Requirements

Video

Cypress: Requirements - Exercise

Required Practice

Start by reading the following document about Cypress: **https://docs.cypress.io/guides/overview/why-cypress** and the installation guide: **https://docs.cypress.io/guides/getting-started/installing-cypress**

In this practice we are going to install Cypress for the first time in order to explore the structure and different files created in the project, on the other hand, we are going to execute the example test cases to familiarize ourselves with the Cypress dashboard and the executions.

Make sure to run all the example scenarios, and to check every interaction on them, in order to fully understand all cypress features.

Additional Resources

Installing Cypress

Writing E2E Test

Intro, install & configuration

Video I

Video II

Video III .

Cypress: Accessing elements

Complementary Practice

Start by reading the following document about E2E test with cypress: **https://docs.cypress.io/guides/end-to-end-testing/writing-your-first-end-to-end-test**

In this exercise, we are going to use Cypress to automate the process of logging into a website. So, we will interact with input fields, buttons, and labels to verify the successful login. The form with which we are going to interact is the following: **http://uitestingplayground.com/sampleapp**

Initially we should install and create a new Cypress project. Once the project has been created, we should obtain all the necessary elements to fill in and submit the form, create a scenario to automate this flow and validate the successful login message is present.

Additional Resources

[Table of contents]

[Accessing elements]

Assertions

Video

Cypress: Assertions

Required Practice

Start by reading the following document about Assertions with cypress: **https://docs.cypress.io/guides/references/assertions**

In this exercise, we are going to use Cypress to automate the process of filling a form with different types of data and elements. The form with which we are going to interact is the following: **https://qavalidation.com/demo-form/**

Initially we should install and create a new Cypress project. Once the project has been created, we should obtain all the necessary elements to fill in and submit the form. On the other hand, we should define the test cases related to validating the data types, sending empty data, sending required data and other test cases that we consider can be defined according to the form structure and its possible response messages. Define and implement at least 10 scenarios on a new Cypress Framework.

Additional Resources

Assertions

Validations

Variables and Fixtures

Video

Cypress: Fixtures

Complementary Practice

Start by reading the following document about fixtures in cypress: **https://docs.cypress.io/api/commands/fixture**

In this practice we are going to test a login form on a website: **http://uitestingplayground.com/sampleapp**. The login form has two input fields: "*username*" and *"Password*", as well as a *"Login"* button and we should test the login functionality using different sets of credentials stored in a fixture file.

For the definition of the required test cases we should take into account the following objectives:

- Load test data from a fixture file.
- Perform data-driven testing by iterating through the test data.
- Test the login functionality with valid and invalid credentials.
- Verify the expected behavior after attempting to log in.

Using a new Cypress project, implement at least 4 different scenarios, all of which must use fixtures to retrieve the different data.

Additional Resources

Fixture

Data driven testing

Waits

Video

Cypress: Waits

Required Practice

Start by reading the following document about waits in cypress: **https://docs.cypress.io/api/commands/wait**

In this practice we are going to implement waiting strategies to simulate the processing of an AJAX request to a web server. We need to create tests that should be able to wait for an element to show up.

The site to test is the following: **http://uitestingplayground.com/ajax**. There we will find a button that, when pressed, will simulate the launch of a request. Once the button is clicked, there will be a waiting time to obtain a response message. The defined test cases should validate that the message is displayed using waiting strategies with Cypress.

Make sure to create at least 3 scenarios, every one of them using a different waiting strategy to wait for the success message to show up. (Waiting strategies should be: waiting a specific time, waiting with a BDD expectation, and waiting for the service to return a response).

Additional Resources

Waits

Assertions

Hooks

Video

Cypress: Hooks

Required Practice

Start by reading the following document about Hooks with cypress: **https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Asset-Files**

In this exercise, we are going to understand the Hooks functionality. Let's imagine that we will be working with a simple e-commerce website and we need to define Cypress tests that implement different types of hooks to interact with the site.

For this practice it will be necessary to define actions for each type of hook:

- **before():** Runs once before all tests.
- **beforeEach():** Runs before each test.
- **afterEach():** Runs after each test.
- **after():** Runs once after all tests.

We should only define the structure of the test cases, the different types of hooks and the actions that could be executed on each of them without implementing the specific logic, for example:

```
before(() => {
```

```
  cy.visit('https://www.ecommerce-example.com');
});
```

Feel free to use pseudocode, comments and console logs to justify any needed logic.

