Automation Testing

simplilearn

Cypress Configuration

A Day in the Life of an Automation Test Engineer

Mark now understands how to install Cypress and work with its framework.

Now, he has been asked to perform end to end testing where he needs to replicate user behavior on the application .

To achieve this, he will learn writing the end to end test cases which will help him to find a solution for the given scenario.

Learning Objectives

By the end of this lesson, you will be able to:

- Describe Spec and Config Files
- State launchpad
- Define E2E Testing
- Brief the types of E2E Testing

Add Test Files

Writing and Organizing Tests

Cypress will automatically support learning out a suggested folder structure after adding a new project.

Example

```
E2E:
/cypress.config.js
/cypress/fixtures/example.json
/cypress/support/commands.js
/cypress/support/e2e.js
```

Writing and Organizing Tests

While Cypress allows users to specify where the tests, fixtures, and support files are stored, users recommend that they use the structure for the first project.

Example

```
Component:
/cypress.config.js
/cypress/fixtures/example.json
/cypress/support/commands.js
/cypress/support/component.js
/cypress/support/component-index.html
```

Writing and Organizing Tests

In the configuration file, users can change the folder configuration.

Example

```
Both:
/cypress.config.js
/cypress/fixtures/example.json
/cypress/support/commands.js
/cypress/support/e2e.js
/cypress/support/component.js
/cypress/support/component-index.html
```

Spec Files

The default location for the test files is cypress/e2e, although this can be changed.

.js

.jsx

.ts

.tsx

.coffee

.cjsx

Config Files

Cypress.config configuration is only valid for the current spec file. Cypress runs each spec file independently, exiting the browser between specs.

Syntax

```
Cypress.config()
Cypress.config(name)
Cypress.config(name, value)
Cypress.config(object)
```

Test Configuration vs Cypress.config()

Test Configuration

A test configuration object can be passed to the test or suite method.

Cypress.config()

While Cypress.config() modifies configuration values across the spec file, test configuration modifies configuration values just within the suite or test when they are set.

Cypress.config vs Cy.config

Generally, anything users call from Cypress has an impact on the global state. Anything users call from cy has an impact on the local state.

Given that the configuration added or changed by Cypress.config is only applicable to the current spec file, one may think it should be cy.config rather than Cypress.config.

However, the fact that Cypress.config has an effect on local state is a byproduct of the API's evolution: Cypress.config used to effect global state—configuration added in one test spec file was available in other specs.

In 3.0.0, the Cypress team smartly made each spec run in isolation, and Cypress.config was a public API by that time.

FULL STACK

Cypress Configuration

simplilearn

Default Application-specific Configurations by Cypress

Cypress includes several default configurations for a variety of application features:

Option	Default	Description
baseUrl	null	Specifies the URL that can be used as a prefix for the cy.visit() or cy.request() commands.
env .	0	Defines the values that will be changed.
numTestsKeptInMemory	50	Defines the values that will be changed.

