Cypress

# What is end to end testing | Cypress introduction (Youtube)

It provides a browser with head not a headless browser.

We can use Cypress to test any web application built on any type of technology.

There is not feature to test things on separate tab like testing third party login.

# Cypress End-to-End Testing (Youtube)

It has four key folders – fixtures, integrations, support and plugins.

Best practices – we should avoid using id’s or css classes to select element from the DOM, it will make the test brittle, because those things are likely to change. We should either use data attribute or actual component name itself.

Text

Description automatically generated

# Introduction to automation testing with Cypress.io (Non-selenium framework) (Youtube)

Whereas selenium executes remote commands through the network, Cypress runs in the same run-loop as our application. Other tools like protector uses selenium under the hood unlike Cypress.

With Cypress, all related things are available out of the box –

Graphical user interface, diagram

Description automatically generated

It provides features like real time loading, time travel, consistent results.

# Cypress in a Nutshell (Youtube)

It is a tool for reliably testing anything that runs in web browser.

Our application will be pulled-in using an iframe –

Graphical user interface, text, application, chat or text message

Description automatically generated

Cypress command API – it is a chained API where subject is passed through the chain.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Test commands are executed in a deterministic manner, resulting in flake-free testing. Cypress will automatically wait for this assertion “.should” (4 seconds by default). So we don’t need to write code for wait and sleep until element is ready –

Graphical user interface, application

Description automatically generated

By using the task command we can execute the javascript at the system level.

By using below approach, using store, we can directly dispatch login functionality through programmatically through our actual application code –

Text

Description automatically generated

In cypress we can also stub network response with fixtures by using cy.server() command.

To run the Cypress in the headless mode use “cypress run” command.

We can record results to Cypress dashboard by using “npx cypress run –record” command.

We can use --parallel flag to optimize CI usage with Parallelization, Cypress will automatically load balance the test files by using command “npx cypress run --record --parallel".

Refer cypress example recipes - <https://github.com/cypress-io/cypress-example-recipes>

# Testing The Way It Should Be (aka Intro Into Cypress) (Youtube)

People suggestion for creating a new tool -

Diagram, text

Description automatically generated

Timeline

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# [Other Resources]

Selenium and similar tools were designed to test applications that require a full-page refresh. Supporting SPAs with Ajax data fetching was an afterthought. This lead to many issues with timing and flakey tests. Tests would sometimes fail due to slow API requests or network latency. Fixing these flakey tests typically required adding sleep statements and increasing timeouts. This made the test code more brittle. Not to mention extremely slow.

It’s worth mentioning Google’s Puppeteer has inner access to web browser events, allowing us to wait on things like Ajax calls. However, writing tests with Puppeteer requires more initial setup work and more effort to write each test than it should.

Cypress.io is a relatively new framework. It overcomes many shortcomings found in Selenium, Phantom.js, and others before them. It uses an event-based architecture that hooks into Google Chrome’s lifecycle events. This enables it to wait for things like Ajax requests to complete without using a polling/timeout mechanism. This leads to reliable and fast tests. In short, it is truly the future of E2E testing and how it should have been in the first place.

You can run Cypress in two modes: full-mode and headless-mode. The former lets you see your app’s UI and tests performed one step at a time. This mode is excellent for building up your test suite and debugging. The latter is great for a Continuous Integration (CI) environment. Another use case for headless-mode: you just want to make sure you haven’t broken anything with new changes but don’t care about the detailed steps.

Headless-mode is useful for running on a Continuous Integration (CI) server like CircleCI. Once you start writing tests more regularly as part of your development, you should invest time in getting a CI server configured so that every git commit runs the entire test suite.

Cypress is an end-to-end framework that was created by Brian Mann, who wanted to solve some pain points that a lot of developers face when writing integration tests: hard to write, unreliable and too slow. Similar to TestCafe, it was built on top of Node, with no dependencies on Selenium, and is a standalone testing framework that supports Javascript.

You can have a 100% code coverage with unit-tests that test all your components in isolation, but your application might still fail when components start to communicate with each other.

The real important tests are the ones that test functionalities that your users use every day. These are things like: “Can a user buy a product?” and “Will my order be shipped to the right address if I change the address later?” These kinds of things are impossible to test with unit tests, as they use all components of your application.