Define on a new Cypress project at least 10 test cases divided on at least 2 different specs, each with their respective hooks properly organized.

Tip: If you need some visual guide to define your tests, you can use any of the following ecommerce examples:

- **https://www.saucedemo.com/**
- **https://demoblaze.com/**
- **https://automationexercise.com/**

Additional Resources

Writing and organizing test

Cypress Hooks

Advanced Cypress: API Testing

Video

Advanced Cypress: API Testing - Exercise

Required Practice

API to test: https://swapi.dev/

For this practice, we are going to test a real API using the version of cypress you like, and an appropriate framework architecture to organize and reuse the API calls and implement test scenarios.

We will use the Star Wars API (swapi) for this exercise, the base URL is **https://swapi.dev/** and we need to Implement tests (a different test for each request) performing the following actions:

1. Test the endpoint **people/2/** and check the success response, the skin color to be gold, and the amount of films it appears on (should be 6).
2. Request the endpoint of the second movie in which **people/2/** was present (using the response from **people/2/**). Check the release date to be in the correct date format, and the response to include characters, planets, starships, vehicles and species, each element including more than 1 element.
3. Request the endpoint of the first planet present in the last film's response (using the previous response). Check the gravity and the terrains matching the exact same values returned by the request (Use fixtures to store and compare the data of terrains and gravity).
4. On the same response from the planet, grab the url element on the response, and request it. Validate the response being exactly the same from the previous one.
5. Request the **/films/7/** and check the response having a 404 code.

Additional Resources

Request Intercept

Advanced Cypress: POM

Video

Advanced Cypress: POM

Complementary Practice

Start by reading the following document about POM with cypress: **https://www.lambdatest.com/learning-hub/cypress-page-object-model**

In this exercise, we are going to implement the design pattern Page Object Model to automate the process of logging into a website. So, we will interact with input fields, buttons, and labels to verify the successful login. The form with which we are going to interact is the following: **https://practicetestautomation.com/practice-test-login/**

For this exercise we should take into account the following instructions:

- Create the page objects for the login and home pages of the website.
- According to the page object, obtain the different elements necessary for the execution of the test cases.
- Define the different actions to be executed making use of the elements.
- Create the login spec file and implement tests for successful login and incorrect credential input.
- Create page objects for the Blog and Courses pages from the given web page

- Create new spec files to test the respective navigation to the Blog and Courses pages, and their correct display (Checking visibility of their important elements.

Additional Resources

Cypress Page Object Model

How to Implement Cypress Page Object Model (POM)

Advanced Cypress: Cucumber

Video

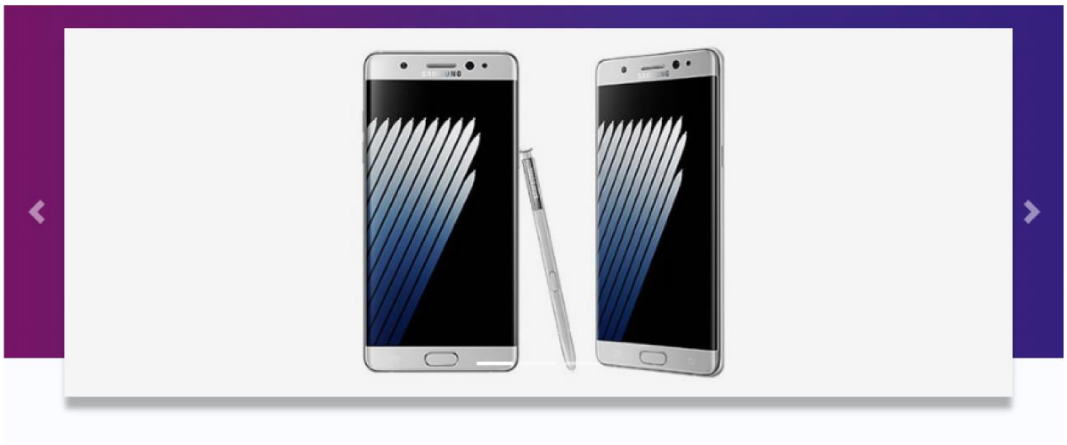
Advanced Cypress: Cucumber - Exercise

Complementary Practice

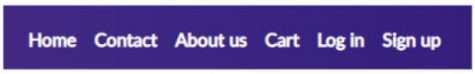
For this practice, we need to create a framework from scratch using any version of cypress and cucumber to test the web page: **https://www.demoblaze.com/**.

