

Testing Your Application with Cypress



Marko Vajs

Software Development Engineer in Test

Module Overview



Set up and explore the demo application

Write our first Cypress test

Examine an interactive test runner

Explore Cypress project organization

Learn in detail how to interact with elements on a web page

Demo



Introducing Vue demo application

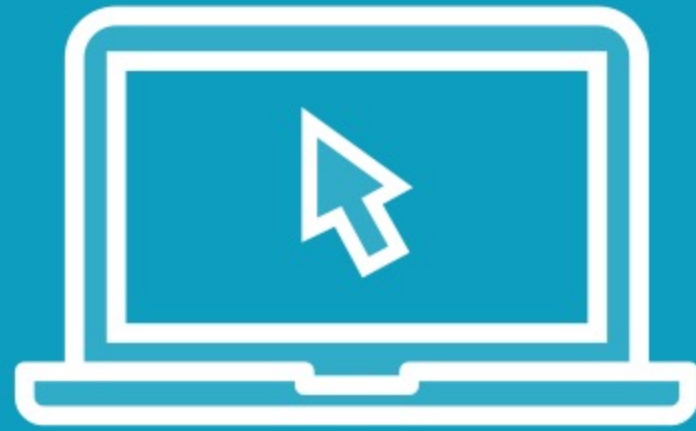
Demo



Running demo application locally

Planning Our First Test

Demo



Create a new JavaScript file for our test

Define a test suite

Write a test

Demo



Implementing our first test






Demo



Using the interactive Test Runner

Organizing Tests

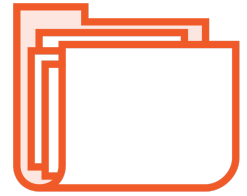
Organizing Tests

- ✓  cypress
 - >  fixtures
 - >  integration
 - >  plugins
 - >  support

Cypress creates four main folders:

- **Fixtures**
- **Integration**
- **Plugins**
- **Support**

Organizing Tests



Fixtures

Static files that are used in tests such as JSON files and images



Integration

Tests



Plugins

Modifications and extension of Cypress's internal behavior



Support

Reusable pieces of code such as custom commands

Example of a Custom Command

```
Cypress.Commands.add('clickLink', (label) => {  
  cy.get('a').contains(label).click()  
});
```

```
cy.clickLink('Buy Now');
```

Cypress generates some temporary folders during and after test execution

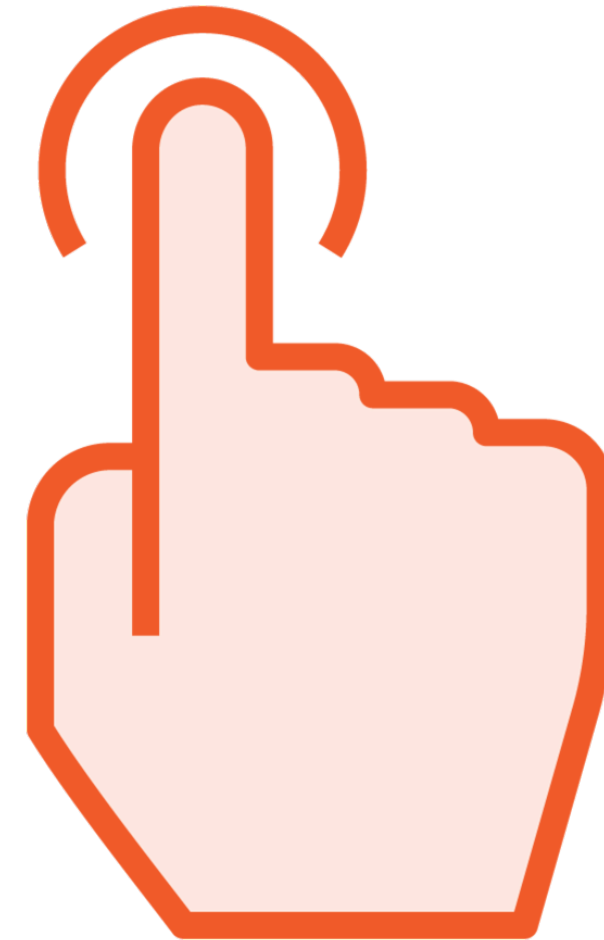
```
.gitignore  
  report/  
  screenshots/  
  videos/
```

Interacting with DOM Elements

Interacting with DOM Elements



**Uniquely identify an
element**



**Perform an action on that
element**

How to Select Element

```
<input name="event-name" id="event-name" class="form-input" data-cy="event-name" type="text">
```

`cy.get('input')`



`cy.get(' .form-input')`



`cy.get(' #event-name')`



`cy.get('input[name="event-name"]')`



`cy.get('input[data-cy="event-name"]')`



<h3 data-v-4749a695="">Test Event</h3>

Commands for Selecting Elements

`cy.get()`

Get one or more elements by selector

`cy.contains()`

Get elements containing the text

`cy.get().find()`

Get the descendent elements of a specific selector

Commands for Simulating User Actions

click

dblclick

type

clear

focus

blur

check

select

Simulating User Actions

click

click simulates a click action on an element.

```
.click()
```

```
.click({ force: true })
```

```
.click(5, 10)
```

```
.click('topLeft')
```

Simulating User Actions

dblclick

dblclick simulates a double click action on an element.

```
.dblclick()
```

```
.dblclick({ force: true })
```

```
.dblclick(5, 10)
```

```
.dblclick('topLeft')
```

Simulating User Actions

type

click simulates a click action on an element.

```
.type( 'text' )
```

```
.type( 'text', { delay: 10 } )
```

```
.type( '{backspace}' )
```

Simulating User Actions

clear

clear clears the value of an input or textarea.

```
.clear()
```

```
.clear({ force: true })
```

Simulating User Actions

blur and **focus**

blur blurs a focused element and **focus** focuses an element.

```
.blur()
```

```
.focus()
```


Simulating User Actions

select

select selects an option of a select.

```
.select('value')
```

```
.select('val', { log: false })
```

```
.select(['val1', 'val2'])
```

```
.select(['v1', 'v2'], { log: false })
```

Simulating User Actions

check

check checks checkboxes or radios.

```
.check('value')
```

```
.check('val', { log: false })
```

```
.check(['val1', 'val2'])
```

```
.check(['v1', 'v2'], { log: false })
```

Demo



Interacting with DOM elements

Module Summary

Module Summary



Cypress tests are kept inside an integration folder and organized in suites

Each suite contains one or more tests

Cypress has an interactive test runner

Interacting with an element means locating the element and performing an action on it

The preferred way to construct a selector is by using a custom data attribute

The most commonly used actions are `click`, `dblclick`, `type`, `clear`, `focus`, `blur`, `check` and `select`

Up Next:
Exploring Features and Core Concepts
