

Testing web applications using Playwright



What you'll learn

- [Prerequisites](#)
- [How tests are defined](#)
- [Implementing tests](#)
- [Integrating with Xray](#)
 - [Validate that the test results are available in Jira](#)
 - [JUnit XML results](#)
 - [JUnit XML results Multipart](#)
 - [Jenkins](#)
 - [JUnit XML](#)
 - [JUnit XML multipart](#)
- [Source-code for this tutorial](#)
 - [Jira UI](#)
- [Tips](#)
- [References](#)

Overview

Playwright is a recent browser automation tool that provides an alternative to Selenium.

Prerequisites

For this example we will use [Playwright Test Runner](#), that accommodate the needs of the end-to-end testing. It does everything you would expect from the regular test runner.

Playwright Test Runner is still fairly new as you can see in the official documentation:

Zero config cross-browser end-to-end testing for web apps. Browser automation with [Playwright](#), Jest-like assertions and built-in support for TypeScript.

Playwright test runner is available in preview and minor breaking changes could happen. We welcome your feedback to shape this towards 1.0.

If you want, you can use other runners (e.g. Jest, AVA, mocha).

What you need:

- Access to a [demo site](#) that you want to test
- Node.js environment with Playwright and [Playwright Test Runner](#)

Implementing tests

To start using the [Playwright Test Runner](#), follow the [Get Started](#) documentation.

The test consists of validating the login feature (with valid and invalid credentials) of the [demo site](#), for which we have created a page object that will represent the loginPage

./models/Login.js

```
const config = require ("../config.json");

// models/Login.js
class LoginPage {

  constructor(page) {
    this.page = page;
  }

  async navigate() {
    await this.page.goto(config.endpoint);
  }

  async login(username, password) {
    await this.page.fill(config.username_field, username);
    await this.page.fill(config.password_field, password);
    await this.page.click(config.login_button);
  }

  async getInnerText(){
    return this.page.innerText("p");
  }

}

module.exports = { LoginPage };
```

plus a configuration file where we have the identifiers that will match the elements in the page

config.json

```
{
  "endpoint" : "https://robotwebdemo.onrender.com/",
  "login_button" : "id=login_button",
  "password_field" : "input[id=\"password_field\"]",
  "username_field" : "input[id=\"username_field\"]"
}
```

And define the test that will assert if the operation is successful or not

login.spec.ts

```
import { it, describe, expect } from "@playwright/test"
import { LoginPage } from "../models/Login";

describe("Login validations", () => {

  it('Login with valid credentials', async({page}) => {
    const loginPage = new LoginPage(page);
    await loginPage.navigate();
    await loginPage.login("demo","mode");
    const name = await loginPage.getInnerText();
    expect(name).toBe('Login succeeded. Now you can logout.');
```

```
  });

  it('Login with invalid credentials', async({page}) => {
    const loginPage = new LoginPage(page);
    await loginPage.navigate();
    await loginPage.login("demo","model");
    const name = await loginPage.getInnerText();
    expect(name).toBe('Login failed. Invalid user name and/or
password.');
```

```
  });
})
```

The [Playwright Test Runner](#) provides a Jest like way of describing test scenarios, here you can see that it uses *'it, describe, expect'*.

These are simple tests that will validate the login functionality by accessing the [demo site](#), inserting the username and password (in one test with valid credentials and in another with invalid credentials), clicking the login button and validating if the page returned is the one that matches your expectation.

For the below example we will do a small change to force a failure, so in the *login.spec.ts* file remove *" /or"* from the expectation on the Test *'Login with invalid credentials'*, this is the end result:

login.spec.ts

```
import { test, expect } from "@playwright/test"
import { LoginPage } from "../models/Login";

test.describe("Login validations", () => {

  test('Login with valid credentials', async({ page }) => {
    const loginPage = new LoginPage(page);
    await loginPage.navigate();
    await loginPage.login("demo","mode");
    const name = await loginPage.getInnerText();
    expect(name).toBe('Login succeeded. Now you can logout.');
```

```
  });

  test('Login with invalid credentials', async({ page }) => {
    const loginPage = new LoginPage(page);
    await loginPage.navigate();
    await loginPage.login("demo","model");
    const name = await loginPage.getInnerText();
    expect(name).toBe('Login failed. Invalid user name and password.');
```

```
  });
})
```