The framework should include test cases for the following functionalities:

1. Test the featured items carousel. It should be in a feature including at least 3 scenarios.



2.Test Navigation to all header links. It should be in a feature including at least 3 scenarios.



3.Test the categories selection, product selection, adding product to cart, buying cart, removing element from cart. All of the defined scenarios for these functionalities should be on a feature.

Keep in mind the following instructions and tips:

- The framework should be organized using best practices and following what suits best to the software under test.
- All defined scenarios should be tagged in any way you decide, including at least tags for organization and for different suites (regression, smoke, sanity), additionally you could add some tags for hooks.
- The framework should include at least the use of: cy.request, cy.intercept, POM, fixtures.
- Test cases to implement should be designed following best practices and correct organization.
- The use of cucumber is mandatory.

Additional Resources

Preprossesor

Advanced Cypress: Reporting

Video

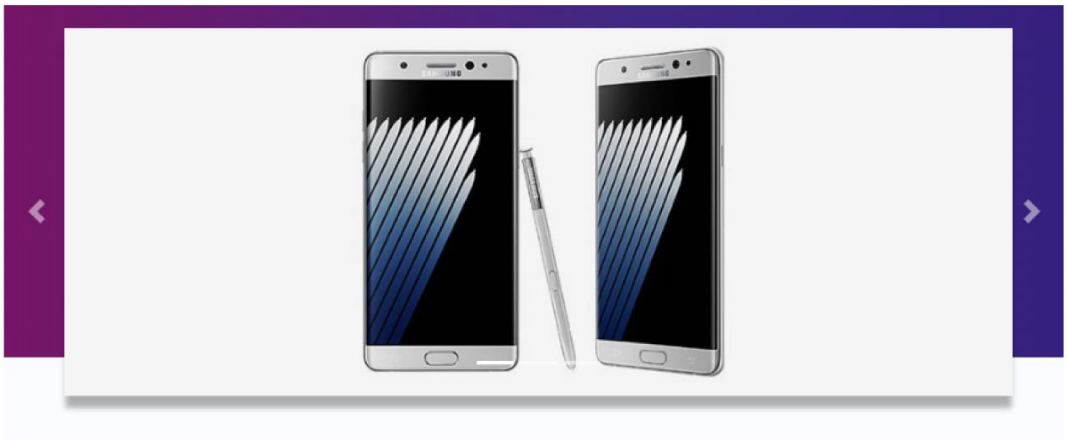
Advanced Cypress: Reporting- Exercise

Required Practice

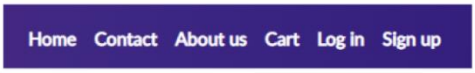
In this practice we will continue working on the previous module exercise which is related to the automation of different test cases for the web page **https://www.demoblaze.com/**.

Remember, the framework should include test cases for the following functionalities:

1. Test the featured items carousel. It should be in a feature including at least 3 scenarios.



2. Test Navigation to all header links. It should be in a feature including at least 3 scenarios.



3. Test the categories selection, product selection, adding product to cart, buying cart, removing element from cart. All of the defined scenarios for these functionalities should be on a feature.

Also, the framework should consider the following instructions and tips:

- The framework should be organized using best practices and following what suits best to the software under test.
- All defined scenarios should be tagged in any way you decide, including at least tags for organization and for different suites (regression, smoke, sanity), additionally you could add some tags for hooks.
- The framework should include at least the use of: cy.request, cy.intercept, POM, fixtures.
- Test cases to implement should be designed following best practices and correct organization.
- The use of cucumber is mandatory.

When the implementation is complete, it is necessary that you read and learn about the integration and configuration of the Allure reporter in the following link: **https://kailash-pathak.medium.com/how-to-integrate-allure-report-in-cypress-2224f4ae815**

Once we have read the integration guide, integrate Allure to the project of the previous module to implement automated test case execution reporting.

Make sure to execute the framework and check if the generated reports are working correctly.

Additional Resources

Allure reports

Automated software testing

Test Case