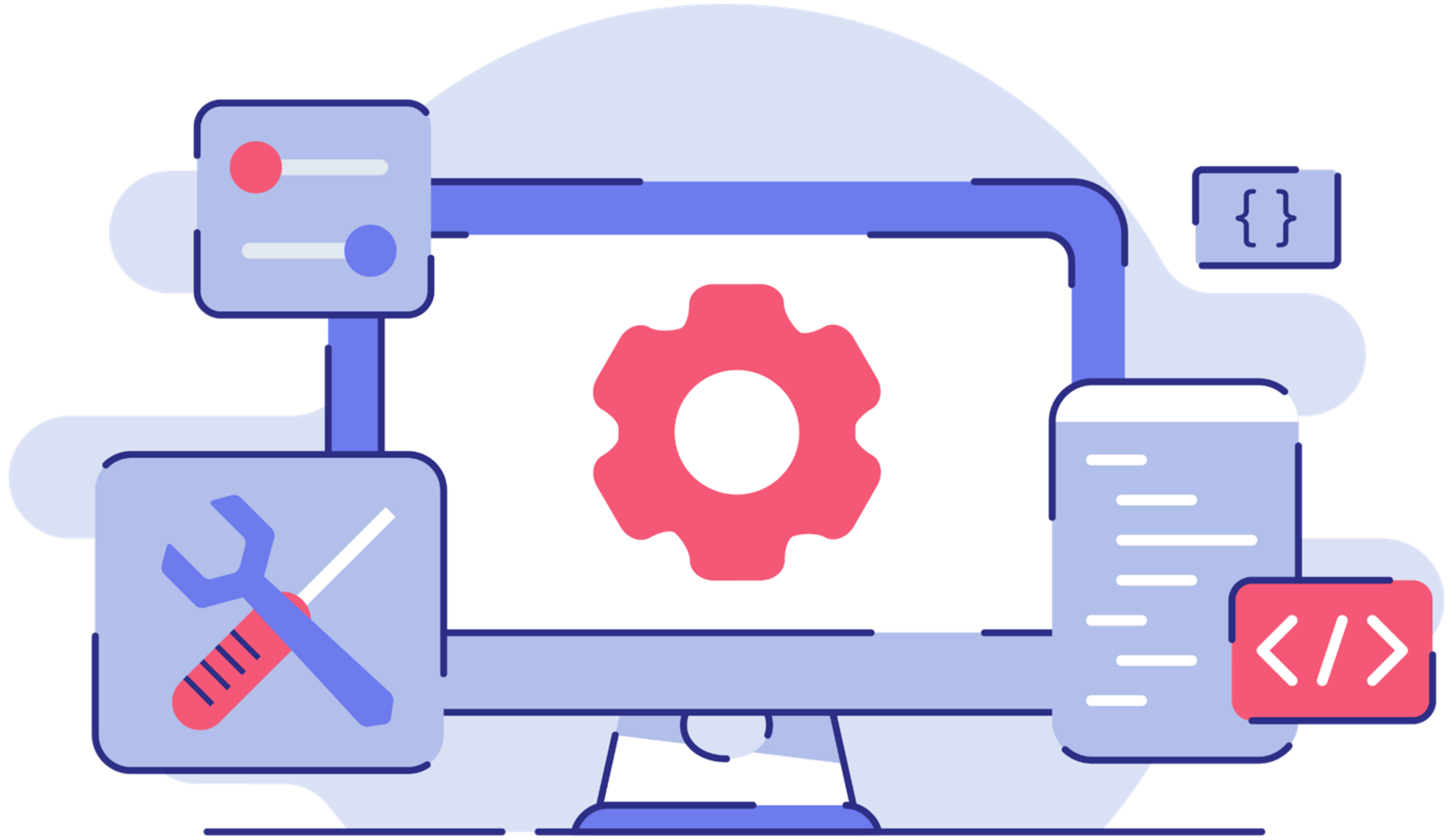
Default Application-specific Configurations by Cypress

Cypress includes several default configurations for a variety of application features:

Option	Default	Description
port	null	Specifies how many tests we store the command data and snapshots in memory for usage.
reporter	spec	Provides the port information needed to host Cypress.
reporterOptions	Null	Describes the reporter choices, which are settings specific to the selected reporter.
watchForFileChanges	TRUE	Determines whether Cypress will keep an eye on and restart tests if the test files are changed during execution.

Defining Configurations in Cypress

In Cypress, configurations define a set of key values that may be utilized across the test framework and will influence the framework's behavior depending on their default or changed values.