Once the code is implemented (and we will make it fail on purpose on the *'Login with invalid credentials'* test due to missing word, to show the failure reports), can be executed with the following command:

```
npx folio -p browserName=chromium --reporter=junit,line --test-match=login.spec.ts
```

First, define one extra parameter: "*browserName*" in order to execute the tests only with the chrome browser (chromium), otherwise the default behaviour is to execute the tests for the three available browsers (chromium, firefox and webkit).

The results are immediately available in the terminal

```
Running 2 tests using 1 worker
  login tests using login with invalid credentials .....
    browserName: null, headless: false, logPath: /tmp/runner/0, screenshot: false, url: null
    Error: expect(received).toBe(expected) // Object is equality
    Expected: login failed. invalid user name and password.
    Received: login failed. invalid user name and password.
    17 |     const loginPage = loginPageFactory();
    18 |     await loginPage.getPageInnerText();
    19 |     expect(nade).toEqual(loginPage.invalid user name and password.);
    20 |
    21 |   });
    22 | });
  at /Users/cristianmouha/Documents/Projects/Faker/gitter/login.spec.ts:19:22
  at WorkerTask.run (/Users/cristianmouha/Documents/Projects/Faker/gitter/node_modules/fake-runner/lib/workerTask.js:15:15)
  at processImmediate (internal/timers.js:46:21)
  at WorkerRunner.run (/Users/cristianmouha/Documents/Projects/Faker/gitter/node_modules/fake-runner/lib/workerRunner.js:119:17)
  at /Users/cristianmouha/Documents/Projects/Faker/gitter/node_modules/fake-runner/lib/runner.js:10:15
    Error: expect(received).toBe(expected) // Object is equality
    Expected: login failed. invalid user name and password.
    Received: login failed. invalid user name and password.
    browserName: null, headless: false, logPath: /tmp/runner/0, screenshot: false, url: null
```

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 corresponding Junit report is below:

JUnit Report

```
<testsuites id="" name="" tests="2" failures="1" skipped="0" errors="0"
time="2.592">
<testsuite name="login.spec.ts" timestamp="1617094735952" hostname=""
tests="2" failures="1" skipped="0" time="2.37" errors="0">
<testcase name="Login validations Login with valid credentials" classname="
login.spec.ts Login validations" time="1.358">
</testcase>
<testcase name="Login validations Login with invalid credentials"
classname="login.spec.ts Login validations" time="1.012">
<failure message="login.spec.ts:14:5 Login with invalid credentials" type="
FAILURE">
  login.spec.ts:14:5 > Login validations Login with invalid credentials
  =====
    browserName=webkit, headful=false, slowMo=0, video=false,
    screenshotOnFailure=false

    Error: expect(received).toBe(expected) // Object.is equality

    Expected: &quot;Login failed. Invalid user name and password.&quot;;
    Received: &quot;Login failed. Invalid user name and/or password.&quot;;

      17 |         await loginPage.login(&quot;demo&quot;;,&quot;
model&quot;);
      18 |         const name = await loginPage.getInnerText();
    > 19 |         expect(name).toBe('Login failed. Invalid user name and
password.');
```

^

```
      20 |     });
      21 | })

    at /Users/cristianocunha/Documents/Projects/Playwrighttest/login.
spec.ts:19:22
    at runNextTicks (internal/process/task_queues.js:58:5)
    at processImmediate (internal/timers.js:434:9)
    at WorkerRunner._runTestWithFixturesAndHooks (/Users/cristianocunha
/Documents/Projects/Playwrighttest/node_modules/folio/out/workerRunner.js:
198:17)

</failure>
</testcase>
</testsuite>
</testsuites>
```

Repeat this process for each browser type in order to have the reports generated for each browser.

Notes:

- By default it will execute tests for the 3 browser types available (that is why we are forcing it to execute for only one browser)
- By default all the tests will be executed in headless mode
- Folio command line will search and execute all tests in the format: `"/**/?(.*)(spec|test).[it]s"`
- In order to get the JUnit test report please follow this [section](#).

