**Nguyễn Thái Quốc – 2154050249 – DH21IM02**

# Các lệnh cơ bản trong Cypress

## Lệnh .as()

* **Chức năng**: được sử dụng để gán một biệt danh (alias) cho một đối tượng DOM. Biệt danh này có thể được sử dụng lại trong các lệnh Cypress tiếp theo, giúp cho việc viết các test case trở nên ngắn gọn và dễ đọc hơn 🡺 **Tạo biệt danh cho các đối tượng để sử dụng lại.**
* **Ví dụ cách dùng**:
  + Giả sử bạn có một form đăng nhập với các trường username và password. Bạn có thể sử dụng .as() để gán biệt danh cho các trường này như sau:

*cy.get('#username')****.as('usernameField')***

*cy.get('#password')****.as('passwordField')***

*// Nhập thông tin đăng nhập*

*cy.get('****@usernameField****').type('your\_username')*

*cy.get('****@passwordField****').type('your\_password')*

*// Click nút đăng nhập*

*cy.get('button[type="submit"]').click()*

## Lệnh .children()

* **Chức năng**: được sử dụng để tìm kiếm và tương tác với các phần tử con trực tiếp của một phần tử cha đã được chọn 🡺 **Tìm kiếm các phần tử con**
* **Ví dụ cách dùng:** 
  + *cy.get('ul')****.children()*** *// Tìm tất cả các li bên trong ul*
  + *cy.get('div.parent')****.children('p')*** *// Tìm tất cả các p bên trong div có class là parent*

## Lệnh .closest()

* **Chức năng:** được sử dụng để tìm kiếm phần tử tổ tiên gần nhất (tức là phần tử cha hoặc ông bà) của một phần tử đã được chọn, mà khớp với một selector nhất định 🡺 **Tìm kiếm tổ tiên gần nhất**
* **Ví dụ cách dùng:** *cy.get('.product-item')****.closest('.product-list')***
  + .get('.product-item'): Tìm một phần tử có class là "product-item".
  + .closest('.product-list'): Tìm phần tử tổ tiên gần nhất của phần tử "product-item" có class là "product-list".

## Lệnh .contains()

* **Chức năng**: được sử dụng để tìm kiếm và tương tác với các phần tử trong DOM mà chứa một văn bản cụ thể 🡺 **Tìm kiếm phần tử chứa văn bản cụ thể**
* **Ví dụ cách dùng**:
  + *cy****.contains('Đăng nhập')*** *// Tìm button có chứa văn bản "Đăng nhập"*
  + *cy.get('button')****.contains('Submit')*** *// Tìm button có chứa văn bản "Submit"*

## Lệnh .document()

* **Chức năng**: cho phép truy cập trực tiếp vào đối tượng **document** của trang web đang được test. Điều này rất hữu ích khi muốn thực hiện các thao tác liên quan đến toàn bộ tài liệu HTML 🡺 **Truy cập vào đối tượng document**
* **Ví dụ cách dùng**:

*// Lấy tiêu đề của trang*

*cy****.document().****its('title').should('eq', 'Trang chủ')*

*// Kiểm tra URL hiện tại*

*cy****.document().****its('location.href').should('eq', 'https://example.com')*

*// Thêm một phần tử vào body*

*cy****.document().****then(($document) => {*

*const newDiv = $document.createElement('div')*

*newDiv.textContent = 'Hello, world!'*

*$document.body.appendChild(newDiv)*

*})*

## Lệnh .eq()

* **Chức năng**: cho phép lấy phần tử ở một vị trí nhất định trong một danh sách các phần tử
* **Ví dụ cách dùng**:

*cy.get('.product') // Tìm tất cả các phần tử có class là 'product'*

***.eq(2)*** *// Lấy phần tử thứ 3 (index bắt đầu từ 0)*

*.click() // Click vào phần tử thứ 3*

## Lệnh .filter()

* **Chức năng**: được sử dụng để lọc một tập hợp các phần tử và trả về một tập hợp con mới chỉ chứa các phần tử thỏa mãn một điều kiện nhất định 🡺 **Lọc các phần tử**
* **Ví dụ cách dùng**:

*cy.get('.product') // Tìm tất cả các phần tử có class là 'product'*

***.filter(':visible')*** *// Lọc ra các phần tử đang hiển thị*

*.should('have.length', 3) // Kiểm tra xem có 3 phần tử đang hiển thị*

## Lệnh .find()

* **Chức năng**: được sử dụng để tìm kiếm các phần tử con bên trong một phần tử cha đã được chọn
* **Ví dụ cách dùng**: *cy.get('.product-list')****.find('.product-item')***
  + .get('.product-list'): Tìm một phần tử có class là "product-list".
  + .find('.product-item'): Tìm tất cả các phần tử con có class là "product-item" bên trong phần tử "product-list".

## Lệnh .first()

* **Chức năng**: được sử dụng để chọn phần tử đầu tiên trong một tập hợp các phần tử đã được tìm thấy
* **Ví dụ cách dùng**: *cy.get('li')****.first().****click()*
  + Tìm tất cả các phần tử <li> trong DOM.
  + Chọn phần tử <li> đầu tiên.
  + Click vào phần tử <li> đó.

## Lệnh .focused()

* **Chức năng**: **Kiểm tra và tương tác với phần tử đang được focus**
  + **Kiểm tra xem một phần tử có đang được focus hay không**
  + **Tương tác trực tiếp với phần tử đang được focus:** Thực hiện các hành động như blur (mất focus), kiểm tra các thuộc tính, hoặc thực hiện các hành động khác trên phần tử đang được focus.
* **Ví dụ cách dùng**:

*// Kiểm tra xem input có ID là 'username' có đang được focus không*

*cy****.focused().****should('have.id', 'username')*

*// Blur phần tử đang được focus*

*cy****.focused().****blur()*

*// Thực hiện một hành động trên phần tử đang được focus*

*cy****.focused().****type('Hello, world!')*

## Lệnh .get()

* **Chức năng**: một trong những lệnh cốt lõi và được sử dụng phổ biến nhất trong Cypress. Nó đóng vai trò như một chiếc chìa khóa để có thể tìm kiếm và tương tác với các phần tử trên một trang web 🡺 **Cánh cửa vào thế giới kiểm thử**
* **Ví dụ cách dùng**:

*// Tìm một button có text là "Submit"*

*cy****.get('button:contains("Submit")')***

*// Tìm một input có ID là "username"*

*cy****.get('#username')***

*// Tìm tất cả các phần tử có class là "product"*

*cy****.get('.product')***

## Lệnh .hash()

* **Chức năng**: được sử dụng để tương tác với phần hash (phần nằm sau dấu #) trong URL của một trang web. Phần hash thường được sử dụng để thực hiện các hành động như điều hướng nội bộ trong trang, đánh dấu một vị trí cụ thể trong trang, hoặc truyền các tham số nhỏ 🡺 **Làm việc với phần hash của URL**
* **Ví dụ cách dùng**: Trong ví dụ dưới, khi click vào link "Đi đến phần 1", Cypress sẽ kiểm tra xem hash của URL có đúng là "#section1" không.

