**CYPRESS**

**Cypress is based on JavaScript:**

1. Unit tests.
2. End to end tests.
3. Integration tests.

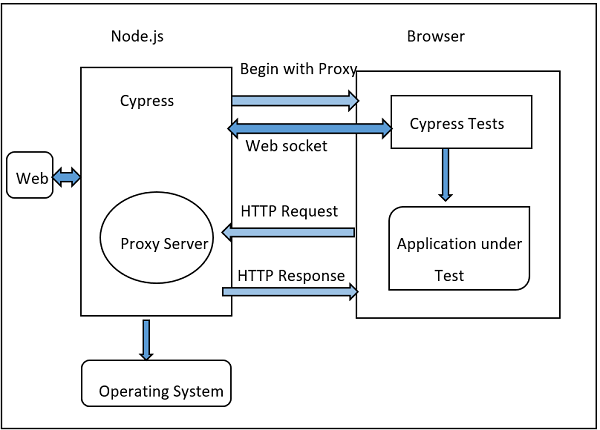
**Features:**

1. Supports Test-Driven development.
2. Provides Dashboard services.
3. Efficient debugging with Developer Tools accompanied with generation of stack trace and errors.
4. Provides the screenshots for failed tests.
5. Not necessary to add waits to stop the execution for some time. By-default, the waits are applied, prior to executing the following step or assertion.
6. Able to monitor and control the characteristics of server response, functions, and timers, which are essentially needed for unit testing.
7. Check and manage network traffic.
8. Allows the multi-browser support.
9. In-built feature to capture videos of execution is available.
10. Can be integrated with continuous integration tools.
11. Page responsiveness with viewport sizing.
12. Reloads changes applied to tests by default.
13. Friendly Application Programming Interfaces (APIs) are available.
14. Test runner available, which allows the test execution straight from the User Interface (UI).

**Disadvantages:**

1. It is only based on JavaScript.
2. A relatively new tool and hence, the community support is not extensive.
3. It cannot perform mobile testing.
4. Shadow Document Object Model (DOM) cannot be accessed.
5. Tabs/child windows are managed by workarounds.

**Architecture:**



.

**Cypress VS Selenium:**

|  |  |  |
| --- | --- | --- |
|  | **Cypress** | **Selenium** |
| **Languages Supported** | Supports JavaScript Only | Supports all popular languages like Java, Python, Ruby, C#, PHP, etc. |
| **Browsers Supported** | Chrome, Edge, Firefox, Electron | Chrome, IE, Safari, Edge, Firefox, Opera |
| **Frameworks Supported** | Supports only Mocha JS | Supports multiple frameworks based on specific programming languages. (For example: JUnit for Java, Cucumber for JavaScript, etc.) |
| **Setup Complexity** | The setup is simple. No dependencies or additional downloads required | Setup is a bit challenging as it requires downloading browser-specific drivers and setting up the test environment |
| **Documentation & Community Support** | Very intuitive documentation along with a rapidly growing community | Well-established documentation and firm community support from users across the globe |

**Installation:**

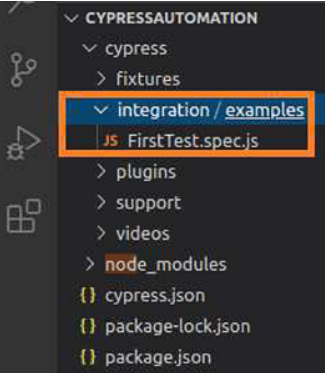
1. npm init
2. npm install cypress --save-dev

**Cypress Test Runner:**

1. Open Cypress by running − node\_modules/.bin/cypress open
2. node\_modules/.bin/cypress open

**Cypress Build First Test:**

**Cypress Folder Structure:**

****

* **fixtures** − Test data in form of key-value pairs for the tests are maintained here.
* **integration** − Test cases for the framework are maintained here.
* **plugins** − Cypress events (prior and post events to be executed for a test) are maintained here.
* **support** − Reusable methods or customized commands, which can be utilised by test cases directly, without object creation are created here.
* **videos** − Executed test steps are recorded in the form of videos and maintained here.
* **node\_modules** − Project dependencies from the npm are maintained in this folder.It is the heart of the Cypress project execution.
* **cypress.json** − Default configurations are set in this folder. The values of the current configurations can be modified here, which overrules the default configurations.
* **package.json** − Dependencies and scripts for the projects are maintained in this folder.

