuites>
```

Notes:

- There are a lot of other options on how to use WebDriverIO, please check their documentation for more information.

Integrating with Xray

As we saw in the example above where we produced Junit reports with the result of the tests, it is now a matter of importing those results to your Jira instance. This can be done by simply submitting automation results to Xray through the REST API, by using one of the available CI/CD plugins (e.g. for Jenkins) or using the Jira interface to do so.

API

API

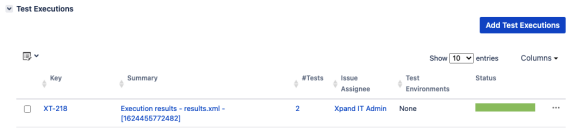
Once you have the report file available you can upload it to Xray through a request to the REST API endpoint for JUnit. The first step is to follow the instructions in v1 or v2 (depending on your usage) to obtain the token we will be using in the subsequent requests.

JUnit XML results

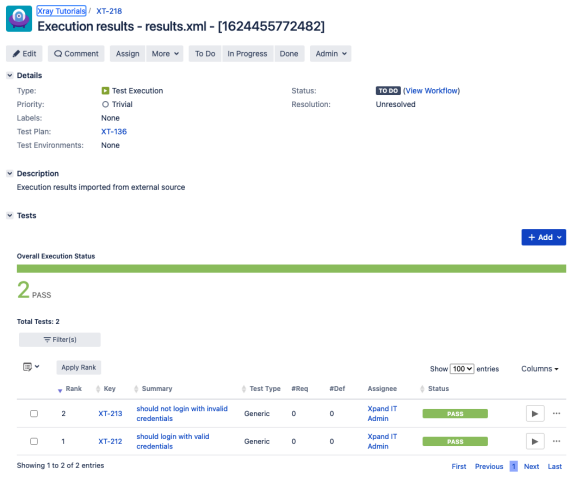
We will use the API to perform the below request with the definition of some common fields on the Test Execution, such as the target project, Test Plan, etc.

```
curl -H "Content-Type: multipart/form-data" -u
$JIRA_USERNAME:$JIRA_PASSWORD -F "file=@results.xml" $JIRA_BASEURL/rest
/raven/1.0/import/execution/junit?projectKey=XT&testPlanKey=XT-136
```

With this command we are creating a new Test Execution in the referred Test Plan with a generic summary and two tests with a summary based on the test name.



In Xray we can see that the tests are associated to a Test Plan and we can identify what tests are failing or passing, below you can see two tests (for valid and invalid credentials):



Jenkins

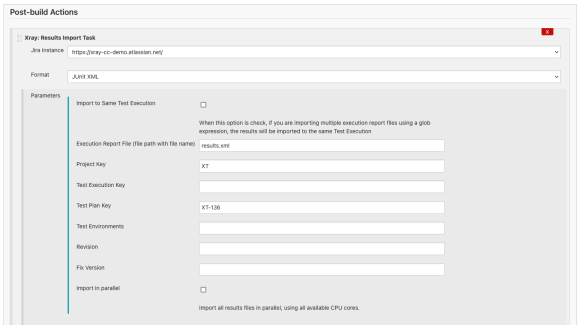
Jenkins

As you can see below we are adding a post-build action using the "*Xray: Results Import Task*" (from the Xray plugin available), where we have some options, we will focus on the one called "*Junit XML*."

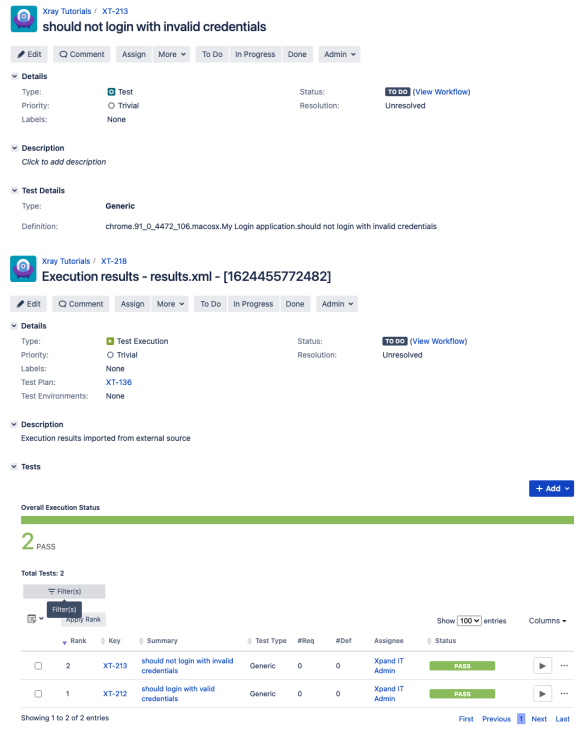
Junit XML

- the Jira instance (where you have your Xray instance installed)

- the format as "*JUnit XML*"
- the test results file we want to import
- the Project key corresponding of the project, in Jira, where the results will be imported

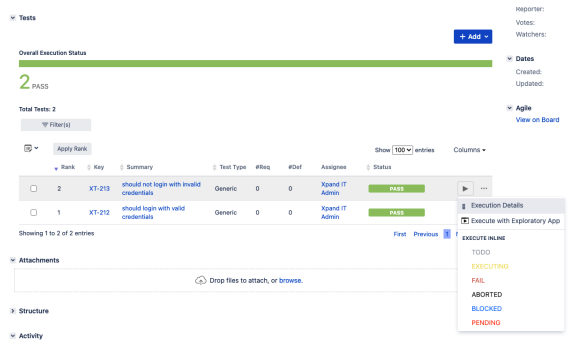


Tests implemented will have a corresponding Test entity in Xray. Once results are uploaded, Test issues are auto-provisioned, unless they already exist.