Integrating with Xray

As we saw in the above example, where we are producing JUnit reports with the result of the tests, it is now a matter of importing those results to your Jira instance. You can do this 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](#). To do that, follow the first step in 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 request with the definition of some common fields on the Test Execution, such as the target project, project version, etc.

In the first version of the API, the authentication used a login and password (not the token that is used in Cloud).

```
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@junit.xml" 'http://<LOCAL_JIRA_INSTANCE>/rest/raven/1.0/import/execution/junit?projectKey=COM&testPlanKey=COM-9'
```

With this command, you will create a new Test Execution in the referred Test Plan with a generic summary and two tests with a summary based on the test name.

The screenshot shows the Jira Xray interface for a test execution. At the top, it says 'Execution results - junit.xml - [1622479652449]'. Below this are tabs for 'Edit', 'Comment', 'Assign', 'More', 'To Do', 'In Progress', 'Done', and 'Admin'. The 'Details' tab is active, showing fields like 'Type: Test Execution', 'Priority: Trivial', 'Component(s): None', 'Labels: None', 'Test Plan: None', and 'Test Environments: None'. The 'Status' is 'Unresolved'. Below the details is a 'Description' section with a 'Tests' tab. The 'Tests' tab shows a table with 2 tests. The first test is 'Login validations Login with valid credentials' and the second is 'Login validations Login with invalid credentials'. Both tests are marked as 'PASS'.

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	BOOK-290	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS
2	BOOK-289	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS

JUnit XML results Multipart

However, there's another endpoint that is more flexible and allows the customization of any field on the target Test Execution; this is the specific [JUnit multipart endpoint](#).

This endpoint follows a JSON-based syntax based on Jira's REST API for updating issues. As an example of uploading the results to a Test Execution with a given Summary, we have created these two additional files: *issueFields.json* and *testIssueFields.json*, where we are doing the above associations.

issueFields.json

```
{
  "fields": {
    "project": {
      "id": "12400"
    },
    "summary": "Login validation [Webkit]",
    "issuetype": {
      "id": "10100"
    },
    "components" : [
      {
        "name": "Interface"
      },
      {
        "name": "Login"
      }
    ]
  }
}
```

testIssueFields.json

```
{
  "fields": {
    "project": {
      "id": "12400"
    }
  }
}
```

To upload the reports through Junit multipart endpoint, use the following command:

```
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@junit.xml" -F "info=@xray_multipart/issueFields.json" -F "testInfo=@xray_multipart/testIssueFields.json" 'http://192.168.56.111:8080/rest/raven/1.0/import/execution/junit/multipart'
```

On Xray, you can see the tests and you can identify which tests are failing or passing. Below you can see two tests (for valid and invalid credentials):

ComicStore / COM-28
Login validation [Webkit]

Edit Comment Assign More To Do In Progress Done Admin

Details

Type: Test Execution
Priority: Trivial
Component/s: Interface, Login
Labels: None
Test Plan: None
Test Environments: None

Status: **TO DO** (View Workflow)
Resolution: Unresolved

Description

Tests

+ Add

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Apply Rank

Show 100 entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
2	COM-24	Login validations Login with invalid credentials	Generic	0	0		PASS
1	COM-23	Login validations Login with valid credentials	Generic	0	0		PASS

You can also notice that the summary is now defined based on the files we used for uploading the test results.

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. For now, we will focus on two of those, one called "*JUnit XML*" (simpler) and another called "*Junit XML multipart*" (both are explained below and will require two extra files).

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

Xray: Results Import Task

Jira Instance: Local Server

Format: JUnit XML

Parameters

Import to Same Test Execution ☐

When this option is checked, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution

Execution Report File (file path with file name):

Project Key:

Test Execution Key:

Test Plan Key:

Test Environments:

Revision:

Fix Version:

Import in parallel ☐

Import all results files in parallel, using all available CPU cores.