**Basic Test**

// test suite name

describe('Tutorialspoint Test', function () {

// Test case

it('Scenario 1', function (){

// test step for URL launching

cy.visit("https://www.google.com/");

});

});

**Test Execution:**

1. From the command line:

./node\_modules/.bin/cypress run

1. From the Test Runner:

./node\_modules/.bin/cypress open

1. For a specific file from command line:   
   cypress run --spec "<spec file path>"

**Supported Browsers:**

Cypress supports: Chrome, Electron, and Firefox

**Execution from other Browsers:**

./node\_modules/.bin/cypress run -- browser chrome

**Execution in headed mode:**

**./node\_modules/.bin/cypress run – headed**

**Cypress Basic Commands:**

* and

used to create an assertion and is an alias of .should ()

//element is visible & enabled

cy.get('#txt').should('be.visible').and('be.enabled')

//element is checked

cy.contains('Subject').and('be.checked')

* as

provides an alias for later usage

//alias element as parent

cy.get('#txt').find('li').first().as('parent')

* blur

blurs an element in focus.

//blur input

cy.get('#txt'). type('abc').blur()

* check

checks radio buttons or checkboxes and is applied to elements having input tags.

//checks element having class attribute chkbox

cy.get('.chkbox').check()

* children

obtains the sub elements of an element.

//obtains children of element n

cy.get('n').children()

* clear

removes the value from textarea or input.

//removes input abc

cy.get('#txt'). type('abc').clear()

* clearCookie

removes a particular browser cookie.

//clear abc cookie

cy.clearCookie('abc')

* clearCookies

removes the browser cookies from an existing domain and subdomain.

//clear all cookies

cy.clearCookies()

* clearLocalStorage

removes the local Storage data from an existing domain and subdomain.

//clear all local storage

cy. clearLocalStorage ()

* click

clicks an element in Document Object Model (DOM).

//click on element with id txt

cy.get('#txt').click()

* contains

obtains an element having a specific text. The element can have more than the text and still match.

//returns element in #txt having Tutor text

cy.get('#txt').contains('Tutor')

* dblclick

double-clicks an element in Document Object Model (DOM).

//double clicks element with id txt

cy.get('#txt').dblclick()

* debug

fixes a debugger and log values are returned by prior command.

//pause to debug at start of command

cy.get('#txt').debug()

* document

obtains window.document on the active page.

cy.document()

* each

iterates through an array having the property length.

//iterate through individual li

cy.get('li').each(() => {...})

* end

ends a command chain.

//obtain null instead of input

cy.contains('input').end()

* eq

 refers to an element at a particular index in an array of elements.

//obtain third td in tr

cy.get('tr>td').eq(2)

* exec
* find
* first
* get
* getCookie
* getCookies
* go
* visit
* next
* parent
* should
* wait
* title
* viewport
* log
* reload

**Commands:**

Cypress commands can be classified one of the following

* query - commands that read the state of your application
* assertion - command which assert on a given state
* action - interact with your application as a user would
* other - any other command that is useful for writing tests

The differences between queries and other commands are outline in the [Retry-ability Guide](https://docs.cypress.io/app/core-concepts/retry-ability).

**Queries**[**​**](https://docs.cypress.io/api/table-of-contents#Queries)

Queries are Cypress commands that read the state of your application. They return a subject for further commands to act or assert on, and retry as needed to make sure the DOM element or other data they yield is always up to date.