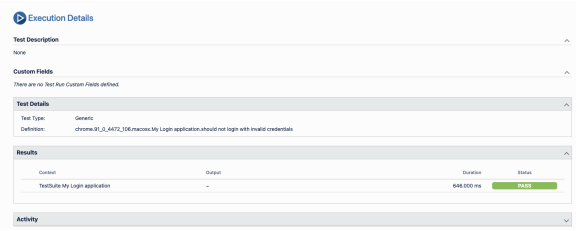


Xray uses a concatenation of the suite name and the test name as the the unique identifier for the test (In this case it was prefixed with the browser name and version).

In Xray, results are stored in a Test Execution, usually a new one. The Test Execution contains a Test Run per each test that was executed.



Detailed results, including logs and exceptions reported during execution of the test, can be seen on the execution screen details of each Test Run, accessible through the *Execution details* as we can see here:

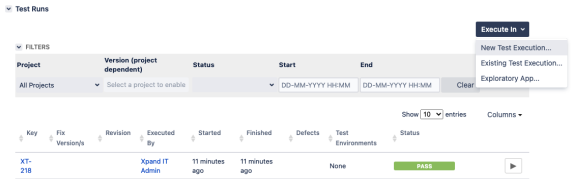


Jira UI

Jira UI

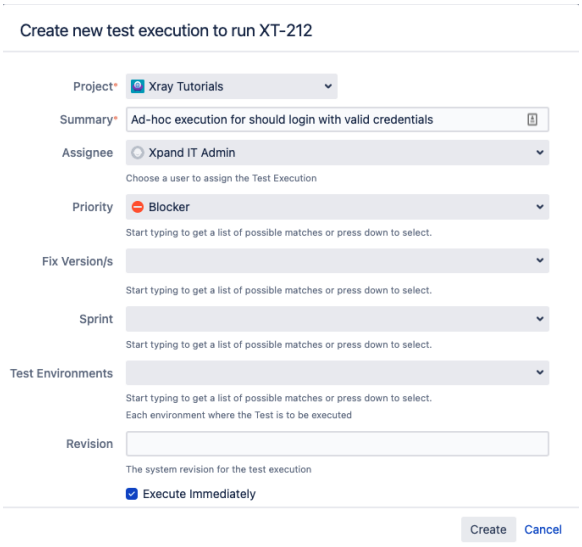
1

Create a Test Execution for the test that you have



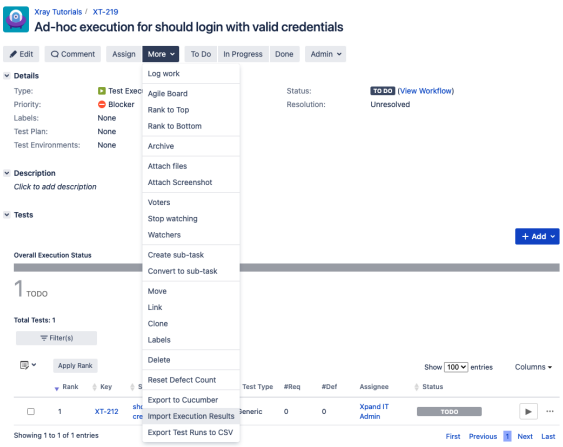
2

Fill in the necessary fields and press "*Create*."



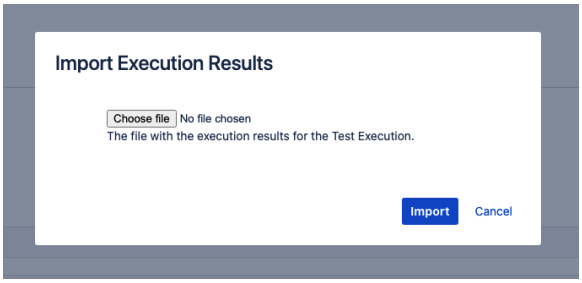
3

Open the Test Execution and import the JUnit report.



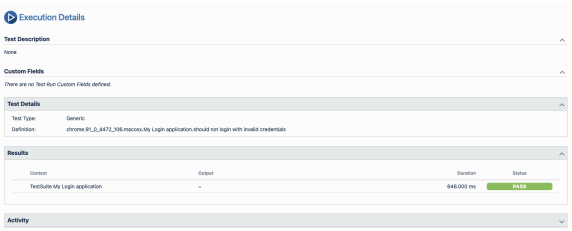
4

Choose the results file and press *"Import."*



5

The Test Execution is now updated with the test results imported.



Tips

- after results are imported in Jira, Tests can be linked to existing requirements/user stories, so you can track the impact of their coverage.
- results from multiple builds can be linked to an existing Test Plan, to facilitate the analysis of test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results by that environment later on. A Test Environment can be a testing stage (e.g. dev, staging, preprod, prod) or an identifier of the device/application used to interact with the system (e.g. browser, mobile OS).

References

- https://webdriver.io/
- https://webdriver.io/docs/gettingstarted