CONFIGURATION

Default Configurations Timeouts by Cypress

Option	Default	Description
defaultCommandTimeout	4000	It specifies the maximum time in milliseconds. Any DOM-based commands(such as cy.get(), etc.) will timeout.
execTimeout	60000	The maximum time is specified in milliseconds. Furthermore, the cy.exec() command will wait for its execution to complete.
taskTimeout	60000	This configuration specifies the maximum time in milliseconds. Moreover, the cy.task() command will wait to finish its execution

Default Configurations Timeouts by Cypress

Option	Default	Description
pageLoadTimeout	60000	The maximum time in milliseconds to wait for page transition events or cy.visit() or cy.reload() commands to fire their page load events is specified in this configuration.
requestTimeout	5000	In a cy.wait() command, this configuration specifies the maximum time in milliseconds to wait for an XHR request to be sent out.
responseTimeout	30000	This configuration specifies the maximum amount of time in milliseconds to wait for a response in commands such as cy.request(), cy.wait(), cy.fixture(), cy.getCookie(), cy.setCookie(), and cy.clearCookie()().

Cypress Configuration-Component Option

Option	Default	Description
devServer	null	Option required for configuring the component testing devServer.
setupNodeEvents	null	Function for registering node events and changing configuration. Replaces the (now-defunct) plugins file.
supportFile	cypress/support/component.js	Provides a path to the file to load before the spec files.

Global Options

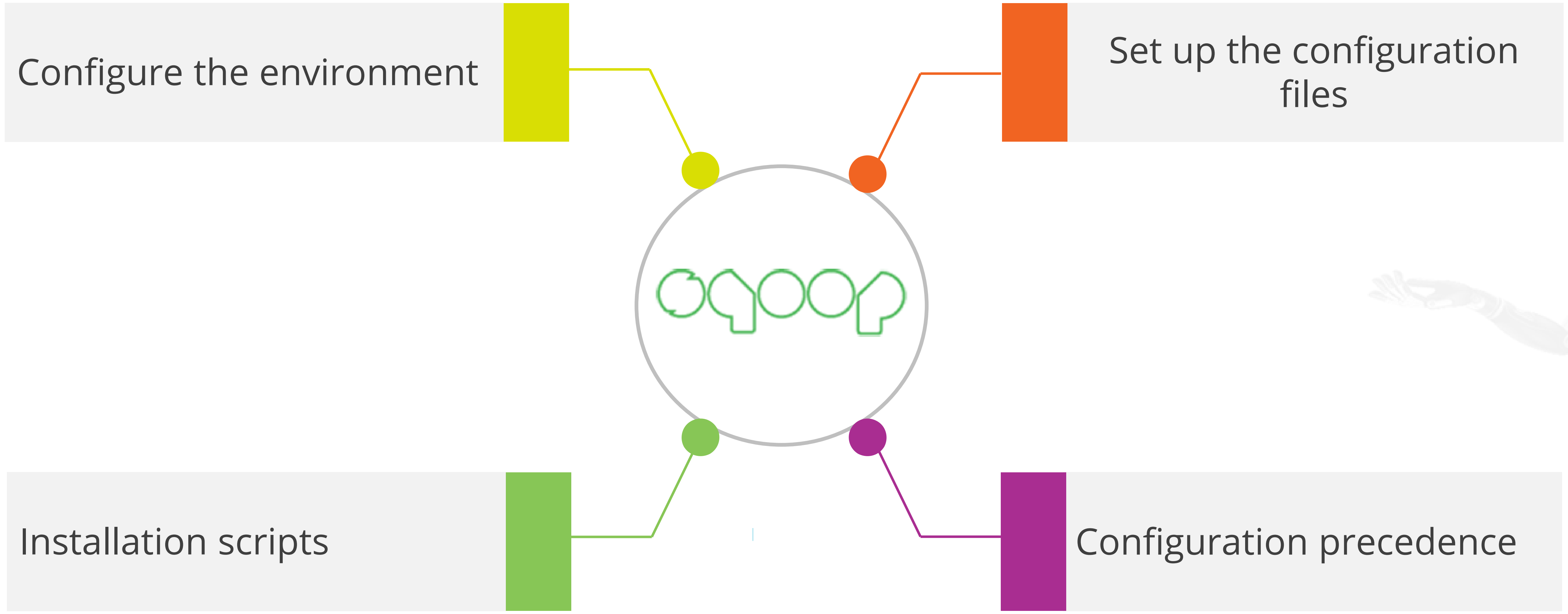
Option	Default	Description
baseURL	null	It may be used with the commands cy.visit() and cy.request()
clientCertificates	[ ]	This option is useful for setting client certificates on a per-URL basis
env	[ ]	This feature will come in handy if we are testing our application in several contexts such as staging or production