| **Command** | **Usage** |
| --- | --- |
| [.as()](https://docs.cypress.io/api/commands/as) | Assign an alias for later use. Reference the alias later within a cy.get() query or cy.wait() command. |
| [.children()](https://docs.cypress.io/api/commands/children) | Get the children of each element within a set of DOM elements. |
| [.closest()](https://docs.cypress.io/api/commands/closest) | Get the first ancestral element that matches a selector. |
| [.contains()](https://docs.cypress.io/api/commands/contains) | Select a DOM element by text content. |
| [.document()](https://docs.cypress.io/api/commands/document) | Get the window.document of the active page. |
| [.eq()](https://docs.cypress.io/api/commands/eq) | Select a DOM element by index from a collection. |
| [.filter()](https://docs.cypress.io/api/commands/filter) | Filter elements with a selector. |
| [.find()](https://docs.cypress.io/api/commands/find) | Find descendent elements with a selector. |
| [.first()](https://docs.cypress.io/api/commands/first) | Select the first item in a collection. |
| [.focused()](https://docs.cypress.io/api/commands/focused) | Get the DOM element that is currently focused. |
| [.get()](https://docs.cypress.io/api/commands/get) | Find DOM elements by selector, or read an alias previously created with the .as() command. |
| [.hash()](https://docs.cypress.io/api/commands/hash) | Get the URL hash of the active page. |
| [.invoke()](https://docs.cypress.io/api/commands/invoke) | Invoke a function on the previously yielded subject. |
| [.its()](https://docs.cypress.io/api/commands/its) | Get a property's value on the previously yielded subject. |
| [.last()](https://docs.cypress.io/api/commands/last) | Select the last item in a collection. |
| [.location()](https://docs.cypress.io/api/commands/location) | Get the window.location object of the active page. |
| [.next()](https://docs.cypress.io/api/commands/next) | Get the next sibling element. |
| [.nextAll()](https://docs.cypress.io/api/commands/nextall) | Get all following sibling elements. |
| [.nextUntil()](https://docs.cypress.io/api/commands/nextuntil) | Get all following sibling elements until reaching a selector. |
| [.not()](https://docs.cypress.io/api/commands/not) | Filter selected elements with a selector. |
| [.parent()](https://docs.cypress.io/api/commands/parent) | Get the parent of a DOM element. |
| [.parents()](https://docs.cypress.io/api/commands/parents) | Get all parent elements of a DOM element. |
| [.parentsUntil()](https://docs.cypress.io/api/commands/parentsuntil) | Get all parent elements up to a selector. |
| [.prev()](https://docs.cypress.io/api/commands/prev) | Get the previous sibling element. |
| [.prevAll()](https://docs.cypress.io/api/commands/prevall) | Get all previous sibling elements. |
| [.prevUntil()](https://docs.cypress.io/api/commands/prevuntil) | Get all following previous sibling elements until reaching a selector. |
| [.root()](https://docs.cypress.io/api/commands/root) | Get the root DOM element. |
| [.shadow()](https://docs.cypress.io/api/commands/shadow) | Traverse into the shadow DOM of an element. |
| [.siblings()](https://docs.cypress.io/api/commands/siblings) | Get all sibling elements. |
| [.title()](https://docs.cypress.io/api/commands/title) | Get the document.title property of the active page. |
| [.url()](https://docs.cypress.io/api/commands/url) | Get the URL of the active page. |
| [.window()](https://docs.cypress.io/api/commands/window) | Get the window object of the active page. |

**Assertions**[**​**](https://docs.cypress.io/api/table-of-contents#Assertions)

Assertions are Cypress commands that assert on the state of your application. They pause your test until the given condition is met, or until they time out. Learn more about Cypress assertions in the [Introduction to Assertions](https://docs.cypress.io/app/core-concepts/introduction-to-cypress#Assertions) and the [Assertions Reference](https://docs.cypress.io/app/references/assertions).

| **Command** | **Usage** |
| --- | --- |
| [.and()](https://docs.cypress.io/api/commands/and) | Alias for .should(). Assert on the state of your application. |
| [.should()](https://docs.cypress.io/api/commands/should) | Assert on the state of your application. Assertions are automatically retried until they pass or time out. |

**Actions**[**​**](https://docs.cypress.io/api/table-of-contents#Actions)

Actions are Cypress commands that interact with your application as a user would. They wait for elements to be actionable before interacting with the page.