*//HTML*

*<a href="#section1">Đi đến phần 1</a>*

*<a href="#section2">Đi đến phần 2</a>*

*// Test case Cypress*

*cy.get('a[href="#section1"]').click()*

*cy****.hash()****.should('eq', '#section1')*

## Lệnh .invoke()

* **Chức năng**: cho phép gọi trực tiếp các phương thức (methods) hoặc truy cập các thuộc tính (properties) của một đối tượng DOM đã chọn 🡺 **Gọi trực tiếp các phương thức trên đối tượng**
* **Ví dụ cách dùng**:

*// Lấy giá trị của thuộc tính 'value' của một input*

*cy.get('#myInput')****.invoke('val')***

*// Gọi phương thức focus() để đặt focus vào một input*

*cy.get('#myInput')****.invoke('focus')***

*// Gọi phương thức scrollIntoView() để cuộn đến một phần tử*

*cy.get('#myElement')****.invoke('scrollIntoView')***

## Lệnh .its()

* **Chức năng**: được sử dụng để truy cập vào các thuộc tính bên trong của một đối tượng đã chọn trước đó. Nó cho phép đi sâu vào cấu trúc dữ liệu phức tạp để lấy ra thông tin chính xác 🡺 **Truy cập sâu vào cấu trúc dữ liệu**
* **Ví dụ cách dùng**:

*// Lấy giá trị của thuộc tính 'text' của phần tử thứ 2 trong một danh sách*

*cy.get('.product-list li').eq(1)****.its('text').****should('eq', 'Product B')*

## Lệnh .last()

* **Chức năng**: được sử dụng để chọn phần tử cuối cùng trong một tập hợp các phần tử đã được tìm thấy
* **Ví dụ cách dùng**: *cy.get('li')****.last().****click()*
  + Tìm tất cả các phần tử <li> trong DOM.
  + Chọn phần tử <li> cuối cùng.
  + Click vào phần tử <li> đó.

## Lệnh .location()

* **Chức năng**: được sử dụng để tương tác với đối tượng **location** của cửa sổ trình duyệt, cho phép truy xuất và thao tác các thông tin liên quan đến URL hiện tại của trang web 🡺 **Tương tác với URL của trang web**
* **Ví dụ cách dùng**:

*// Kiểm tra xem URL hiện tại có chứa "/products" không*

*cy****.location('pathname')****.should('include', '/products')*

*// Kiểm tra xem hash của URL có bằng "#about" không*

*cy****.location('hash')****.should('eq', '#about')*

*// Kiểm tra xem ứng dụng có chuyển hướng đến trang đăng nhập khi người dùng đăng xuất hay không*

*cy.visit('/')*

*cy.get('button[data-cy="logout"]').click()*

*cy****.location('pathname')****.should('eq', '/login')*

## Lệnh .next()

* **Chức năng**: tìm kiếm và trả về phần tử ngay sau phần tử đang chọn
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li>Item 2</li>*

*<li>Item 3</li>*

*</ul>*

*// Chọn phần tử "Item 2" và sau đó chọn phần tử tiếp theo (Item 3):*

*cy.get('li').eq(1)****.next()****.should('have.text', 'Item 3')*

## Lệnh .nextAll()

* **Chức năng**: tìm kiếm và trả về tất cả các phần tử cùng cấp và nằm sau phần tử đang chọn
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li>Item 2</li>*

*<li>Item 3</li>*

*<li>Item 4</li>*

*</ul>*

*// Chọn phần tử "Item 2" và lấy tất cả các phần tử anh em kế tiếp*

*cy.get('li').eq(1)****.nextAll()****.should('have.length', 2)*

## Lệnh .nextUntil()

* **Chức năng**: cho phép chọn tất cả các phần tử anh em kế tiếp của một phần tử đã chọn, nhưng chỉ đến khi gặp một phần tử khác được chỉ định 🡺 **Chọn các phần tử anh em kế tiếp cho đến một phần tử cụ thể**
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li>Item 2</li>*

*<li class="special">Item 3</li>*

*<li>Item 4</li>*

*<li>Item 5</li>*

*</ul>*

*// Chọn tất cả các phần tử <li> từ "Item 2" đến trước phần tử có class "special"*

*cy.get('li').eq(1)****.nextUntil('.special')****.should('have.length', 1)*

* + cy.get('li').eq(1): Chọn phần tử <li> thứ hai (Item 2).
  + .nextUntil('.special'): Chọn tất cả các phần tử <li> kế tiếp bắt đầu từ Item 2 cho đến trước phần tử có class "special".
  + .should('have.length', 1): Kiểm tra xem có đúng 1 phần tử được chọn (Item 3).

## Lệnh .not()

* **Chức năng**: được sử dụng để chọn tất cả các phần tử **không** khớp với điều kiện đã cho
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li class="item">Item 1</li>*

*<li class="item disabled">Item 2 (disabled)</li>*

*<li class="item">Item 3</li>*

*</ul>*

*// Chọn tất cả các phần tử <li> có class "item" nhưng không có class "disabled"*

*cy.get('.item')****.not('.disabled')***

## Lệnh .parent()

* **Chức năng**: được sử dụng để chọn phần tử cha trực tiếp của một phần tử đã chọn
* **Ví dụ cách dùng**:

*// HTML*

*<div class="container">*

*<p>Đây là một đoạn văn.</p>*

*</div>*

*// Chọn phần tử <p> và lấy phần tử cha của nó*

*cy.get('p')****.parent()****.should('have.class', 'container')*

* + cy.get('p'): Chọn phần tử <p>
  + .parent(): Tìm phần tử cha của <p>, tức là <div> có class "container"
  + .should('have.class', 'container'): Kiểm tra xem phần tử cha có class "container" không

## Lệnh .parents()

* **Chức năng**: được sử dụng để chọn **tất cả** các phần tử cha của một phần tử đã chọn, từ phần tử cha trực tiếp cho đến gốc của cây DOM 🡺 **Di chuyển lên nhiều cấp trong cây DOM**
* **Ví dụ cách dùng**:

*// HTML*

*<div class="container">*

*<div class="section">*

*<p>Đây là một đoạn văn.</p>*

*</div>*

*</div>*

*// Chọn phần tử <p> và lấy tất cả các phần tử cha của nó*

*cy.get('p')****.parents()****.should('have.length', 2)*

* + cy.get('p'): Chọn phần tử <p>
  + .parents(): Tìm tất cả các phần tử cha của <p>, tức là <div class="section"> và

<div class="container">

* + .should('have.length', 2): Kiểm tra xem có đúng 2 phần tử cha được tìm thấy không

## Lệnh .parentsUntil()

* **Chức năng**: cho phép chọn tất cả các phần tử cha của một phần tử đã chọn, nhưng chỉ đến khi gặp một phần tử khác được chỉ định 🡺 **Di chuyển lên các phần tử cha cho đến khi gặp một phần tử cụ thể**
* **Ví dụ cách dùng**:

*// HTML*

*<div class="container">*

*<div class="another-section">*

*<div class="section">*

*<p>Đây là một đoạn văn.</p>*

*</div>*

*</div>*

*</div>*

*// Chọn phần tử <p> và lấy tất cả các phần tử cha cho đến khi gặp phần tử có class*

*"another-section"*

*cy.get('p')****.parentsUntil('.another-section')****.should('have.length', 1)*

* + cy.get('p'): Chọn phần tử <p>
  + .parentsUntil('.another-section'): Tìm tất cả các phần tử cha của <p>, nhưng chỉ đến khi gặp phần tử có class "another-section". Trong trường hợp này, chỉ có

<div class="section"> được tìm thấy

* + .should('have.length', 1): Kiểm tra xem có đúng 1 phần tử cha được tìm thấy không

## Lệnh .prev()

* **Chức năng**: được sử dụng để tìm và tương tác với phần tử trực tiếp trước phần tử hiện tại trong DOM
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li class="active">Item 2</li>*

*<li>Item 3</li>*

*</ul>*

*cy.get('.active')****.prev()*** 🡺 Lệnh này sẽ chọn phần tử <li> có nội dung "Item 1"

## Lệnh .prevAll()

* **Chức năng**: được sử dụng để tìm và trả về **tất cả** các phần tử liền trước một phần tử đã chọn trong DOM
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li>Item 2</li>*

*<li class="active">Item 3</li>*

*<li>Item 4</li>*

*</ul>*

*cy.get('.active')****.prevAll()*** 🡺 Lệnh này sẽ trả về một danh sách chứa các phần tử <li> có nội dung "Item 1" và "Item 2".

## Lệnh .prevUntil()

* **Chức năng**: được sử dụng để tìm và trả về tất cả các phần tử liền trước một phần tử đã chọn, cho đến khi gặp một phần tử khác thoả mãn một điều kiện nhất định 🡺 **Tìm kiếm các phần tử trước đó cho đến khi gặp một phần tử cụ thể**
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li class="category">Category A</li>*

*<li>Item 2</li>*

*<li>Item 3</li>*

*<li class="category">Category B</li>*

*<li>Item 4</li>*

*</ul>*

*cy.get('.category').eq(1)****.prevUntil('.category')* 🡺** Chọn tất cả các phần tử <li> trước phần tử có class "category" thứ hai(Trả về “Item 2” và “Item 3”)

## Lệnh .root()

* **Chức năng**: được sử dụng để **trỏ về phần tử gốc** của DOM, thường là thẻ <html>
* **Ví dụ cách dùng**:

*cy.get('.modal').within(() => {*

*cy****.root()****.find('button.close').click() // Tìm nút đóng trong modal*

*})*

🡺 Vào trong một modal và sử dụng .root() để tìm nút đóng trong modal đó, mặc dù nút đóng có thể nằm ở một cấp độ cao hơn trong DOM so với modal.

## Lệnh .shadow()

* **Chức năng**: Shadow DOM là một tính năng của HTML cho phép các nhà phát triển tạo ra các phần của DOM được cách ly, giúp tạo ra các component tái sử dụng và phong cách hóa một cách độc lập. Điều này có nghĩa là các phần tử bên trong Shadow DOM không thể truy cập trực tiếp từ JavaScript thông thường 🡺 **Tương tác với Shadow DOM**
* **Ví dụ cách dùng**:

*// Giả sử có một component custom với Shadow DOM*

*<my-custom-element>*

*<div class="shadow-root">*

*<button class="my-button">Click me</button>*

*</div>*

*</my-custom-element>*

*cy.get('my-custom-element')****.shadow()****.find('.my-button').click()* 🡺 Click vào nút "Click me" bên trong Shadow DOM

## Lệnh .siblings()

* **Chức năng**: được sử dụng để tìm và trả về tất cả các phần tử có cùng cha với phần tử hiện tại
* **Ví dụ cách dùng**:

*// HTML*

*<ul>*

*<li>Item 1</li>*

*<li class="active">Item 2</li>*

*<li>Item 3</li>*

*</ul>*

*cy.get('.active')****.siblings()* 🡺** Chọn tất cả các anh em của phần tử có class "active". Lệnh này sẽ trả về một danh sách chứa các phần tử <li> có nội dung "Item 1" và "Item 3"

## Lệnh .title()

* **Chức năng**: được sử dụng để lấy tiêu đề của trang web hiện tại
* **Ví dụ cách dùng**:

*// Kiểm tra xem tiêu đề của trang có phải là "Trang chủ" không*

*cy****.title()****.should('eq', 'Trang chủ')*

## Lệnh .url()

* **Chức năng**: được sử dụng để lấy URL đầy đủ của trang web hiện tại mà Cypress đang tương tác
* **Ví dụ cách dùng**:

*// Kiểm tra xem Cypress đã điều hướng đến trang sản phẩm chưa*

*cy.visit('/products')*

*cy****.url()****.should('include', '/products')*

*// Kiểm tra xem URL có chứa tham số id=123 không*

*cy****.url()****.should('include', '?id=123')*

## Lệnh .window()

* **Chức năng**: cho phép truy cập vào đối tượng **window** của trình duyệt, tương tự như cách làm trong JavaScript thông thường. Đối tượng **window** đại diện cho toàn bộ cửa sổ trình duyệt và cung cấp quyền truy cập vào các đối tượng và phương thức toàn cục của trình duyệt 🡺 **Tương tác với đối tượng window**
* **Ví dụ cách dùng**:
  + Truy cập localStorage(Lưu trữ dữ liệu của người dùng một cách lâu dài, ngay cả khi đóng trình duyệt)

*cy****.window()****.then((win) => {*

*win.localStorage.setItem('myKey', 'myValue')*

*})*

* + Mở một cửa sổ mới

*cy.****window()****.then((win) => {*

*win.open('https://example.com', '\_blank')*

*})*

* + Kiểm tra một biến toàn cục

*cy****.window()****.its('myGlobalVariable').should('eq', 'someValue')*

## Lệnh .and()

* **Chức năng**: được sử dụng để kết nối nhiều khẳng định (assertions) hoặc hành động (actions) vào cùng một đối tượng (subject). Về cơ bản, .and() là một alias của .should() 🡺 **Liên kết các khẳng định và hành động**
* **Ví dụ cách dùng**:

*cy.get('button.submit')*

*.should('be.visible') // Khẳng định nút có thể nhìn thấy*

***.and('have.attr', 'disabled', 'false')*** *// Khẳng định nút không bị vô hiệu hóa*

***.and('contain', 'Submit')*** *// Khẳng định nút có chứa văn bản "Submit"*

*.click() // Click vào nút*

## Lệnh .should()

* **Chức năng**: thực hiện các kiểm tra và khẳng định trong quá trình viết test. Nó cho phép xác minh rằng các phần tử trên trang web đang ở đúng trạng thái như mong đợi 🡺 **Kiểm tra và khẳng định**
* **Ví dụ cách dùng**:

*cy.get('button.submit')*

***.should('be.visible')*** *// Khẳng định nút có thể nhìn thấy*

*.and('have.attr', 'disabled', 'false') // Khẳng định nút không bị vô hiệu hóa*

*.and('contain', 'Submit') // Khẳng định nút có chứa văn bản "Submit"*

*.click() // Click vào nút*

## Lệnh .check()

* **Chức năng**: được sử dụng để tương tác với các phần tử checkbox và radio button trên trang web. Khi gọi lệnh này trên một phần tử checkbox hoặc radio button, Cypress sẽ tự động đánh dấu (check) phần tử đó 🡺 **Đánh dấu các checkbox và radio button**
* **Ví dụ cách dùng**:

*// Đánh dấu một checkbox có id là "agreeTerms"*

*cy.get('#agreeTerms')****.check()***

*// Đánh dấu một radio button có giá trị là "option2"*

*cy.get('input[type="radio"][value="option2"]')****.check()***

*// Kiểm tra xem checkbox đã được đánh dấu chưa*

*cy.get('#rememberMe').should('be****.checked****')*

## Lệnh .clear()

* **Chức năng**: được sử dụng để xóa toàn bộ nội dung hiện có bên trong một ô nhập liệu (input) hoặc vùng văn bản (textarea)
* **Ví dụ cách dùng**:

*// Xóa nội dung của một ô nhập liệu có id là "username"*

*cy.get('#username')****.clear()***

*// Xóa nội dung của một textarea có class là "comment"*

*cy.get('.comment')****.clear()***

## Lệnh .click()

* **Chức năng**: mô phỏng hành động click chuột lên một phần tử cụ thể trên trang web
* **Ví dụ cách dùng**:

*// Click vào một nút có id là "submit"*

*cy.get('#submit')****.click()***

*// Click vào một liên kết có văn bản là "Đăng nhập"*

*cy.contains('Đăng nhập')****.click()***

## Lệnh .dblclick()

* **Chức năng**: mô phỏng hành động click đúp chuột vào một phần tử trên trang web
* **Ví dụ cách dùng**:

*// Click đúp vào một phần tử có id là "editable-text" để chuyển sang chế độ chỉnh sửa*

*cy.get('#editable-text')****.dblclick()***

## Lệnh .rightclick()

* **Chức năng**: mô phỏng hành động click chuột phải vào một phần tử trên trang web
* **Ví dụ cách dùng**:

*// Click chuột phải vào một phần tử có id là "context-menu"*

*cy.get('#context-menu')****.rightclick()***

## Lệnh .scrollIntoView()

* **Chức năng**: được sử dụng để đảm bảo rằng một phần tử cụ thể trên trang web nằm trong vùng hiển thị (viewport) của trình duyệt 🡺 **Đưa phần tử vào tầm nhìn**
* **Ví dụ cách dùng**:

*cy.get('#element-to-scroll')****.scrollIntoView****({*

*// Các tùy chọn:*

*ensureScrollable: true, // Đảm bảo phần tử có thể cuộn trước khi cuộn*

*easing: 'linear', // Thiết lập hàm easing cho hiệu ứng cuộn*

*duration: 500 // Thiết lập thời gian của hiệu ứng cuộn tính bằng mili giây*

*});*

## Lệnh .scrollTo()

* **Chức năng**: cho phép điều khiển việc cuộn của trang web một cách chính xác. Không giống như .scrollIntoView() chỉ đảm bảo phần tử hiển thị, .scrollTo() cho phép chỉ định tọa độ cụ thể hoặc hướng cuộn 🡺 **Cuộn đến vị trí cụ thể**
* **Ví dụ cách dùng**:

*// Cuộn đến vị trí có tọa độ X là 200 và Y là 300*

*cy.get('#element').****scrollTo(200, 300)***

*// Cuộn đến cuối của phần tử*

*cy.get('.scrollable-container').****scrollTo('bottom')***

*// Cuộn đến 50% chiều cao của phần tử*

*cy.get('.scrollable-container').****scrollTo('50%', 'center')***

*// Cuộn mượt mà đến cuối trang trong vòng 2 giây*

*cy.****scrollTo****('bottom', { duration: 2000, easing: 'linear' })*

## Lệnh .select()

* **Chức năng**: được sử dụng để tương tác với các phần tử danh sách thả xuống (select element) trên một trang web
* **Ví dụ cách dùng**:

*// Chọn tùy chọn đầu tiên trong danh sách*

*cy.get('#mySelect')****.select('first')***

*// Chọn tùy chọn cuối cùng trong danh sách*

*cy.get('#mySelect')****.select('last')***

*// Chọn một tùy chọn dựa trên chỉ số (bắt đầu từ 0)*

*cy.get('#mySelect')****.select(2)*** *// Chọn tùy chọn thứ 3*

*// Chọn nhiều tùy chọn (nếu select element cho phép chọn nhiều)*

*cy.get('#multiSelect')****.select(['option1', 'option3'])***

## Lệnh .selectFile()

* **Chức năng**: được sử dụng để mô phỏng hành động chọn một hoặc nhiều file trong một phần tử input dạng file (file input) của HTML5 🡺 **Tương tác với phần tử upload file**
* **Ví dụ cách dùng**:

*// Chọn nhiều file*

*cy.get('input[type="file"]').****selectFile(['file1.pdf', 'file2.docx'])***

*// Kéo thả file (simulating drag and drop)*

*cy.get('.dropzone').****selectFile('image.png', { action: 'drag-drop' })***

## Lệnh .trigger()

* **Chức năng**: được sử dụng để mô phỏng các sự kiện người dùng tương tác với các phần tử trên trang web. Cho phép kiểm tra cách ứng dụng phản ứng với các hành động như click, hover, keypress, ... 🡺 **Kích hoạt sự kiện trên phần tử**
* **Ví dụ cách dùng**:

*// Kích hoạt sự kiện click trên một button*

*cy.get('button').click() // Tương đương với cy.get('button')****.trigger('click')***

*// Kích hoạt sự kiện hover trên một div*

*cy.get('.tooltip')****.trigger('mouseover')***

*// Kích hoạt sự kiện change trên một input text khi nhập giá trị*

*cy.get('input[type="text"]').type('hello')****.trigger('change')***

*// Kích hoạt sự kiện keydown để nhập một chuỗi ký tự*

*cy.get('input[type="text"]')****.trigger('keydown', { keyCode: 65 })*** *// Nhập chữ 'a'*

*// Kích hoạt sự kiện click với tọa độ chuột cụ thể*

*cy.get('img')****.trigger('click', { clientX: 100, clientY: 50 })***

## Lệnh .type()

* **Chức năng**: được sử dụng để mô phỏng hành động nhập văn bản vào các phần tử input trên một trang web
* **Ví dụ cách dùng**:

*// Nhập một chuỗi ký tự đặc biệt*

*cy.get('#password'****).type('{selectall}Your new password')*** *// Chọn tất cả và nhập password mới*

*// Nhập một chuỗi ký tự với độ trễ*

*cy.get('#username')****.type('admin', { delay: 100 })*** *// Nhập từng ký tự với độ trễ 100ms*

*// Nhập một chuỗi ký tự đặc biệt (ví dụ: {enter}, {leftarrow}, {backspace})*

*cy.get('.search-input')****.type('{enter}')*** *// Nhấn Enter để thực hiện tìm kiếm*

## Lệnh .uncheck()

* **Chức năng**: được sử dụng để mô phỏng hành động bỏ chọn một checkbox trên trang web
* **Ví dụ cách dùng**:

*// Bỏ chọn tất cả các checkbox trong một form*

*cy.get('input[type="checkbox"]')****.uncheck()***

*// Bỏ chọn một checkbox và kiểm tra xem nó đã được bỏ chọn chưa*

*cy.get('#newsletter')****.uncheck()****.should('not.be.checked')*

## Lệnh .blur()

* **Chức năng**: được sử dụng để mô phỏng hành động khi di chuyển con trỏ chuột ra khỏi một phần tử trên trang web, làm cho phần tử đó mất tiêu điểm 🡺 **Mất tiêu điểm khỏi phần tử**
* **Ví dụ cách dùng**:

*// Làm mất tiêu điểm khỏi một input có id là "username"*

*cy.get('#username')****.blur()***

## Lệnh .clearAllCookies()

* **Chức năng**: được sử dụng để xóa tất cả các cookie(đóng vai trò như một "bộ nhớ" nhỏ, giúp website "nhớ" những thông tin của người dùng khi quay lại trang web đó) hiện có trong trình duyệt trước khi thực hiện một test case 🡺 **Xóa sạch tất cả cookie**
* **Ví dụ cách dùng**: Trong ví dụ dưới, trước mỗi test case trong khối describe('Đăng nhập'), Cypress sẽ xóa sạch tất cả cookie để đảm bảo rằng mỗi lần chạy test, người dùng đều ở trạng thái chưa đăng nhập.

*describe('Đăng nhập', () => {*

*beforeEach(() => {*

*cy****.clearAllCookies();***

*cy.visit('https://example.com');*

*});*

*it('Đăng nhập thành công', () => {*

*// Các bước đăng nhập*

*});*

*});*

## Lệnh .clearAllLocalStorage()

* **Chức năng**: được sử dụng để xóa toàn bộ dữ liệu được lưu trữ trong localStorage của trình duyệt. **LocalStorage** là một API của trình duyệt cho phép các trang web lưu trữ một lượng nhỏ dữ liệu của người dùng một cách vĩnh viễn 🡺 **Xóa sạch toàn bộ dữ liệu localStorage**
* **Ví dụ cách dùng**: Trong ví dụ dưới, trước mỗi test case, Cypress sẽ xóa toàn bộ dữ liệu trong localStorage để đảm bảo rằng người dùng luôn ở trạng thái chưa đăng nhập và không có dữ liệu nào được lưu trữ từ các lần chạy trước.

*describe('Đăng nhập', () => {*

*beforeEach(() => {*

*cy****.clearAllLocalStorage()***

*cy.visit('https://example.com');*

*});*

*it('Đăng nhập thành công', () => {*

*// Các bước đăng nhập*

*});*

*});*

## Lệnh .clearAllSessionStorage()

* **Chức năng**: được sử dụng để xóa toàn bộ dữ liệu được lưu trữ trong sessionStorage của trình duyệt. **SessionStorage** là một API của trình duyệt cho phép các trang web lưu trữ một lượng nhỏ dữ liệu của người dùng trong một phiên duyệt web. Khi đóng trình duyệt, dữ liệu trong sessionStorage sẽ bị xóa 🡺 **Xóa sạch toàn bộ dữ liệu sessionStorage**
* **Ví dụ cách dùng**: Trong ví dụ dưới, trước mỗi test case, Cypress sẽ xóa toàn bộ dữ liệu trong sessionStorage để đảm bảo rằng mỗi lần tìm kiếm đều bắt đầu từ trạng thái không có dữ liệu tìm kiếm trước đó

*describe('Tìm kiếm sản phẩm', () => {*

*beforeEach(() => {*

*cy****.clearAllSessionStorage()****;*

*cy.visit('https://example.com/san-pham');*

*});*

*it('Tìm kiếm sản phẩm thành công', () => {*

*// Các bước tìm kiếm sản phẩm*

*});*

*});*

## Lệnh .clearCookie()

* **Chức năng**: được sử dụng để loại bỏ một cookie cụ thể khỏi trình duyệt
* **Ví dụ cách dùng**:

*// Xóa cookie có tên 'session\_id'*

*cy.****clearCookie('session\_id')***

## Lệnh .clearCookies()

* **Chức năng**: được sử dụng để xóa **tất cả** các cookie của trình duyệt liên quan đến **domain** hiện tại
* **Ví dụ cách dùng**: Trong ví dụ dưới, trước mỗi test case, Cypress xóa sạch tất cả cookie để đảm bảo người dùng luôn ở trạng thái chưa đăng nhập.

*describe('Đăng nhập', () => {*

*beforeEach(() => {*

*cy****.clearCookies()****;*

*cy.visit('https://example.com');*

*});*

*it('Đăng nhập thành công', () => {*

*// Các bước đăng nhập*

*});*

*});*

## Lệnh .clearLocalStorage()

* **Chức năng**: Xóa dữ liệu trong localStorage của **domain** hiện tại
* **Ví dụ cách dùng**: Trong ví dụ dưới, trước mỗi test case, Cypress xóa sạch localStorage để đảm bảo người dùng luôn ở trạng thái chưa đăng nhập.

*describe('Đăng nhập', () => {*

*beforeEach(() => {*

*cy****.clearLocalStorage()****;*

*cy.visit('https://example.com');*

*});*

*it('Đăng nhập thành công', () => {*

*// Các bước đăng nhập*

*});*

*});*

## Lệnh .clock()

* **Chức năng**: cho phép điều khiển và thao tác thời gian trong các test
* **Ví dụ cách dùng**:

*describe('Kiểm tra tính năng đếm ngược', () => {*

*it('Đếm ngược đến 0', () => {*

*cy.visit('https://example.com/countdown');*

*cy****.clock()****; // Bắt đầu đồng hồ ảo*

*// Giả định có một phần tử hiển thị thời gian đếm ngược với id là 'countdown'*

*cy.get('#countdown').should('contain', '10');*

*// Tiến đồng hồ ảo về trước 10 giây*

*cy****.clock()****.tick(10000);*

*cy.get('#countdown').should('contain', '0');*

*});*

*});*

* + Bắt đầu đồng hồ ảo bằng cy.clock()
  + Kiểm tra xem phần tử đếm ngược ban đầu hiển thị số 10
  + Tiến đồng hồ ảo về trước 10 giây bằng clock.tick(10000)
  + Kiểm tra lại xem phần tử đếm ngược đã hiển thị số 0

## Lệnh .debug()

* **Chức năng**: giúp gỡ lỗi và hiểu rõ hơn về quá trình thực thi các test case. Khi gặp phải lỗi hoặc muốn kiểm tra xem ứng dụng đang ở trạng thái nào tại một bước cụ thể trong test, có thể sử dụng .debug() để tạm dừng test và khám phá sâu hơn 🡺 **Dừng và khám phá Test**
* **Ví dụ cách dùng**: Trong ví dụ dưới, sau khi click vào nút submit, test sẽ tạm dừng tại lệnh .debug(). Có thể sử dụng DevTools để kiểm tra xem nút có được click thành công không, form có được submit hay không, và dữ liệu có được gửi đi đúng hay không.

