Testing web applications using Cypress



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.

~	What needs to be done?	
	Pay electric bill	
	Walk the dog	
2 item	ns left All Active Completed	

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 diferences 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
<pre> context('with a checked task', () => {</pre>

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.

(<u>Run Starting</u>)					
Cypress: Browser: Node Version: Specs: Searched:	12.4.0 Electron 106 (headless) v14.16.0 (/usr/local/bin/no 1 found (todo.cy.js) cypress/e2e/*				
Running: todo. (<u>Results</u>)					
Tests: Passing: Failing: Pending: Skipped: Screenshots: Video: Duration: Spec Ran:	5 5 9 9 9 9 9 4 false 4 seconds todo.cy.js				
(<u>Run Finished</u>) Spec					
🖌 todo.cy.js		00:04			
 All specs 	passed!	00:04			

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: npx 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.



Jira UI

Jira UI



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

Projects / 😲 Xray Tutorials / 📔 XT-601							
tutorial-js-cypress							
🖉 Attach 🕑 Create subtask 🔗 Link i	ssue 👻 🔶 Test	ts					
Description							
Add a description							
Tests							
Tests Test Executions							
Add Tests V Create Test Execution V	l						View on board
Overall Execut All tests					All Enviro	nments	, final status 🐱
With status							
6 _{PASSED}							TOTAL TESTS: 6
 Only My Issues Filters 					10	~	Columns 🗸
Key Summary		Assignee	#Test Executions	Dataset	Test Type	Lates	t Status
C XT- 603 example to-do app displays two t	odo items by default		1		Generic	P	ASSED
XT- 604 example to-do app can add new to 604 example to-do app can add new to	rodo items		1		Generic	P	ASSED
 XT- 605 example to-do app can check off 	an item as comple		1		Generic	P	ASSED
XT- 606 example to-do app with a checke	d task can filter for		1		Generic	P	ASSED
 XT- 607 example to-do app with a checke 	d task can filter for		1		Generic	P	ASSED



Fill in the necessary fields and press "Create"

Create planned Test Execution

roject	
Yray Tutorials	
Aley fotoriela	
ummary *	
Test Execution for Test Plan XT-601	
ssignee	
Cristiano Cunha	~
hoose a user to assign the Test Execution	
ix Version/s	
Select	~
est Environment	
Select	~

3

Create Cancel

Open the Test Execution and import the JUnit XML report.

est Execution for Test Plan XT-601	To Do 🗸		Log work
🧬 Attach 🔄 Create sabtask 🖉 Linkissue 👻 🖸 Tests …	Details		Add Tag
bescription	Aasignoe	G C1	zi Xporter
	Reporter	(3) 04	Xray - Generate Cucumber Feature file
testa	Developm	ort Dr Creat	Xray - Import Execution Results
Add Toess 👒	View on board		Nray - Document Generator
Overall Execution Status	Labels	None	Connect Slack channel
	Freity	= Me	g Convert to Subtank
6 _{то во}	TOTAL TESTS: 6 Automatic	un ∳Rub	Move
	Kperter	Open Xp	Clane
	Test Plane	a Open Ter	Defete
	Test Drvin	onnorts Open Ter	Find your field
	More fiel	ids Original estimate, Time tra	Actions manu .
	Created 1	minute ago	Print
	Updated 5	minate ago	Export XML
Activity			Export Word
how: All Commons History Worklog Xray History	Newest first 17		Take a tour
Add a comment			Find out more



Import Execution Results

Choose file No file chosen The file with the execution results for the Test Execution.