Learn more about action commands in our [Actionability Guide](https://docs.cypress.io/app/core-concepts/interacting-with-elements).

| **Command** | **Usage** |
| --- | --- |
| [.check()](https://docs.cypress.io/api/commands/check) | Check checkbox or radio elements. |
| [.clear()](https://docs.cypress.io/api/commands/clear) | Clear the value of an input or textarea. |
| [.click()](https://docs.cypress.io/api/commands/click) | Click a DOM element. |
| [.dblclick()](https://docs.cypress.io/api/commands/dblclick) | Double-click a DOM element. |
| [.rightclick()](https://docs.cypress.io/api/commands/rightclick) | Right click a DOM element. |
| [.scrollIntoView()](https://docs.cypress.io/api/commands/scrollintoview) | Scroll an element into view. |
| [.scrollTo()](https://docs.cypress.io/api/commands/scrollto) | Scroll to a specific position. |
| [.select()](https://docs.cypress.io/api/commands/select) | Select an <option> within a <select>. |
| [.selectFile()](https://docs.cypress.io/api/commands/selectfile) | Selects a file in an HTML5 input element, or simulates dragging a file into the browser. |
| [.trigger()](https://docs.cypress.io/api/commands/trigger) | Trigger an event on a DOM element. |
| [.type()](https://docs.cypress.io/api/commands/type) | Type into a DOM element. |
| [.uncheck()](https://docs.cypress.io/api/commands/uncheck) | Uncheck checkbox(es). |

**Other Commands**[**​**](https://docs.cypress.io/api/table-of-contents#Other-Commands)

Cypress has a variety of additional commands to help write tests.

| **Command** | **Usage** |
| --- | --- |
| [.blur()](https://docs.cypress.io/api/commands/blur) | Blur a focused element. Blur is not an action command, because users cannot directly unfocus an element, only focus a new one, but .blur() can be used as a testing shortcut. |
| [.clearAllCookies()](https://docs.cypress.io/api/commands/clearallcookies) | Clear all browser cookies. |
| [.clearAllLocalStorage()](https://docs.cypress.io/api/commands/clearalllocalstorage) | Clear localStorage data for all origins with which the test has interacted. |
| [.clearAllSessionStorage()](https://docs.cypress.io/api/commands/clearallsessionstorage) | Clear sessionStorage data for all origins with which the test has interacted. |
| [.clearCookie()](https://docs.cypress.io/api/commands/clearcookie) | Clear a specific browser cookie. |
| [.clearCookies()](https://docs.cypress.io/api/commands/clearcookies) | Clear browser cookies for a domain. |
| [.clearLocalStorage()](https://docs.cypress.io/api/commands/clearlocalstorage) | Clear data in localStorage for current domain and subdomain. |
| [.clock()](https://docs.cypress.io/api/commands/clock) | Override the browser's built-in time objects (Date, setTimeout and similar), allowing them to be controlled via cy.tick(). |
| [.debug()](https://docs.cypress.io/api/commands/debug) | Set a debugger statement and log what the previous command yields. |
| [.each()](https://docs.cypress.io/api/commands/each) | Invoke a callback on each item in an array. |
| [.end()](https://docs.cypress.io/api/commands/end) | Explicitly end a chain of Cypress commands. |
| [.exec()](https://docs.cypress.io/api/commands/exec) | Execute a system command. |
| [.fixture()](https://docs.cypress.io/api/commands/fixture) | Load a fixed set of data from disk. |
| [.focus()](https://docs.cypress.io/api/commands/focus) | Focus on a DOM element. Focus is not an action command, because users cannot directly focus an element, but .focus() can be used as a testing shortcut. |
| [.getAllCookies()](https://docs.cypress.io/api/commands/getallcookies) | Get all browser cookies. |
| [.getAllLocalStorage()](https://docs.cypress.io/api/commands/getalllocalstorage) | Get localStorage data for all origins with which the test has interacted. |