[Click here for more details](#)

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

ComicStore / COM-24

Login validations Login with invalid credentials

Edit
Comment
Assign
More
To Do
In Progress
Done
Admin

Details

Type: **Test**
Priority: **Trivial**
Component(s): **None**
Labels: **Automation** **JUnit** **Testing**

Status: **TO DO** (View Workflow)
Resolution: **Unresolved**

Description

Click to add description

Test Details

Type: **Generic**
Definition: login.spec.ts Login validations.Login validations Login with invalid credentials

Pre-Conditions

This test is not associated with Pre-Conditions yet.

Create Pre-Condition
Associate Pre-Conditions

Xray uses a concatenation of the suite name and the test name as the the unique identifier for the test.

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 using playwright-test runner.

ComicStore / COM-29

Execution results - junit.xml - [1622537543505]

Edit
Comment
Assign
More
To Do
In Progress
Done
Admin

Details

Type: **Test Execution**
Priority: **Trivial**
Component(s): **None**
Labels: **None**
Test Plan: **COM-9**
Test Environments: **None**

Status: **TO DO** (View Workflow)
Resolution: **Unresolved**

Description

Execution results imported from external source

Tests

+ Add

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Rank
Key
Summary
Test Type
#Req
#Def
Assignee
Status

2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS
1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS

Showing 1 to 2 of 2 entries

First Previous Next Last

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

Tests

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Rank
Key
Summary
Test Type
#Req
#Def
Assignee
Status

2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS
1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS

Showing 1 to 2 of 2 entries

First Previous Next Last

Attachments

Drop files to attach, or browse.

Structure

Activity

All
Comments
Work Log
History
Activity

There are no comments yet on this issue.

Execution Details

Test Description

None

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: **Generic**
Definition: login.spec.ts Login validations.Login validations Login with invalid credentials

Results

Context	Output	Duration	Status
Testbed login.spec.ts	-	990,000 ms	PASS

EXECUTE NONE

TODO

EXECUTING

FAIL

ABORTED

BLOCKED

PENDING

FAIL_DISCARDABLE

As you can see here:

Execution Details

Test Description

None

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: **Generic**
Definition: login.spec.ts Login validations.Login validations Login with invalid credentials

Results

Context	Output	Duration	Status
Testbed login.spec.ts	-	990,000 ms	PASS

JUnit XML multipart

- the Jira instance (where you have your Xray instance installed)
- the format as "JUnit XML Multipart"

- the two files already added to the repo: "[issueFields.json](#)" and "[testIssueFields.json](#)" (in the **xray_multipart** directory, note that you must update the inner values to have the correct labels, projectId, issueType and environments)
- The results file, in our case "*junit.xml*"

In this integration we have more control over the import to Jira. In this particular case, you can see that we will import these results to the Project with the id defined in the file, with a specific summary, all of this is specified in the files (issueFields.json and testIssuesFields.json).

Tests

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Apply Rank

Show 100 entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS
1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS

Showing 1 to 2 of 2 entries

Jira UI

Jira UI

1

Create a Test Execution for the test that you have

Tests

Test Plan Board

Create Test Execution

Overall Execution Status

6 PASS

Total Tests: 6

Filter(s)

1 selected

Remove

+ Create Test Execution

Create

Summary	# Executions	Issue Assignee	Component/s	Begin Date	End Date	Test Plan	Fix Version/s	Latest Status
CanDoStuff	1	Administrator				None		PASS
CanAddNumbers	1	Administrator				None		PASS
CanSubtract	1	Administrator				None		PASS
CanMultiply	1	Administrator				None		PASS
Login validations Login with valid credentials	1	Administrator				None		PASS
Login validations Login with invalid credentials	1	Administrator				None		PASS

Showing 1 to 6 of 6 entries

2

Fill in the necessary fields and press "Create."

Create new test execution for tests in test plan COM-9

Project ComicStore

Summary

Assignee Administrator
Choose a user to assign the Test Execution

Priority Blocker
Start typing to get a list of possible matches or press down to select.

Fix Version/s
Start typing to get a list of possible matches or press down to select.

Sprint
Start typing to get a list of possible matches or press down to select.