「he ⁻	Te	st E	xecut	on is	now upda	ted wi	th th	ie tes	st resu	ults ii	mpo	rte
Projects	1 😡	Xray Tu	itorials / 🖪	XT-609								
Test	Exe	cutio	on for Te	st Plan	KT-601							
@ Att	ach	() Cr	eate subtask	🖉 Link is	sue 👻 🖸 Tests							
Descrip	tion											
Add a de	escript	tion										
Tests												
Add Te	ests -	× .									View or	h board
Overa	II Exe	ecutio	n Status									
6 _{PA}	SSED										TOTAL T	ESTS: 6
	~	Only N	ly Test Runs	Filters ~						10 🗸	Colum	15 ¥
			Summary :			Test	Dataset	#Defects :	Status :			Actions
	Rank	t≑ Key∶				Type :						
-	Rank	xT- 603	example to-c	o app display	s two todo items by defa	Type : ult Generic		0	PASSED		≣D	
	Rank 1 2	xt- 603 XT- 604	example to-o	o app display o app can ad	s two todo items by defa d new todo items	Type : ult Generic Generic		0 0	PASSED		≣D ≣D	
	Rank 1 2 3	xT- 603 XT- 604 XT- 605	example to-o example to-o example to-o	o app display o app can ad o app can ch	s two todo items by defa d new todo items ack off an item as comple	Type : ult Generic Generic e Generic		0 0 0	PASSED PASSED PASSED		≣D ≣D ≣D	··· ···
	Rank 1 2 3 4	xT- 603 XT- 604 XT- 605 XT- 606	example to-c example to-c example to-c example to-c	o app display o app can ad o app can ch o app with a i	s two todo items by defa d new todo items bock off an item as comple checked task can filter fo	Type : ult Generic Generic e Generic f Generic		0 0 0	PASSED PASSED PASSED PASSED		=D =D =D =D	
	Rank 1 2 3 4 5	xt- 603 XT- 604 XT- 605 XT- 606 XT- 606 XT- 607	example to-co example to-co example to-co example to-co example to-co	o app display o app can ad o app can ch o app with a o app with a	s two todo items by defa i new todo items bok off an item as comple checked task can filter fo checked task can filter fo	Type : ult Generic Generic e Generic r Generic r Generic		0 0 0 0	PASSED PASSED PASSED PASSED PASSED		ED ED ED ED ED	···· ··· ···

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 de	efault
@ Attach ☑ Create subtask ⊘ Link issue 🗸 🖌 Test details	
Description	
Add a description	
Test details	
🔲 Test details 🧕 Preconditions 💽 Test Sets 💋 Test Plans 🗕 Te	ust Runs
Test Type	Test Repository
Generic 🗸	
Definition	
displays two todo items by default.example to-do app displays two todo items by defa	ult

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	/ 1	xray IL	itoriais / 🖬	X1-609									
Test	Exe	cutio	on for Te	est Plan	XT-60	1							
@ Att	ach	Cr Cr	eate subtask	🖉 Link	issue 👻	O Tests	•••						
Descript	tion												
Add a de	script	tion											
Tests													
Add Te	ists	~										View on	board
Overa	ll Ex	ecutio	n Status										
6	PPED											TOTAL TE	ESTS: 6
U PAC	3350											TO TAL TE	
	~	Only M	y Test Runs	Filters ~							10 🗸	Columns	s v
	Rank	te Key :	Summary :				Test Type :	Dataset	#Defects	Status			Actions
0	1	XT- 603	example to-	do app displa	iys two todo	items by defa	ult Generic		0	PASSED		ΞD	
	2	XT- 604	example to-	do app can a	dd new todo	items	Generic		0	PASSED		≣D	
	3	XT- 605	example to-	do app can c	heck off an it	em as comple	Generic		0	PASSED		ΞD	
	4	XT- 606	example to-	do app with a	checked tas	ik can filter fo	r Generic		0	PASSED		≣D	
	5	XT- 607	example to-	do app with a	checked tas	sk can filter fo	r Generic		0	PASSED		≣D	

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:

Pages / Kny Jacké / IntelPer / 1500 / IntelBearder / 1500 / IntelBearder / 1500 / IntelBearder / 1500 / IntelBearder / IntelBe										
Execution Status PASSED 10	Timer Becalcolo D No time logged	Started On St[Jam/2523 09:40 AM Finished On St[Jam/2523 09:40 AM	Assignee Oristiano Cunha Executed By Oristiano Cunha	Versions - Revision -	Text Environments					
> Findings O										
Test details										
 Definition 	mele In-da ana disebua bua kala kama	by circlas it								
- Results O		.,								
Centext TestSuite example ta-da app		Childrent -			Duration 842ms	Status 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