| [.getAllSessionStorage()](https://docs.cypress.io/api/commands/getallsessionstorage) | Get sessionStorage data for all origins with which the test has interacted. |
| [.getCookie()](https://docs.cypress.io/api/commands/getcookie) | Get a browser cookie by its name. |
| [.getCookies()](https://docs.cypress.io/api/commands/getcookies) | Get browser cookies for the current domain or a specified domain. |
| [.go()](https://docs.cypress.io/api/commands/go) | Navigate back or forward using the browser's history. |
| [.hover()](https://docs.cypress.io/api/commands/hover) | Cypress does not have a cy.hover() command. See [Issue #10](https://github.com/cypress-io/cypress/issues/10). |
| [.intercept()](https://docs.cypress.io/api/commands/intercept) | Spy and stub network requests and responses. |
| [.log()](https://docs.cypress.io/api/commands/log) | Print a message to the Cypress Command Log. |
| [.mount()](https://docs.cypress.io/api/commands/mount) | Mount a component for Cypress Component Testing. |
| [.origin()](https://docs.cypress.io/api/commands/origin) | Visit multiple domains of different origin in a single test. |
| [.pause()](https://docs.cypress.io/api/commands/pause) | Pause test execution, allowing interaction with the application under test before resuming. |
| [.readFile()](https://docs.cypress.io/api/commands/readfile) | Read a file from disk. |
| [.reload()](https://docs.cypress.io/api/commands/reload) | Reload the page. |
| [.request()](https://docs.cypress.io/api/commands/request) | Make an HTTP request. |
| [.screenshot()](https://docs.cypress.io/api/commands/screenshot) | Take a screenshot of the application under test. |
| [.session()](https://docs.cypress.io/api/commands/session) | Cache and restore cookies, localStorage, and sessionStorage (i.e. session data) in order to recreate a consistent browser context between tests. |
| [.setCookie()](https://docs.cypress.io/api/commands/setcookie) | Set a browser cookie. |
| [.spread()](https://docs.cypress.io/api/commands/spread) | Invoke a callback function with multiple arguments. Similar to .then(). |
| [.spy()](https://docs.cypress.io/api/commands/spy) | Wrap a method in a spy in order to record calls to the function. |
| [.stub()](https://docs.cypress.io/api/commands/stub) | Replace a method, record its usage and control its behavior. |
| [.submit()](https://docs.cypress.io/api/commands/submit) | Submit a form. |
| [.task()](https://docs.cypress.io/api/commands/task) | Execute code in Node via the task plugin event. |
| [.then()](https://docs.cypress.io/api/commands/then) | Invoke a callback function with the current subject, allowing you to execute code at specific points during a Cypress test. |
| [.tick()](https://docs.cypress.io/api/commands/tick) | Move time after overriding a native time functions with cy.clock(). |
| [.viewport()](https://docs.cypress.io/api/commands/viewport) | Control the size and orientation of the screen for your application. |
| [.visit()](https://docs.cypress.io/api/commands/visit) | Visit a remote URL. Many tests begin with this command. |
| [.wait()](https://docs.cypress.io/api/commands/wait) | Wait a number of milliseconds, or wait for an aliased resource to resolve. |
| [.within()](https://docs.cypress.io/api/commands/within) | Scopes all subsequent cy commands to within this element. Useful when working within a particular group of elements such as a <form>. |
| [.wrap()](https://docs.cypress.io/api/commands/wrap) | Yield the object passed into .wrap(). If the object is a promise, yield its resolved value. |
| [.writeFile()](https://docs.cypress.io/api/commands/writefile) | Write to a file with the specified contents. |