Global Options

Option	Default	Description
port	null	On hosting Cypress, we may pass the port number. A random port is produced, however we may specify the port number.
numTestsKeptInMemory	50	This parameter determines the number of test snapshots and command data retained in memory.
retries	{ "runMode": 0, "openMode": 0 }	This option allows you to select the number of times a failing test should be retried.

Configuring Cypress in Multiple Environments

Users can use a few simple procedures to customize Cypress for diverse situations:



Cypress Open

Users may now launch Cypress from the project root in one of the following ways:

Making use of npx

Note: npx is provided with npm > v5.2 and may also be installed individually.

Cypress Open

Users may now launch Cypress from the project root in one of the following ways:

```
yarn run cypress open
```

Note: npx is provided with npm > v5.2 and may also be installed individually.

Adding Npm Scripts

While there's nothing wrong with typing out the complete path to the Cypress executable each time, adding Cypress commands to the scripts field in the package.json file is considerably easier and clearer.

Command:

```
{
"scripts": {
"cypress:open": "cypress open"
}
}
```

Note: npx is provided with npm > v5.2 and may also be installed individually.

Tooling: IDE Integration

Cypress offers options to choose where one wants to open the file:

- The file system
- An IDE that is installed on your system
- A particular application route

File Opener Preference

Tooling: IDE Integration

Visual Studio Code

Tooling: IDE Integration

IntelliJ Platform

Cypress Support

Cypress Support Pro

Tooling: IDE Integration

Cypress may now use IntelliSense. While writing tests, it provides intelligent code suggestions right in the IDE.

Tooling: IDE Integration

Plugins

- Plugins allow users to interact with the Node process that is operating outside the browser.
- Plugins provide a "seam" for you to write a custom code that executes at specific points of the Cypress lifecycle.

Tooling: IDE Integration

Plugins: Use cases

Configuration

Users may use this to perform things like:

- Use several environments, each with its own set of settings.
- Change the environment variables based on the environment.
- The built-in fs lib is used to read configuration files.
- Change the browsers used for testing.
- Construct the configuration in yml.

Tooling: IDE Integration

Plugins: Use cases

Preprocessors

- Include the most recent ES* support.
- ClojureScript is the language to be used for the test code.
- Modify the Babel parameters to include any owned plugin.
- Customize the TypeScript compilation parameters.
- Replace webpack with Browserify or anything else.

Tooling: IDE Integration

Plugins: Use cases

Run Lifecycle

We can use the following:

Use case for before shows tasks such as configuration and reporting on a run and set a timer for the run to determine how long it will take.

Use case for after shows tasks such as finishing and reporting run along with stopping the timer.

Tooling: IDE Integration

Plugins: Use cases

Spec Lifecycle

Before:spec can be used to do things like:

- Set up reporting on a running specification
- Set a timer for the specification to see how long it takes

After:spec can be used to achieve things like:

- Complete the reporting setup in before:spec
- Stop the timer for the spec set up previously:spec
- Delete the video that was made for the specification

Tooling: IDE Integration

Plugins: Use cases

Browser Launching

Users may use the before:browser:launch event to accomplish the following:

- Install a Chrome extension
- Turn on or off experimental Chrome features
- Users can choose which Chrome components are loaded

