Testing Angular Application

Marko Vajs
Software Development Engineer in Test

Module Overview

Explore the Angular application

Learn how to locate elements on a page

Explore interactive test runner

Learn how to interact with elements and write assertions

Demo

Introducing sample Angular application

Selecting DOM Elements

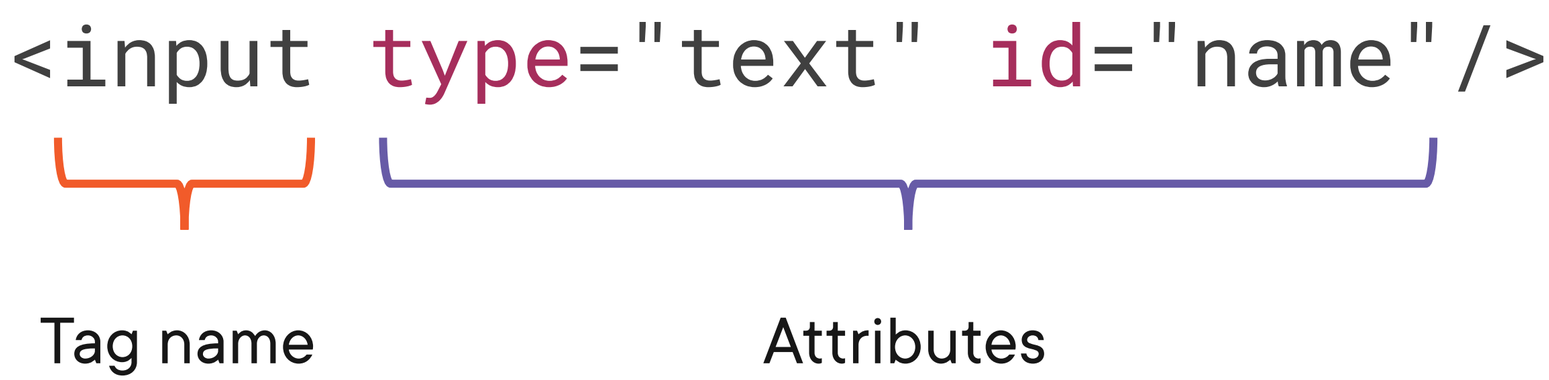
HTML Elements


```
<button type="submit">Submit</button>
```

```
<input type="text" id="name"/>
```

```
<input type="checkbox" class="mt-2"/>
```

...


<input type="text" id="name"/>
Tag name
Attributes

```
<div id="intro"></div>
```

```
<input id="name" type="text" class="intro"/>

<input id="email-address" type="email" class="intro"/>
```

