

Testing web applications using Cypress



What you'll learn

- [Prerequisites](#)
- [Integrating Cypress](#)
 - [Run the test and push the test report to Xray](#)
 - [Validate in Jira that test results are available](#)
 - [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.

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

AllActiveCompleted

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:      6
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                        00:04   6        6       -        -        -
✓ All specs passed!                 00:04   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>
```

Notes:

- You can invoke Cypress locally and use it to assist you to write and execute tests with: `npm 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.

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=@todo-results.xml" 'http://<LOCAL_JIRA_INSTANCE>/rest/raven/2.0/import/execution/junit?projectKey=XT&testPlanKey=XT-401'
```

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.

The screenshot shows the Jira Test Execution interface for 'Execution results - todo-results.xml - [1675182946651]'. The interface includes a top navigation bar with 'Edit', 'Comment', 'Assign', 'More', 'To Do', 'In Progress', 'Done', and 'Admin'. The 'Details' section shows the test execution type as 'Test Execution', priority as 'Trivial', labels as 'None', test plan as 'XT-401', and test environments as 'None'. The 'Description' section states 'Execution results imported from external source'. The 'Tests' section shows an 'Overall Execution Status' of 5 PASS. Below this, a table lists the tests with columns for Rank, Key, Summary, Test Type, #Req, #Def, Assignee, Dataset, and Status. The table contains three visible rows, all with a status of 'PASS'.

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Dataset	Status
1	XT-403	example to-do app can add new todo items	Generic	0	0	Xpand IT Admin		PASS
2	XT-404	example to-do app can check an item as completed	Generic	0	0	Xpand IT Admin		PASS
3	XT-405	example to-do app with a checked task can filter for uncompleted tasks	Generic	0	0	Xpand IT Admin		PASS

Jira UI

Jira UI

1

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

Xray Tutorials / XT-401

tutorial-js-cypress

Edit

Comment

Trigger Jenkins build

More ▾

To Do

In Progress

Done

Admin ▾

Details

Type:

Test Plan

Trivial

Priority:

None

Status:

10:50 (View Workflow)

Resolution:

Unresolved

Description

Click to add description

Tests

Add Tests ▾

Create Test Execution ▾

Test Plan Board

Overall Execution 5

All tests

With status

All tests

5 PASS

Total Tests: 5

Filter(s)

Show 10 ▾ entries

All Environments ▾

Columns ▾

Key	Summary	Requirements	#Test Executions	Issue Assignee	Test Type	Latest Status
<input type="checkbox"/> ▶ XT-403	example to-do app can add new todo items		1	Xpand IT Admin	Generic	PASS
<input type="checkbox"/> ▶ XT-404	example to-do app can check an item as completed		1	Xpand IT Admin	Generic	PASS

Fill in the necessary fields and press "Create"

Create Issue

⚙️ Configure Fields ▾

Project*

Xray Tutorials (XT) ▾

Issue Type*

Test Execution ▾ ⓘ

General Test Execution Details

Description

Summary*

Test Execution for Test Plan XT-401

Style ▾ B I U A ▾ ↕ ⌂ 🔍 🗑 + ▸ ⌨

Visual Text

Reporter*

Xpand IT Admin

Start typing to get a list of possible matches.

Assignee

Xpand IT Admin ▾
Assign to me

Priority

Trivial ▾ ⓘ

Affects Version/s

None

Fix Version/s

None

Original Estimate

(eg. 3w 4d 12h) ⓘ

The original estimate of how much work is involved in resolving this issue.

Remaining Estimate

(eg. 3w 4d 12h) ⓘ

An estimate of how much work remains until this issue is resolved.

Component/s

None

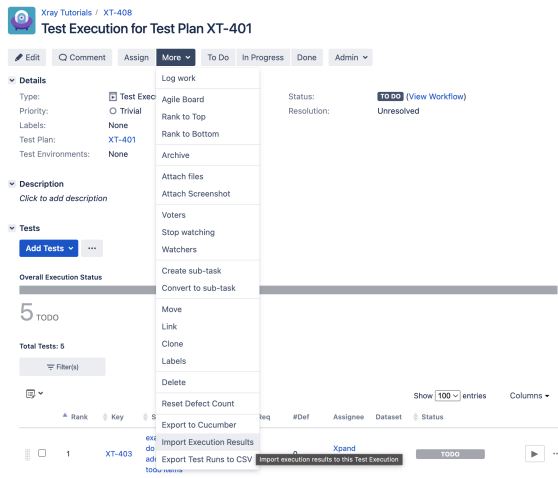
Attachment

📎 Drop files to attach, or browse.