Tooling: IDE Integration

Plugins: Use cases

Screenshot handling

Users can use the after:screenshot event to accomplish the following:

- Save screenshot information
- Rename the image
- Resize or crop the screenshot image to the liking

Tooling: IDE Integration

Plugins: Use cases

cy.task

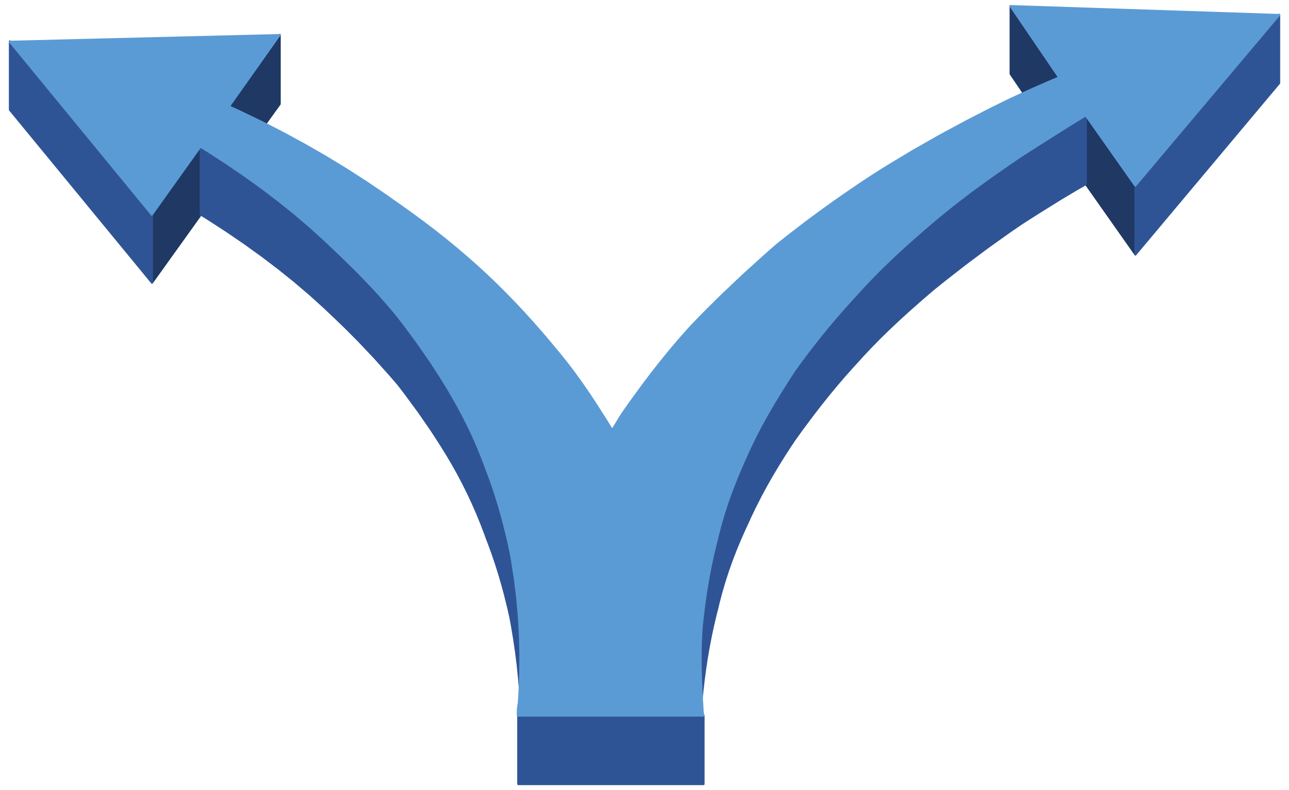
Users can use the task event to accomplish the following:

- Database manipulation
- Storing different states in Nodes that users wish to be persistent with
- Performing concurrent tasks
- External process execution

Tooling: IDE Integration

In the Cypress configuration, users can add the plugin to the setupNodeEvents function.