Test Environments
Start typing to get a list of possible matches or press down to select.
Each environment where the Test is to be executed

Revision
The system revision for the test execution

☒ Redirect to Test Execution

Create Cancel

3

Open the Test Execution and import the JUnit report.

ComicStore / COM-30

Test Execution for Test Plan COM-9

Edit Comment Assign More To Do In Progress Done Admin

Log work

Type: Test Exec
Priority: Blocker
Component/s: None
Labels: None
Test Plan: COM-9
Test Environments: None

Status: TODO (View Workflow)
Resolution: Unresolved

Details

Description
Click to add description

Tests

Overall Execution Status

1 TODO

Total Tests: 1

Apply Rank

Rank Key

Export to Cucumber
Import Execution Results
Export Test Runs to CSV

Test Type	#Req	#Def	Assignee	Status
Generic	0	0	Administrator	TODO

Show: 100 entries Columns

4

Choose the results file and press "Import."

Import Execution Results

Browse... No file selected.

The file with the execution results for the Test Execution.

Import Cancel

5

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

ComicStore / COM-30

Test Execution for Test Plan COM-9

Edit
Comment
Assign
More
To Do
In Progress
Done
Admin

Details

Type: Test Execution

Priority: Blocker

Component(s): None

Labels: None

Test Plan: COM-9

Test Environments: None

Status: TO DO (View Workflow)

Resolution: Unresolved

Description

Click to add description

Tests

+ Add

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Apply Rank

Show 100 entries

Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS	▶ ...
1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS	▶ ...

Showing 1 to 2 of 2 entries

First Previous Next Last

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

ComicStore / COM-23

Login validations Login with valid credentials

Edit
Comment
Assign
More
To Do
In Progress
Done
Admin

Details

Type: Test

Priority: Trivial

Component(s): None

Labels: Automation JUnit Testing

Status: TO DO (View Workflow)

Resolution: Unresolved

Description

Click to add description

Test Details

Type: Generic

Definition: login.spec.ts Login validations.Login validations Login with valid credentials

Xray uses a concatenation of the suite name and the test name as the the unique identifier for the test.

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 using playwright-test runner.

ComicStore / COM-30

Test Execution for Test Plan COM-9

Edit
Comment
Assign
More
To Do
In Progress
Done
Admin

Details

Type: Test Execution

Priority: Blocker

Component(s): None

Labels: None

Test Plan: COM-9

Test Environments: None

Status: TO DO (View Workflow)

Resolution: Unresolved

Description

Click to add description

Tests

+ Add

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Apply Rank

Show 100 entries

Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS	▶ ...
1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS	▶ ...

Showing 1 to 2 of 2 entries

First Previous Next Last

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*:

Tests

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)

Apply Mark

Rank

Key

Summary

Test Type

#Req

#Def

Assignee

Status

Show 100 entries

Columns

<input type="checkbox"/>	2	COM-24	Login validations Login with invalid credentials	Generic	0	0	Administrator	PASS	...
<input type="checkbox"/>	1	COM-23	Login validations Login with valid credentials	Generic	0	0	Administrator	PASS	...

Showing 1 to 2 of 2 entries

First Previous

Attachments

Drop files to attach, or browse.

Structure

Activity

Execution Details

Test Description

None

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type

Generic

Definition

login-spec-to login validations Login with invalid credentials

Results

Context	Output	Duration	Status
TestSuite login-spec-to	-	1 sec	PASS

Activity

As we can see here:

Execution Details

Test Description

None

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type

Generic

Definition

login-spec-to login validations Login with invalid credentials

Results

Context	Output	Duration	Status
TestSuite login-spec-to	-	1 sec	PASS

Activity

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 in order 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://playwright.dev/docs/test-intro/>
- <https://playwright.dev/>
- [Overview](#)
- [Prerequisites](#)
- [Implementing tests](#)
- [Integrating with Xray](#)
 - [API](#)
 - [JUnit XML results](#)
 - [JUnit XML results Multipart](#)
 - [Jenkins](#)
 - [JUnit XML](#)
 - [JUnit XML multipart](#)
 - [Jira UI](#)
- [Tips](#)
- [References](#)