**Cypress API**[**​**](https://docs.cypress.io/api/table-of-contents#Cypress-API)

**Events**[**​**](https://docs.cypress.io/api/table-of-contents#Events)

Cypress emits a series of events as it runs in your browser. These events are useful not only to control your application's behavior, but also for debugging purposes. Learn more about them in the [Catalog of Events](https://docs.cypress.io/api/cypress-api/catalog-of-events).

**Custom Commands and Queries**[**​**](https://docs.cypress.io/api/table-of-contents#Custom-Commands-and-Queries)

Cypress exposes interfaces to write [Custom Commands](https://docs.cypress.io/api/cypress-api/custom-commands) and [Custom Queries](https://docs.cypress.io/api/cypress-api/custom-queries). See those pages for more details.

The Cypress community has created a large number of

[Command Plugins](https://docs.cypress.io/app/plugins/plugins-list#custom-commands) which you can install or download.

**APIs**[**​**](https://docs.cypress.io/api/table-of-contents#APIs)

Cypress exposes a number of properties and methods on the global Cypress object.

The *key* difference between Cypress APIs and Cypress commands is that Cypress APIs execute the moment they are invoked and are **not** enqueued to run at a later time.

| **Property** | **Usage** |
| --- | --- |
| [Cypress.arch](https://docs.cypress.io/api/cypress-api/arch) | CPU architecture name of the underlying OS, as returned from Node's os.arch(). |
| [Cypress.browser](https://docs.cypress.io/api/cypress-api/browser) | Information about the current browser, such as browser family and version. |
| [Cypress.Commands](https://docs.cypress.io/api/cypress-api/custom-commands) | Create new custom commands and extend or override existing ones. |
| [Cypress.config()](https://docs.cypress.io/api/cypress-api/config) | Get and set Cypress configuration from inside your tests. |
| [Cypress.Cookies.debug()](https://docs.cypress.io/api/cypress-api/cookies) | Generate console logs whenever a cookie is modified. |
| [Cypress.currentRetry](https://docs.cypress.io/api/cypress-api/currentretry) | A number representing the current test retry count. |
| [Cypress.currentTest](https://docs.cypress.io/api/cypress-api/currenttest) | An object with information about the currently executing test. |
| [Cypress.log](https://docs.cypress.io/api/cypress-api/cypress-log) | This is the internal API for controlling what gets printed to the Command Log. Useful when writing your own custom commands. |
| [Cypress.dom](https://docs.cypress.io/api/cypress-api/dom) | A collection of DOM related helper methods. |
| [Cypress.env](https://docs.cypress.io/api/cypress-api/env) | Get environment variables from inside your tests. |
| [Cypress.isBrowser()](https://docs.cypress.io/api/cypress-api/isbrowser) | Checks if the current browser matches the given name or filter. |
| [Cypress.isCy()](https://docs.cypress.io/api/cypress-api/iscy) | checks if a variable is a valid instance of cy or a cy chainable. |
| [Cypress.Keyboard.defaults()](https://docs.cypress.io/api/cypress-api/keyboard-api) | Set default values for how the .type() command is executed. |
| [Cypress.platform](https://docs.cypress.io/api/cypress-api/platform) | The underlaying OS name, as returned by Node's os.platform(). |
| [Cypress.require](https://docs.cypress.io/api/cypress-api/require) | Enables utilizing dependencies within the [cy.origin()](https://docs.cypress.io/api/commands/origin) callback function. |
| [Cypress.Screenshot.defaults()](https://docs.cypress.io/api/cypress-api/screenshot-api) | Set defaults for screenshots captured by the .screenshot() command and the automatic screenshots taken during test failures. |
| [Cypress.SelectorPlayground](https://docs.cypress.io/api/cypress-api/selector-playground-api) | Configure options used by the [Selector Playground](https://docs.cypress.io/app/core-concepts/open-mode#Selector-Playground). |
| [Cypress.session](https://docs.cypress.io/api/cypress-api/session) | A collection of helper methods related to the .session() command. |
| [Cypress.spec](https://docs.cypress.io/api/cypress-api/spec) | An object with information about the currently executing spec file. |
| [Cypress.testingType](https://docs.cypress.io/api/cypress-api/testing-type) | The current testing type, eg. "e2e" or `"component". |
| [Cypress.version](https://docs.cypress.io/api/cypress-api/version) | The current Cypress version. |

**Utilities**[**​**](https://docs.cypress.io/api/table-of-contents#Utilities)

Cypress makes several common libraries available directly on the global Cypress object.

| **Utility** | **Usage** |
| --- | --- |
| [Cypress.$](https://docs.cypress.io/api/utilities/$) | Cypress automatically includes [jQuery](http://jquery.com/) and exposes it as Cypress.$. |
| [Cypress.Blob](https://docs.cypress.io/api/utilities/blob) | Cypress automatically includes a [Blob](https://github.com/nolanlawson/blob-util) library and exposes it as Cypress.Blob. |
| [Cypress.Buffer](https://docs.cypress.io/api/utilities/buffer) | Cypress automatically includes a [Buffer polyfill](https://github.com/feross/buffer) for the browser and exposes it as Cypress.Buffer. |
| [Cypress.\_](https://docs.cypress.io/api/utilities/_) | Cypress automatically includes [lodash](https://lodash.com/" \t "_blank) and exposes it as Cypress.\_. |
| [Cypress.minimatch](https://docs.cypress.io/api/utilities/minimatch) | Cypress automatically includes [minimatch](https://github.com/isaacs/minimatch" \t "_blank) and exposes it as Cypress.minimatch. |
| [Cypress.Promise](https://docs.cypress.io/api/utilities/promise) | Cypress automatically includes Bluebird and exposes it as Cypress.Promise. |
| [Cypress.sinon](https://docs.cypress.io/api/utilities/sinon) | Cypress automatically includes [Sinon.JS](http://sinonjs.org/) and exposes it as Cypress.sinon. |