Users can add the plugin to the plugins file if they are using an older version of Cypress.

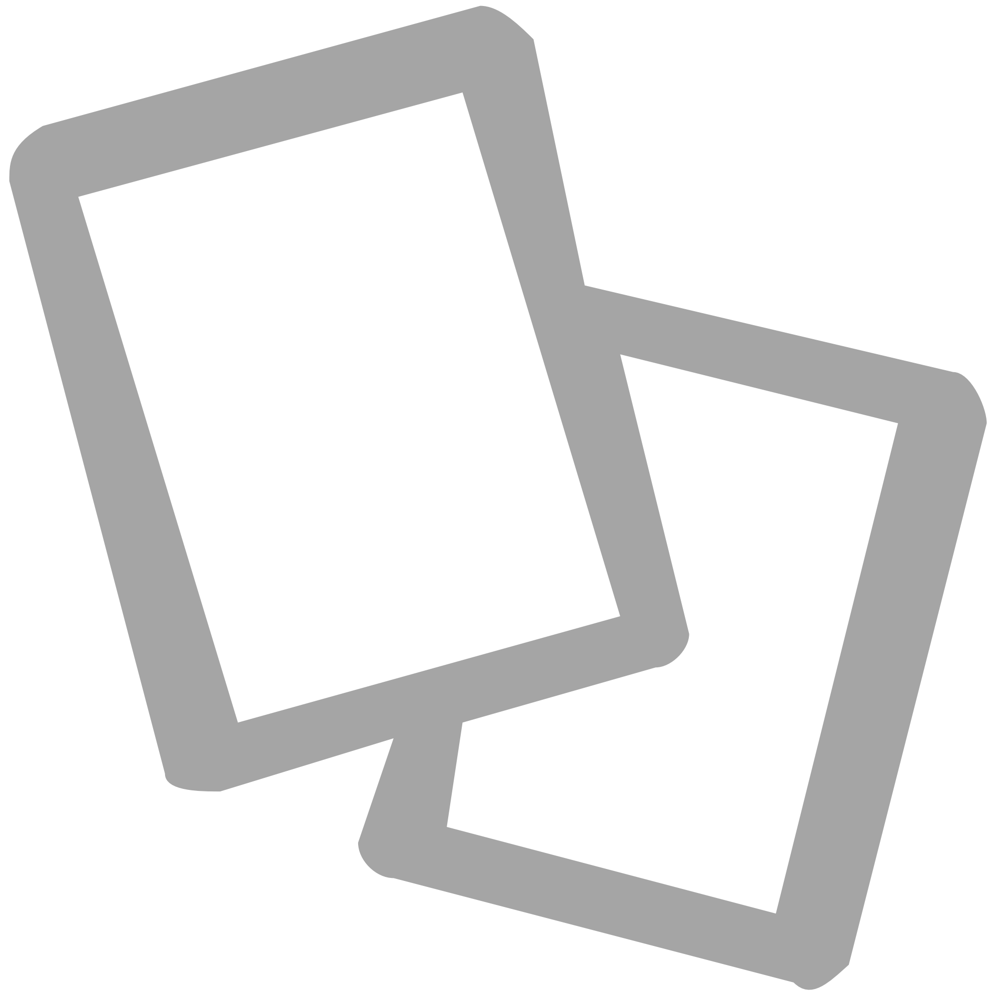


How to Use a Plugin?

