

Testing web applications using Cypress



What you'll learn

Prerequisites

- [Integrating Cypress](#)
 - [Define tests using Cypress](#)
 - [Run the test and push the test report to Xray](#)
 - [Validate in Jira that the test results are available](#)
 - [JUnit XML results](#)
 - [Jira UI](#)

Tips

References

Source-code for this tutorial

- code is available in [GitHub](#)

Overview

Cypress is a JavaScript based testing framework for test automation. Cypress is often compared to Selenium, but it is different; unlike Selenium that is executed outside of the browser Cypress is executed within it, in the same run loop as your application.

Cypress runs in a NodeJS server process that allows Cypress and the NodeJS server to constantly communicate, synchronize, and perform tasks on behalf of each other. This provides Cypress the ability to respond to the application's events in real time, and at the same time work outside of the browser for tasks that require a higher privilege.



Cypress and Cucumber

If you're using Cypress and Cucumber (i.e. using Gherkin test scenarios), please see the tutorial [Testing using Cypress and Cucumber in JavaScript](#) instead.

Prerequisites

For this example we will use [Cypress](#) to write tests that aim to validate the [Cypress todo example](#).

We will need:

- Access to a [Cypress todo example](#) site that we aim to test
- [Cypress](#) installed in your machine

To start using the [Cypress](#) please follow the [Get Started](#) documentation.

The tests consists in validating the operations over todo's elements of the [Cypress todo example](#), for that we have defined several tests to:

- Validate that we can add new todo items;
- Validate that we can check an item as completed;
- Validate that we can filter for completed/uncompleted tasks;
- Validate that we can delete all completed tasks.

The target web application is a simple "todos" made available by Cypress.

todos

▼

What needs to be done?

☐

Pay electric bill

☐

Walk the dog

2 items left

All

Active

Completed

Double-click to edit a todo
Forked from [TodoMVC](#)

Each of these tests will have a series of actions and validations to check that the desired behavior is happening as we can see below:

todo.cy.js

```

describe('example to-do app', () => {
  beforeEach(() => {
    cy.visit(Cypress.config('baseUrl'))
  })

  it('can add new todo items', () => {
    const newItem = 'Feed the cat'
    cy.get('[data-test=new-todo]').type(`${newItem}{enter}`)

    cy.get('.todo-list li')
      .should('have.length', 3)
      .last()
      .should('have.text', newItem)
  })

  it('can check an item as completed', () => {
    cy.contains('Pay electric bill')
      .parent()
      .find('input[type=checkbox]')
      .check()

    cy.contains('Pay electric bill')
      .parents('li')
      .should('have.class', 'completed')
  })

  context('with a checked task', () => {
    beforeEach(() => {
      cy.contains('Pay electric bill')
        .parent()
        .find('input[type=checkbox]')
        .check()
    })

    it('can filter for uncompleted tasks', () => {
      cy.contains('Active').click()

      cy.get('.todo-list li')
        .should('have.length', 1)
        .first()
        .should('have.text', 'Walk the dog')

      cy.contains('Pay electric bill').should('not.exist')
    })

    it('can filter for completed tasks', () => {
      cy.contains('Completed').click()

      cy.get('.todo-list li')
        .should('have.length', 1)
        .first()
        .should('have.text', 'Pay electric bill')

      cy.contains('Walk the dog').should('not.exist')
    })

    it('can delete all completed tasks', () => {
      cy.contains('Clear completed').click()

      cy.get('.todo-list li')
        .should('have.length', 1)
        .should('not.have.text', 'Pay electric bill')

      cy.contains('Clear completed').should('not.exist')
    })
  })
})

```

The tests are simple but let's look into two differences that allow a little more control, the first one is the possibility to use hooks like *beforeEach* to, as the name implies, execute some operations before each test execution. In this example we are accessing the target page before each test avoiding repeating this instruction in each test.

beforeEach

```
...
beforeEach(() => {
  cy.visit('https://example.cypress.io/todo')
})
...
```

The other one helps in the test organization and have a direct effect on how the results will be written in the result file, in our case we are using *context* (but we could use *describe* or *specify*). This will group the tests beneath into the same testsuite.

context

```
...
context('with a checked task', () => {
  ...
```

These tests are defined to validate the application ability to manage todo's by accessing the [Cypress todo example](#) and performing operations that will generate an expected output.