**Node Events**[**​**](https://docs.cypress.io/api/table-of-contents#Node-Events)

The Node Events allow you to hook into and extend Cypress behavior. For more details, see the [How to use a Plugin](https://docs.cypress.io/app/plugins/plugins-guide) and [Node Events Overview](https://docs.cypress.io/api/node-events/overview).

| **API** | **Usage** |
| --- | --- |
| [After Run Event](https://docs.cypress.io/api/node-events/after-run-api) | The after:run event fires after a run is finished. When running cypress via cypress open, the event will fire when closing a project. |
| [After Screenshot Event](https://docs.cypress.io/api/node-events/after-screenshot-api) | After a screenshot is taken, you can get details about it via the after:screenshot plugin event. |
| [After Spec Event](https://docs.cypress.io/api/node-events/after-spec-api) | The after:spec event fires after a spec file is run. When running cypress via cypress open, the event will fire when the browser closes. |
| [Before Run Event](https://docs.cypress.io/api/node-events/before-run-api) | The before:run event fires before a run starts. When running cypress via cypress open, the event will fire when opening a project. |
| [Before Spec Event](https://docs.cypress.io/api/node-events/before-spec-api) | The before:spec event fires before a spec file is run. When running cypress via cypress open, the event will fire when the browser launches. |
| [Browser Launch API](https://docs.cypress.io/api/node-events/browser-launch-api) | Before Cypress launches a browser, it gives you the opportunity to modify the browser preferences, install extensions, add and remove command-line arguments, and modify other options from the [setupNodeEvents](https://docs.cypress.io/app/plugins/plugins-guide" \l "Using-a-plugin) function. |
| [Configuration API](https://docs.cypress.io/api/node-events/configuration-api) | Cypress enables you to dynamically modify configuration values and environment variables from your Cypress configuration. |
| [Preprocessors API](https://docs.cypress.io/api/node-events/preprocessors-api) | A preprocessor is the plugin responsible for preparing a support file or a test file for the browser. A preprocessor also typically watches the source files for changes, processes them again, and then notifies Cypress to re-run the tests. |

**Cypress API custom commands:** [**https://docs.cypress.io/api/cypress-api/custom-commands**](https://docs.cypress.io/api/cypress-api/custom-commands)

**Cypress Query:**

Syntax:

Cypress.Commands.addQuery(name, callbackFn)  
Cypress.Commands.overwriteQuery(name, callbackFn)

Eg:

Cypress.Commands.addQuery('getById', *function* (id) {  
 *return* (subject) => newSubject  
})  
  
Cypress.Commands.overwriteQuery('get', *function* (originalFn, ...args) {  
 *return* originalFn.apply(*this*, args)  
})

**Cypress cookies: (**[**https://docs.cypress.io/api/cypress-api/cookies**](https://docs.cypress.io/api/cypress-api/cookies)**)**

Cypress.Cookies.debug(enable, options)

**Cypress API Testing**

**Syntax**

***cy.request(method,URL,body)***

In the above command, method, and body are optional but the URL is mandatory.

|  |  |
| --- | --- |
| **Argument** | **Definition** |
| **method** | This is the type of HTTP method we would like to implement. By default, Cypress makes this request as GET. |
| **URL** | URL to which we need to send the request |
| **body** | Body to send along with the request |