*cy.get('button[type="submit"]').click()*

***.debug()***

*.should('have.text', 'Submitted');*

## Lệnh .each()

* **Chức năng**: lặp qua các mảng hoặc các đối tượng có thuộc tính length (như các danh sách phần tử DOM)
* **Ví dụ cách dùng**:

*cy.get('li') // Lấy danh sách các phần tử li*

***.each(($li, index, $list)*** *=> {*

*// Thực hiện hành động trên mỗi phần tử li*

*cy.wrap($li)*

*.click()*

*.should('have.class', 'active');*

*});*

* + $li: Phần tử li hiện tại đang được lặp qua
  + index: Vị trí của phần tử trong danh sách
  + $list: Toàn bộ danh sách các phần tử li

## Lệnh .end()

* **Chức năng**: được sử dụng để kết thúc một chuỗi lệnh hiện tại và bắt đầu một chuỗi lệnh mới
* **Ví dụ cách dùng**:

*cy.get('.some-element')*

*.click()*

***.end()***

*.get('.another-element')*

*.should('be.visible');*

* + .get('.some-element')**:** Lấy một phần tử có class là "some-element"
  + .click()**:** Click vào phần tử đó
  + .end()**:** Kết thúc chuỗi lệnh trên phần tử "some-element" và bắt đầu một chuỗi lệnh mới
  + .get('.another-element')**:** Lấy một phần tử khác có class là "another-element"
  + .should('be.visible')**:** Kiểm tra xem phần tử mới có hiển thị hay không

## Lệnh .exec()

* **Chức năng**: cho phép thực thi các lệnh hệ thống(như các lệnh trong terminal) trực tiếp từ trong test
* **Ví dụ cách dùng**:

cy**.exec('node -v')**

.its('stdout')

.should('contain', 'v16')

* + Lệnh này sẽ chạy lệnh “node -v” để kiểm tra phiên bản Node.js đang sử dụng
  + its('stdout') : Lấy đầu ra của lệnh
  + should('contain', 'v16') : Kiểm tra xem đầu ra có chứa chuỗi "v16" hay không (nghĩa là phiên bản Node.js là 16)

## Lệnh .fixture()

* **Chức năng**: quản lý dữ liệu thử nghiệm. Cho phép tải các tệp bên ngoài, thường là các tệp JSON hoặc văn bản.
* **Ví dụ cách dùng**:
  + Tạo một tệp trong thư mục cypress/fixtures
  + Đặt tên cho tệp với một tên mô tả, ví dụ: users.json
  + Điền dữ liệu vào tệp

*[*

*{*

*"name": "Alice",*

*"email": "alice@example.com"*

*},*

*{*

*"name": "Bob",*

*"email": "bob@example.com"*

*}*

*]*

* + Tải Fixture trong Test Case

*cy****.fixture('users.json')****.then((users) => {*

*cy.get('#name').type(users[0].name);*

*cy.get('#email').type(users[0].email);*

*});*

## Lệnh .focus()

* **Chức năng**: được sử dụng để mô phỏng hành động khi nhấp chuột vào một phần tử trên trang web, khiến phần tử đó trở thành đối tượng nhận sự tập trung (focus) 🡺 **Đưa một phần tử vào trạng thái focus**
* **Ví dụ cách dùng**:

*cy.get('input[type="text"]')****.focus()***

* + **cy.get('input[type="text"]'):** Tìm kiếm một phần tử input có kiểu là text
  + **.focus():** Đưa phần tử vừa tìm được vào trạng thái focus

## Lệnh .getAllCookies()

* **Chức năng**: được sử dụng để lấy danh sách tất cả các cookie liên quan đến **domain** hiện tại
* **Ví dụ cách dùng**:

*cy****.getAllCookies()****.then((cookies) => {*

*// Làm việc với mảng cookies*

*// Ví dụ:*

*cy.log(cookies); // In ra danh sách tất cả các cookie*

*// Tìm kiếm một cookie cụ thể*

*const myCookie = cookies.find(cookie => cookie.name === 'myCookie');*

*cy.log(myCookie.value);*

*});*

## Lệnh .getAllLocalStorage()

* **Chức năng**: được sử dụng để lấy ra toàn bộ dữ liệu được lưu trữ trong localStorage của trình duyệt
* **Ví dụ cách dùng**:

*cy****.getAllLocalStorage()****.then((localStorageData) => {*

*// localStorageData là một object, các key là origin (ví dụ: "http://localhost:3000")*

*// và value là một object chứa các cặp key-value của localStorage*

*// Ví dụ:*

*cy.log(localStorageData['http://localhost:3000']); // Log ra dữ liệu localStorage của domain hiện tại*

*// Truy cập một giá trị cụ thể*

*const myValue = localStorageData['http://localhost:3000']['myKey'];*

*cy.log(myValue);*

*});*

## Lệnh .getAllSessionStorage()

* **Chức năng**: được sử dụng để lấy ra toàn bộ dữ liệu được lưu trữ trong sessionStorage của trình duyệt
* **Ví dụ cách dùng**:

*cy****.getAllSessionStorage()****.then((sessionStorageData) => {*

*// sessionStorageData là một object, các key là origin (ví dụ: "http://localhost:3000")*

*// và value là một object chứa các cặp key-value của sessionStorage*

*// Ví dụ:*

*cy.log(sessionStorageData['http://localhost:3000']); // Log ra dữ liệu sessionStorage của domain hiện tại*

*// Truy cập một giá trị cụ thể*

*const myValue = sessionStorageData['http://localhost:3000']['myKey'];*

*cy.log(myValue);*

*});*

## Lệnh .getCookie()

* **Chức năng**: được sử dụng để lấy thông tin chi tiết về một cookie cụ thể dựa trên tên của nó
* **Ví dụ cách dùng**:

*cy****.getCookie('myCookie')****.then((cookie) => {*

*// cookie là một đối tượng chứa thông tin chi tiết về cookie*

*cy.log(cookie.value); // In ra giá trị của cookie*

*expect(cookie.domain).to.equal('example.com'); // Kiểm tra domain của cookie*

*});*

## Lệnh .getCookies()

* **Chức năng**: được sử dụng để lấy danh sách tất cả các cookie liên quan đến **domain** hiện tại
* **Ví dụ cách dùng**:

*cy****.getCookies()****.then((cookies) => {*

*// Làm việc với mảng cookies*

*// Ví dụ:*

*cy.log(cookies); // In ra danh sách tất cả các cookie*

*// Tìm kiếm một cookie cụ thể*

*const myCookie = cookies.find(cookie => cookie.name === 'myCookie');*

*cy.log(myCookie.value);*

*});*

## Lệnh .go()

* **Chức năng**: cho phép mô phỏng hành động điều hướng khi nhấn nút "Quay lại" hoặc "Đi tới" trên trình duyệt
* **Ví dụ cách dùng**:

cy**.go('back')** // Quay lại một trang

cy**.go('forward')** // Đi tới một trang

cy**.go(-2)** // Quay lại hai trang

cy**.go(2)** // Đi tới hai trang

## Lệnh .hover()

* **Cypress không có sẵn lệnh** .hover()**.** Tuy nhiên, có thể dùng lệnh .trigger() để mô phỏng hành động di chuột trong Cypress
  + .trigger('**mouseover**')**:** Mô phỏng sự kiện di chuột vào một phần tử
  + .trigger('**mousemove**')**:** Mô phỏng sự kiện di chuyển chuột trên một phần tử

## Lệnh .intercept()

* **Chức năng**: kiểm soát và xử lý các yêu cầu và phản hồi HTTP trong quá trình kiểm thử
* **Ví dụ cách dùng**: Kiểm tra lỗi khi người dùng nhập mật khẩu sai

*cy****.intercept****('POST', '/login', (req) => {*

*// Kiểm tra xem mật khẩu có đúng không*

*if (req.body.password !== 'correct\_password') {*

*req.reply(401, { message: 'Unauthorized' });*

*} else {*

*req.reply(200, { token: 'your\_token' });*

*}*

*});*

## Lệnh .log()

* **Chức năng**: được sử dụng để in các thông tin, tin nhắn hoặc giá trị của biến ra **Command Log**
* **Ví dụ cách dùng**: Kiểm tra lỗi khi người dùng nhập mật khẩu sai

*cy****.intercept****('POST', '/login', (req) => {*

*// Kiểm tra xem mật khẩu có đúng không*

*if (req.body.password !== 'correct\_password') {*

*req.reply(401, { message: 'Unauthorized' });*

*} else {*

*req.reply(200, { token: 'your\_token' });*

*}*

*});*

## Lệnh .mount()

* **Chức năng**: cho phép gắn một thành phần vào DOM và tương tác với nó một cách độc lập, giúp viết các bài kiểm thử đơn vị và tích hợp hiệu quả hơn 🡺 **Nối liền các thành phần UI và kiểm thử**
* **Ví dụ cách dùng**: để sử dụng .mount(), cần cài đặt các thư viện Cypress tương ứng với framework đang sử dụng (ví dụ: cypress/react, cypress/vue). Sau đó có thể gắn một thành phần vào DOM như sau

*// Kiểm thử một button tăng số lượng*

*import Counter from './Counter';*

*describe('Counter', () => {*

*it('should increment the count when clicked', () => {*

*cy****.mount(<Counter />)****;*

*cy.contains('count: 0').should('exist');*

*cy.get('button').click();*

*cy.contains('count: 1').should('exist');*

*});*

*});*

## Lệnh .origin()

* **Chức năng**: cho phép thực hiện các tương tác với các trang web có nguồn gốc khác nhau trong cùng một bài kiểm thử, hữu ích khi cần kiểm tra các ứng dụng web có nhiều domain hoặc iframe 🡺 **Vượt qua giới hạn cùng gốc(Same-Origin Policy)**
* **Ví dụ cách dùng**:

*cy****.origin****('https://example.com', () => {*

*cy.visit('/about');*

*cy.contains('Về chúng tôi').click();*

*});*

* + cy.origin('https://example.com', () => { ... }) tạo ra một ngữ cảnh mới với nguồn gốc là <https://example.com>
  + Bất kỳ lệnh Cypress nào bên trong callback sẽ được thực thi trong ngữ cảnh mới này

## Lệnh .pause()

* **Chức năng**: tạm dừng việc thực thi test case tại một điểm cụ thể, cho phép kiểm tra trạng thái hiện tại của ứng dụng và gỡ lỗi nếu có vấn đề xảy ra 🡺 **Tạm dừng để kiểm tra và gỡ lỗi**
* **Ví dụ cách dùng**:

*cy.get('.my-button').click();*

***cy.pause()****; // Tạm dừng tại đây*

*cy.get('.result').should('contain', 'Kết quả');*

## Lệnh .readFile()

* **Chức năng**: đọc nội dung của một tệp tin trực tiếp từ hệ thống tệp của máy tính, và sau đó sử dụng nội dung đó trong các test case
* **Ví dụ cách dùng**:

*cy****.readFile****('cypress/fixtures/example.json')*

*.then(data => {*

*// Sử dụng dữ liệu đọc được*

*cy.log(data);*

*});*

* + cy.readFile('cypress/fixtures/example.json'): Đọc nội dung của tệp JSON có tên example.json nằm trong thư mục cypress/fixtures
  + .then(data => { ... }): Sử dụng hàm then() để xử lý dữ liệu đã đọc được

## Lệnh .reload()

* **Chức năng**: được sử dụng để mô phỏng hành động khi tải lại một trang web
* **Ví dụ cách dùng**:

*cy.visit('https://example.com');*

*// Thực hiện các tương tác với trang*

*cy.get('.some-button').click();*

*// Tải lại trang*

*cy****.reload()****;*

*// Kiểm tra trạng thái sau khi tải lại*

*cy.get('.some-element').should('be.visible');*

## Lệnh .request()

* **Chức năng**: cho phép gửi các yêu cầu HTTP (GET, POST, PUT, DELETE, v.v.) và kiểm tra phản hồi từ server mà không cần phải tương tác trực tiếp với giao diện người dùng 🡺 **Tương tác với API**
* **Ví dụ cách dùng**:

*cy****.request****('GET', '/api/users')*

*.its('body')*

*.then(users => {*

*expect(users.length).to.be.greaterThan(0);*

*});*

* + cy.request('GET', '/api/users'): Gửi một yêu cầu GET đến endpoint /api/users
  + .its('body'): Lấy phần body của phản hồi
  + .then(users => { ... }): Xử lý dữ liệu trong phần body

## Lệnh .screenshot()

* **Chức năng**: cho phép chụp ảnh màn hình của ứng dụng web tại một điểm cụ thể trong quá trình thực hiện test case
* **Ví dụ cách dùng**: Ảnh chụp màn hình sẽ được lưu vào thư mục cypress/screenshots