Once the code is implemented it can be executed with the following command:

```
npx cypress run
```

The results are immediately available in the terminal.

```
=====
(Run Starting)

Cypress:      12.4.0
Browser:      Electron 106 (headless)
Node Version: v14.16.0 /usr/local/bin/node
Specs:        1 found (todo.cy.js)
Searched:     cypress/e2e/*

Running: todo.cy.js (1 of 1)

(Results)

Tests:        0
Passing:      0
Failing:      0
Pending:      0
Skipped:      0
Screenshots:  0
Video:        false
Duration:     4 seconds
Spec Ran:     todo.cy.js

=====
(Run Finished)

Spec                                Tests  Passing  Failing  Pending  Skipped
✓ todo.cy.js                        0:04    6         6        -        -        -
✓ All specs passed!                 0:04    6         6        -        -        -
```

In this example, all tests have succeed, as seen in the previous terminal screenshot. It generates the following JUnit XML report.

JUnit Report

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites name="Mocha Tests" time="4.404" tests="6" failures="0">
  <testsuite name="Root Suite" timestamp="2023-01-30T17:46:57" tests="0"
file="cypress/e2e/todo.cy.js" time="0.000" failures="0">
    </testsuite>
    <testsuite name="example to-do app" timestamp="2023-01-30T17:46:57"
tests="3" time="0.000" failures="0">
      <testcase name="example to-do app displays two todo items by default"
time="0.842" classname="displays two todo items by default">
        </testcase>
        <testcase name="example to-do app can add new todo items" time="0.477"
classname="can add new todo items">
          </testcase>
          <testcase name="example to-do app can check off an item as completed"
time="0.267" classname="can check off an item as completed">
            </testcase>
          </testsuite>
          <testsuite name="with a checked task" timestamp="2023-01-30T17:47:00"
tests="3" time="1.060" failures="0">
            <testcase name="example to-do app with a checked task can filter for
uncompleted tasks" time="0.345" classname="can filter for uncompleted
tasks">
              </testcase>
              <testcase name="example to-do app with a checked task can filter for
completed tasks" time="0.350" classname="can filter for completed tasks">
                </testcase>
                <testcase name="example to-do app with a checked task can delete all
completed tasks" time="0.341" classname="can delete all completed tasks">
                  </testcase>
                </testsuite>
              </testsuites>
            </testcase>
          </testsuites>
        </testcase>
      </testcase>
    </testcase>
  </testcase>
</testsuites>
```

Notes:

- You can invoke Cypress locally and use it to assist you to write and execute tests with: `npm run cypress:open`
- Use `cypress.config.js` to define configuration values such as taking screenshots, recordings or the reporter to use (more info [here](#)).
- Different parameters can be used in the command line (more info [here](#))
- We are using JUnit reporter but others are available (more info [here](#))

Integrating with Xray

As we saw in the previous example, where we are producing JUnit reports with the test results. 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](#), and for that 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.

Authentication

The request made will look like:

```
curl -H "Content-Type: application/json" -X POST --data '{ "client_id": "CLIENTID", "client_secret": "CLIENTSECRET" }' https://xray.cloud.getxray.app/api/v2/authenticate
```

The response of this request will return the token to be used in the subsequent requests for authentication purposes.

JUnit XML results

Once you have the token we will use it in the API request with the definition of some common fields on the Test Execution, such as the target project, project version, etc.