***cy.request(options)***

In the above command, the argument options should be in the form of object

|  |  |  |
| --- | --- | --- |
| **Option** | **Default value** | **Description** |
| log | *true* | Whether it should display the command in command log |
| url | *null* | This is the URL to which we need to make request |
| method | *GET* | Type of HTTP request which we need to request |
| auth | *null* | Authorization header which needs to send along with the request |
| body | *null* | Body of the request |
| failOnStatusCode | *true* | Whether to make the script fail for response code other than 2xx and 3xx |
| followRedirect | *true* | Whether to follow redirects automatically |
| form | *false* | Whether to convert the body values to URL encoded content and set the URL encoded header |
| encoding | *utf8* | This is the type of encoding to be used when serializing the response body. Below are the supported encodings:  ***ascii, base64, binary, hex, latin1, utf8, utf-8, ucs2, ucs-2, utf16le, utf-16le***. |
| gzip | *true* | Whether to accept the gzip encoding |
| headers | *null* | Additional header which needs to be sent. |
| qs | *null* | Query paraments to append to the URL of the request. |
| retryOnStatusCodeFailure | *false* | This set the retry ability to Cypress request command. If it is set to true, Cypress will give a retry up to 4 times. |
| retryOnNetworkFailure | *true* | Retry ability on network error. This will give a retry up to 4 times. |
| timeout | *responseTimeout* | Time up to which it has to wait for the completion of the command. By default, it will wait up to the time which is mentioned in the config file responseTimeout |

**Making HTTP Request with Cypress**

Let’s make a simple GET request with cy.request() command. We will make a GET request to the following URL – <https://www.bstackdemo.com/> with the below piece of code,

describe('template spec', () => {

it('passes', () => {

//Getting response from BrowserStack demo website

cy.request('GET','https://www.bstackdemo.com/')

})

})

**Methods:**

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| GET | This method is for getting the list of data / specific data, available from the API server |
| POST | POST request is used to create data in the server |
| PUT | PUT is for updating the existing data |
| PATCH | This method is for updating the data partially |
| DELETE | As the name implies, it is for deleting the data in target server |

**GET => 200**

describe('template spec', () => {

it('passes', () => {

//Getting response from browserstack demo website

cy.request('GET','https://www.bstackdemo.com/').then((response) =>{

//Expecting the response status code to be 200

expect(response.status).to.eq(200)

})

})

})

**With the below piece of code, you can execute POST, PUT, PATCH and DELETE request and understand it more clearly**

describe('template spec', () => {

//This block inserts the data which is mentioned in the body of the request

it('POST request', () => {

cy.request({method: 'POST', url: 'https://reqres.in/api/users', body: {

"name": "Bingo",

"job": "Team lead"

}}).then((response) =>{

//Asserting the status code to be 201 for successful execution

expect(response.status).to.eq(201)

//Asserting the name which we have inserted into

expect(response.body.name).to.eq("Bingo")

//Asserting the status text to confirm whether it is created

expect(response.statusText).to.eq("Created")

})

})

//This block will execute PUT request

it('PUT request', () => {

cy.request({method: 'PUT', url: 'https://reqres.in/api/users/2', body: {

"name": "Angel",

"job": "zion resident"

}}).then((response) =>{

//Asserting the status code to be 200 for successful execution

expect(response.status).to.eq(200)

//Asserting the name which we have inserted into

expect(response.body.name).to.eq("Angel")

})

})

//This block will execute the PATCH request and update the record

it('PATCH request', () => {

cy.request({method: 'PATCH', url: 'https://reqres.in/api/users/2', body: {

"name": "Angel",

"job": "zion resident"

}}).then((response) =>{

//Asserting the status code to be 200 for successful execution

expect(response.status).to.eq(200)

//Asserting the name which we have inserted into

expect(response.body.name).to.eq("Angel")

})

})

//This block will delete the user which exist

it('DELETE request', () => {

cy.request({method: 'DELETE', url: 'https://reqres.in/api/users/2'}).then((response) =>{

//Asserting the status code to be 204 for successful execution

expect(response.status).to.eq(204)

})

})

})

***cypress.config.js***

env: {

myURL: "https://reqres.in/"

}

Create a custom command inside /support/commands.js

//We are creating a custom command named GETrequest which has URL and userDataLink as argument

Cypress.Commands.add('GETrequest', (url,userDataLink) =>{

//Usual GET request command

cy.request('GET',`${Cypress.env(url)+userDataLink}`)

})

Ex. of working code:

describe('template spec', () => {

it('passes', () => {

//Get the API request with URL and user details arguments and store response in a variable named details

cy.GETrequest('myURL',"api/users/2").as('details')

//With cy.get() validating the status code of response

cy.get('@details').its('status').should('eq',200)

})

})

Reference: <https://www.browserstack.com/guide/cypress-api-testing>