Selecting Elements

.intro → **input[class="intro"]**

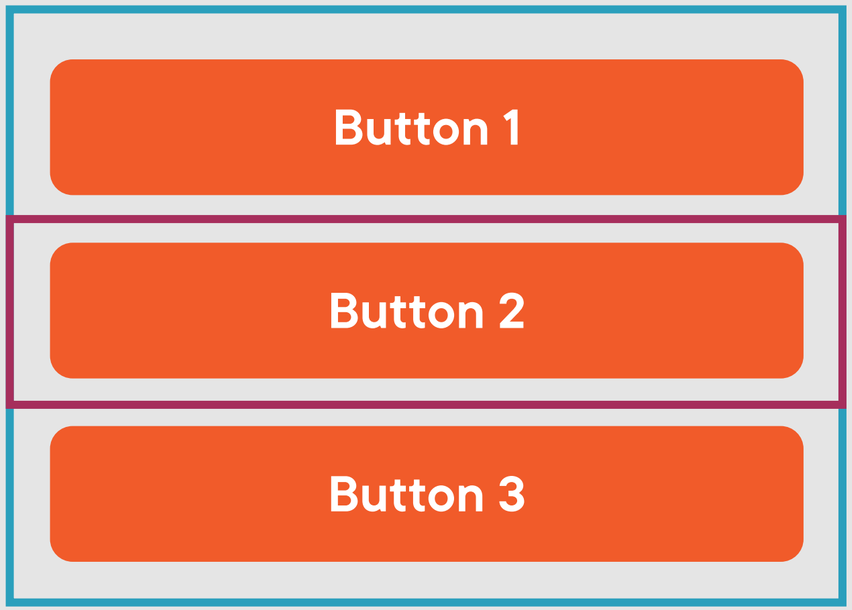
#intro → **div[id="intro"]**

A Good Selector

Unambiguous

Future-proof

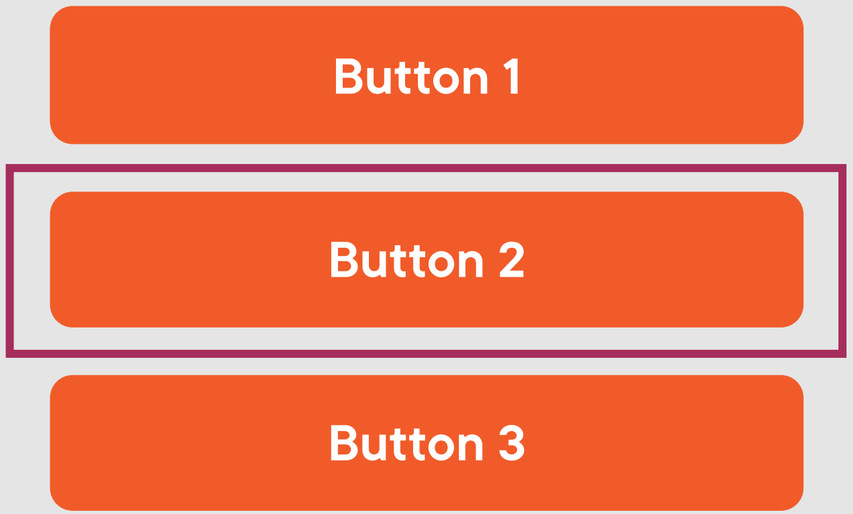
```
<div>
    <button>Button 1</button>
    <button>Button 2</button>
    <button>Button 3</button>
</div>
```




```
button
```

```
div > button:nth-child(2)
```

```
<div>
    <button>Button 1</button>
    <button>Button 2</button>
    <button>Button 3</button>
</div>
```




```
button
```

~~`div > button:nth-child(2)`~~

```
button#read-more
```

How to Select an Element

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

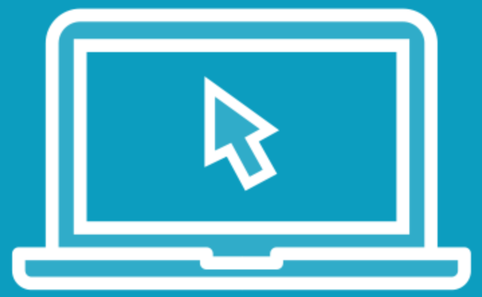
1. `cy.get('input')` 
2. `cy.get('.form-input')` 
3. `cy.get('#adventure')` 
4. `cy.get('input[name="adventure"]')` 
5. `cy.get('input[data-cy="adventure"]')` 

Data Attributes

data-* attributes allow us to store extra information on standard, semantic HTML elements.

https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

Demo

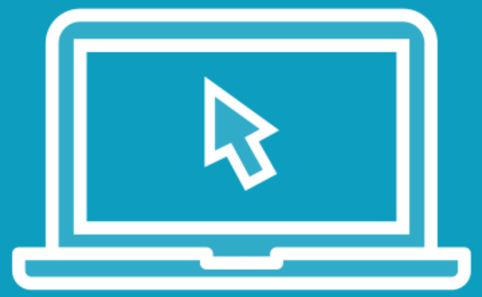


Creating a test suite

Demo

Implementing our first tests

Demo



Using interactive test runner

Interacting with DOM Elements

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`

`type` **enters text into an** `input` **or** `textarea`.

```
.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.**

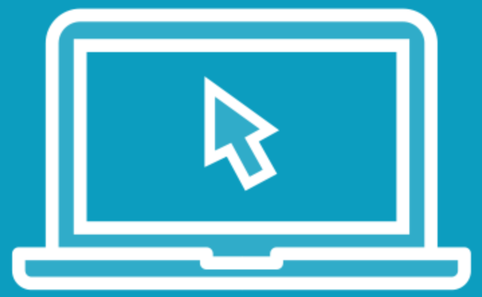
```
.check('value')

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

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

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

Demo



Interacting with DOM elements

Performing Assertions

Test Assertion

An assertion is a logical expression that evaluates as true or false depending on the test condition.

Assertions

length

```
cy.get('li.selected').should('have.length', 1)
```

class

```
cy.get('input#name').should('not.have.class', 'disabled')
```

value

```
cy.get('textarea#comment').should('have.value', 'Great!')
```

visibility

```
cy.get('button#add-comment').should('be.visible')
```

Assertions

existance

```
cy.get('span[name="loading"]').should('not.exist')
```

css

```
cy.get('.danger').should('have.css', 'color', '#FF0000')
```

Multiple Assertions

```
cy.get('a.more-details')
      .should('have.class', 'btn')
      .and('have.attr', 'href')
      .and('include', 'adventures')
```

~~cy.get('span[name="loading"]')~~
~~.should('exist')~~
~~.and('not.exist')~~

```
cy.get('span[name="loading"]').should('exist')
cy.get('span[name="loading"]').should('not.exist')
```

BDD and TDD Style

https://docs.cypress.io/guides/references/assertions

Module Summary

Module Summary

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

Each suite contains one or more tests

Selectors are usually a combination of HTML tags and attributes

A good selector is unambiguous and future-proof

Interactive test runner can help when writing tests

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

Up Next:
Understanding Core Concepts of Cypress