↩ Redirect to Test Execution

Create

Cancel



4

Choose the results file and press "Import"

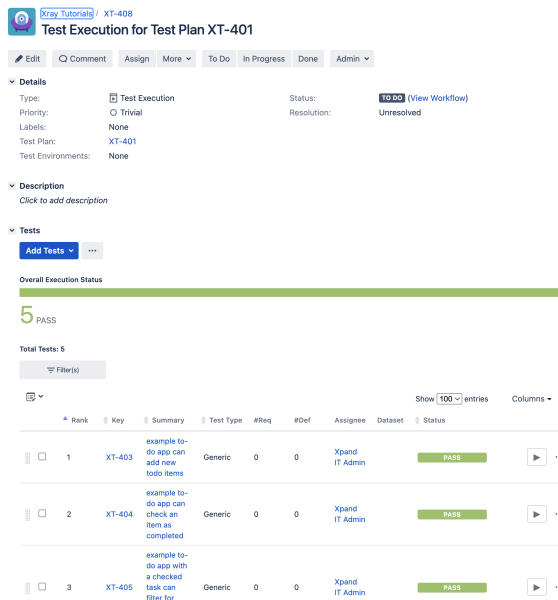
Import Execution Results

No file chosen

The file with the execution results for the Test Execution.

5

The Test Execution is now updated with the test results imported



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.

Xray TutorialstXT-403

example-to-do app can add new todo items

EditCommentAssignMoreTo DoIn ProgressDoneAdmin

Details

Type:TestPriority:TrivialLabels:NoneStatus:10 DO (View Workflow)Resolution:Unresolved

Description

Click to add description

Test Details

Type:GenericDefinition:can add new todo items.example-to-do app can add new todo items

Pre-Conditions

This test is not associated with Pre-Conditions yet.

Add Pre-Conditions

Test Sets

This test is not associated with Test Sets yet.

Add Test Sets

Test Plans

Add Test Plans

Show 10 entriesColumns

Key	Summary	Test Plan Status
XT-401	tutorial-3-cypress	

Showing 1 to 1 of 1 entriesFirstPreviousNextLast

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.

Xray TutorialstXT-402

Execution results - todo-results.xml - [1675182946651]

EditCommentAssignMoreTo DoIn ProgressDoneAdmin

Details

Type:Test ExecutionPriority:TrivialLabels:NoneTest Plan:XT-401Test Environments:NoneStatus:10 DO (View Workflow)Resolution:Unresolved

Description

Execution results imported from external source

Tests

Add Tests

Overall Execution Status

5 PASS

Total Tests: 5

Filter(s)

Show 100 entriesColumns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Dataset	Status
1	XT-403	example to-do app can add new todo items	Generic	0	0	Xpand IT Admin		PASS
2	XT-404	example to-do app can check an item as completed	Generic	0	0	Xpand IT Admin		PASS
3	XT-405	example to-do app with a checked task can filter key	Generic	0	0	Xpand IT Admin		PASS

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*:

Xray Test Results - todo-results.xml - [1675182946651]

Execution results - todo-results.xml - [1675182946651]

Details

Type: Test Execution
Priority: Trivial
Labels: None
Test Plan: XT-401
Test Environments: None

Status: Unresolved
Resolution: Unresolved

Xporter

Template: Xray Test
Output format: DOCK
Export

People

Assignee: Xpand
Reporter: Xpand
Votes: 0
Watchers: 1 Stop

Date

Created: 28 minute
Updated: 28 minute

Agile

View on Board

Description

Execution results imported from external source

Tests

Add Tests

Overall Execution Status

5 PASS

Total Tests: 5

Filter

Show 100 entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Dataset	Status
1	XT-403	example to-do app can add new todo items	Generic	0	0	Xpand IT Admin		PASS
2	XT-404	example to-do app can check an item as completed	Generic	0	0	Xpand IT Admin		PASS
3	XT-405	example to-do app with a scheduled task can filter for uncompleted tasks	Generic	0	0	Xpand IT Admin		PASS

Execution Details

Execute with Exploratory

EXECUTE ONLINE

TODO

EXECUTING

FAIL

ABORTED

BLOCKED

As we can see here:

Xray Test Results - todo-results.xml - [1675182946651]

example to-do app can add new todo items

Execution Status: PASS

Started On: 31/Jan/23 4:35 PM

Finished On: 31/Jan/23 4:35 PM

Assignee: Xpand IT Admin

Execution By: Xpand IT Admin

Test Environment: None

Comment

Click to add comment

Execution Defects (0)

No defects yet...

Execution Evidence (0)

No evidence yet...

Test Details

Custom Fields

There are no Test Run Custom Fields defined

Test Description

Test Type: Generic

Definition: can add new todo items example to-do app can add new todo items

Results

Context	Output	Duration	Status
TestRun: example to-do app	-	764,000 ms	PASS

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>