E2E Testing Option

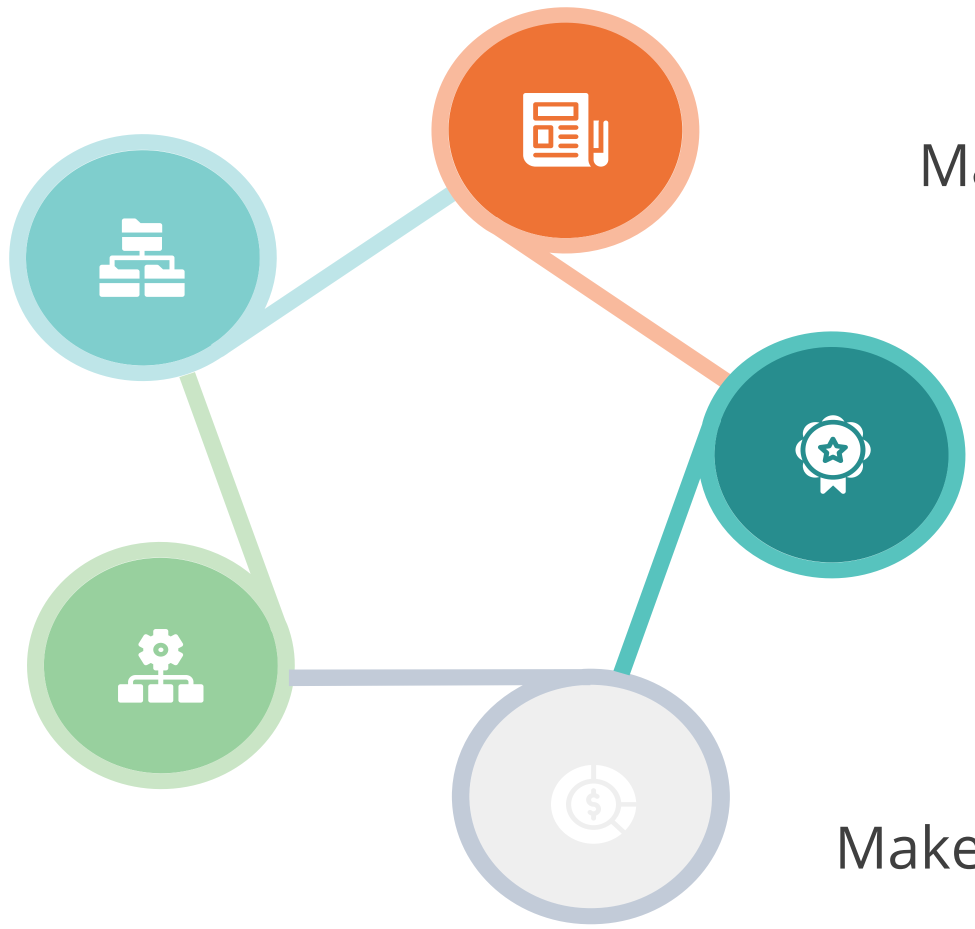
Define E2E Testing

E2E testing allows to test the app in the same way the end user will test.



Testing Application Using Cypress E2E

Cypress is very popular to test an application because of the following reasons:



Has a nice test runner

Makes debugging easy

Waits for instructions, assertions, and API calls automatically

Extends using custom commands or a plugin

Makes use of Chai and Mocha

Writing Cypress For End-to-end Tests

Users will simulate doing the action the user is anticipated to perform throughout the test in order to figure out how to create an E2E test for that scenario, and they will then assert that the ensuing application state conforms to the expectations.

Test One: A user performs a search from the homepage.

Test Two: A user switches languages from the homepage.

Component Option

Cypress Component Testing offers a testable component workbench that allows users to easily build and test any component, no matter how simple or complex it is.

Create E2E Configuration

Problem Statement:

You are required to create E2E configuration.

Assisted Practice: Guidelines

Steps to create E2E configuration are:

1. Create E2E configuration

Key Takeaways

- Cypress gives users the ability to dynamically change environment variables and configuration parameters from the Cypress setup.
- Each specification file is run separately by Cypress; the browser is closed in between specifications.
- End-to-end testing (E2E) concentrates on what ultimately matters: ensuring that users will be able to accomplish their goals within the application.
- E2E testing files must end in spec.js to be presented correctly.