```
curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer $token" --data @"todo-results.xml" https://xray.cloud.getxray.app/api/v2/import/execution/junit?projectKey=XT&testPlanKey=XT-601
```

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

Projects / Xray Tutorials / XT-601

tutorial-js-cypress

Attach Create subtask Link issue Tests ...

Description

Add a description...

Tests

Tests Test Executions

Add Tests Create Test Execution View on board

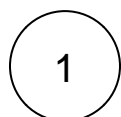
Overall Execution Status All Environments, final status

6 PASSED TOTAL TESTS: 6

Key	Summary	Assignee	#Test Executions	Dataset	Test Type	Latest Status
XT-603	example to-do app displays two todo items by default		1		Generic	PASSED
XT-604	example to-do app can add new todo items		1		Generic	PASSED
XT-605	example to-do app can check off an item as comple...		1		Generic	PASSED
XT-606	example to-do app with a checked task can filter for...		1		Generic	PASSED
XT-607	example to-do app with a checked task can filter for...		1		Generic	PASSED
XT-608	example to-do app with a checked task can delete a...		1		Generic	PASSED

Jira UI

Jira UI



Create a Test Execution linked to the Test Plan that you have.

tutorial-js-cypress

Attach Create subtask Link issue Tests ...

Description

Add a description...

Tests

Tests Test Executions

Create Tests Create Test Execution View on board

Overall Execut All tests... With status... All Environments, final status

6 PASSED TOTAL TESTS: 6

Only My Issues Filters 10 Columns

Key	Summary	Assignee	#Test Executions	Dataset	Test Type	Latest Status
<input type="checkbox"/> XT-603	example to-do app displays two todo items by default		1		Generic	PASSED
<input type="checkbox"/> XT-604	example to-do app can add new todo items		1		Generic	PASSED
<input type="checkbox"/> XT-605	example to-do app can check off an item as complete...		1		Generic	PASSED
<input type="checkbox"/> XT-606	example to-do app with a checked task can filter for...		1		Generic	PASSED
<input type="checkbox"/> XT-607	example to-do app with a checked task can filter for...		1		Generic	PASSED

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

Create planned Test Execution

Project

Xray Tutorials

Summary *

Test Execution for Test Plan XT-601

Assignee

Cristiano Cunha

Choose a user to assign the Test Execution

File Version/s

Select...

Test Environment

Select...

☒ Go to Test Execution

Create Cancel

Open the Test Execution and import the JUnit XML report.

[illegible]

Choose the results file and press *"Import"*

Import Execution Results

Choose file

No file chosen

The file with the execution results for the Test Execution.

Not found

SubmitCancel

5

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

Projects / Xray Tutorials / XT-609

Test Execution for Test Plan XT-601

AttachCreate subtaskLink issueTests

Description

Add a description...

Tests

Add TestsView on board

Overall Execution Status

6 PASSEDTOTAL TESTS: 6

Only My Test RunsFilters10Columns

Rank	Key	Summary	Test Type	Dataset	#Defects	Status	Actions
1	XT-603	example to-do app displays two todo items by default	Generic		0	PASSED	
2	XT-604	example to-do app can add new todo items	Generic		0	PASSED	
3	XT-605	example to-do app can check off an item as comple...	Generic		0	PASSED	
4	XT-606	example to-do app with a checked task can filter for...	Generic		0	PASSED	
5	XT-607	example to-do app with a checked task can filter for...	Generic		0	PASSED	
6	XT-	example to-do app with a checked task can delete a...	Generic		0	PASSED	

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

Projects / Xray Tutorials / XT-603

example to-do app displays two todo items by default

AttachCreate subtaskLink issueTest details

Description

Add a description...

Test details

Test detailsPreconditionsTest SetsTest PlansTest Runs

Test Type

Generic

Definition

displays two todo items by default.example to-do app displays two todo items by default

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 Cypress.

Projects / Xray Tutorials / XT-609

Test Execution for Test Plan XT-601

Attach Create subtask Link issue Tests ...

Description
Add a description...

Tests
Add Tests View on board

Overall Execution Status

6 PASSED TOTAL TESTS: 6

Rank	Key	Summary	Test Type	Dataset	#Defects	Status	Actions
1	XT-603	example to-do app displays two todo items by default	Generic		0	PASSED	...
2	XT-604	example to-do app can add new todo items	Generic		0	PASSED	...
3	XT-605	example to-do app can check off an item as comple...	Generic		0	PASSED	...
4	XT-606	example to-do app with a checked task can filter for...	Generic		0	PASSED	...
5	XT-607	example to-do app with a checked task can filter for...	Generic		0	PASSED	...

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*:

Projects / Xray Tutorials / XT-609

Test Execution for Test Plan XT-601

Attach Create subtask Link issue Tests ...

Description
Add a description...

Tests
Add Tests View on board

Overall Execution Status

6 PASSED TOTAL TESTS: 6

Rank	Key	Summary	Test Type	Dataset	#Defects	Status	Actions
1	XT-603	example to-do app displays two todo items by default	Generic		0	PASSED	...
2	XT-604	example to-do app can add new todo items	Generic		0	PASSED	...
3	XT-605	example to-do app can check off an item as comple...	Generic		0	PASSED	...
4	XT-606	example to-do app with a checked task can filter for...	Generic		0	PASSED	...
5	XT-607	example to-do app with a checked task can filter for...	Generic		0	PASSED	...

As we can see here:

Projects / Xray Tutorials / Test Plans / Test Plans / XT-601 / Test Executions / XT-609 / Test XT-603

example to-do app displays two todo items by default

Execute with Exploratory App Import Execution Results

Execution Status: PASSED

Findings: No findings reported

Test details

Definition: @Select two todo items by default example to-do app displays two todo items by default

Results

Context	Output	Duration	Status
Testbed example to-do app		44.01s	PASSED

Activity

Tips

- after results are imported, in Jira Tests can be linked to existing requirements/user stories, so you can track the impacts on 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 a identifier of the device/application used to interact with the system (e.g. browser, mobile OS).

References

- <https://www.cypress.io/>
- <https://docs.cypress.io/guides/overview/why-cypress>