*// Kiểm tra giao diện đăng nhập*

*cy.visit('/login');*

*cy.get('#username').type('user123');*

*cy.get('#password').type('password123');*

*cy.get('.submit-button').click();*

*// Chụp ảnh màn hình trang dashboard sau khi đăng nhập*

*cy****.screenshot('dashboard')****;*

## Lệnh .session()

* **Chức năng**: quản lý trạng thái phiên làm việc của trình duyệt
* **Ví dụ cách dùng**:

*cy****.session****('mySession', () => {*

*cy.visit('/login');*

*// Thực hiện đăng nhập*

*cy.get('#username').type('user');*

*cy.get('#password').type('password');*

*cy.get('.submit-button').click();*

*// Xác thực trạng thái đăng nhập*

*cy.url().should('include', '/dashboard');*

*});*

* + cy.session('mySession', () => { ... }): Tạo một phiên làm việc có tên là "mySession"
  + Tất cả các lệnh bên trong callback sẽ được thực thi trong phiên làm việc này
  + Trạng thái của phiên làm việc (cookies, localStorage, sessionStorage) sẽ được lưu lại

## Lệnh .setCookie()

* **Chức năng**: cho phép quản lý, thiết lập cookie trong trình duyệt
* **Ví dụ cách dùng**: Thiết lập một cookie có tên là "myCookie" với giá trị là "myValue"

*cy****.setCookie****('myCookie', 'myValue');*

## Lệnh .spread()

* **Chức năng**: cho phép phân tách các giá trị trong mảng thành các đối số riêng biệt, giúp cho việc thao tác và kiểm tra trở nên linh hoạt hơn
* **Ví dụ cách dùng**:

*cy.get('.product-item') // Lấy tất cả các phần tử sản phẩm*

*.then($products => {*

*// Sử dụng .spread() để lấy 2 sản phẩm đầu tiên*

*cy.wrap($products)****.spread((product1, product2)*** *=> {*

*// Kiểm tra giá của mỗi sản phẩm*

*cy.wrap(product1).find('.price').should('have.text', '100$');*

*cy.wrap(product2).find('.price').should('have.text', '200$');*

*});*

*});*

## Lệnh .spy()

* **Chức năng**: theo dõi và kiểm tra cách các hàm được gọi trong ứng dụng
* **Ví dụ cách dùng**:

*// Giả sử có một hàm calculateTotal()*

*function calculateTotal(price, quantity) {*

*return price \* quantity;*

*}*

*// Tạo một spy cho hàm calculateTotal()*

*const calculateTotalSpy = cy****.spy(window, 'calculateTotal')****;*

*// Gọi hàm calculateTotal()*

*cy.get('.calculate-button').click();*

*// Kiểm tra xem hàm calculateTotal() có được gọi hay không*

*expect(calculateTotalSpy).to.have.been.calledOnce;*

*// Kiểm tra các đối số truyền vào hàm*

*expect(calculateTotalSpy).to.have.been.calledWith(100, 2);*

* + **.calledOnce:** Kiểm tra xem hàm đã được gọi một lần
  + **.calledTwice:** Kiểm tra xem hàm đã được gọi hai lần
  + **.calledWith(arg1, arg2):** Kiểm tra xem hàm đã được gọi với các đối số cụ thể
  + **.returned(value):** Kiểm tra xem hàm đã trả về giá trị cụ thể

## Lệnh .stub()

* **Chức năng**: cho phép thay thế một hàm bằng một hàm giả (stub), từ đó có thể kiểm soát hoàn toàn hành vi của hàm đó trong quá trình kiểm thử
* **Ví dụ cách dùng**:

*// Giả sử chúng ta có một hàm fetchUserData()*

*function fetchUserData() {*

*// Thực hiện một yêu cầu API để lấy dữ liệu người dùng*

*// ...*

*}*

*// Tạo một stub cho hàm fetchUserData()*

*const fetchUserDataStub = cy****.stub()****.as('fetchUserData');*

*// Thay thế hàm fetchUserData() bằng stub*

*cy.on('window:alert', fetchUserDataStub);*

*// Gọi hàm fetchUserData()*

*cy.get('.fetch-button').click();*

*// Kiểm tra xem stub có được gọi hay không*

*cy.get('@fetchUserData').should('have.been.calledOnce');*

*// Kiểm tra các đối số truyền vào stub*

*cy.get('@fetchUserData').should('have.been.calledWith', 'userId');*

*// Thiết lập giá trị trả về của stub*

*fetchUserDataStub.returns({ name: 'John Doe', email: 'johndoe@example.com' });*

## Lệnh .submit()

* **Chức năng**: mô phỏng hành động khi gửi một biểu mẫu trên trang web
* **Ví dụ cách dùng**:

*// HTML*

*<form id="loginForm">*

*<input type="text" id="username" />*

*<input type="password" id="password" />*

*<button type="submit">Đăng nhập</button>*

*</form>*

*cy.visit('/login');*

*cy.get('#username').type('admin');*

*cy.get('#password').type('password123');*

*cy.get('#loginForm')****.submit()****;*

## Lệnh .task()

* **Chức năng**: cho phép thực hiện các tác vụ bên ngoài phạm vi mà Cypress có thể trực tiếp tương tác
* **Ví dụ cách dùng**:
  + Trong file cypress/plugins/index.js, định nghĩa các task mà muốn sử dụng

*module.exports = (on, config) => {*

*on('task', {*

*exampleTask(message) {*

*return `Hello, ${message}!`;*

*}*

*});*

*};*

* + Trong file test, sử dụng lệnh cy.task() để gọi task đã định nghĩa

*cy****.task****('exampleTask', 'world').then(result => {*

*expect(result).to.equal('Hello, world!');*

*});*

## Lệnh .then()

* **Chức năng**: cầu nối giữa các lệnh trong chuỗi test. Nó cho phép thực hiện các hành động tiếp theo sau khi một lệnh trước đó hoàn thành
* **Ví dụ cách dùng**:

*cy.get('button.submit').click()*

***.then****(() => {*

*// Kiểm tra xem một thông báo thành công có xuất hiện không*

*cy.contains('Thao tác thành công').should('be.visible');*

*});*

* + cy.get('button.submit').click(): Tìm nút có class là "submit" và click vào nó
  + .then(() => { ... }): Khi nút được click, hàm callback bên trong .then() sẽ được thực thi
  + cy.contains('Thao tác thành công').should('be.visible'): Kiểm tra xem có một phần tử chứa văn bản "Thao tác thành công" và nó có hiển thị trên màn hình hay không

## Lệnh .tick()

* **Chức năng**: cho phép điều khiển và mô phỏng sự trôi qua của thời gian trong các test
* **Ví dụ cách dùng**:

*// Giả sử có một nút khi click vào sẽ hiện một thông báo sau 2 giây*

*cy.get('button').click();*

*// Di chuyển đồng hồ về phía trước 2 giây*

*cy****.tick(2000)****;*

*// Kiểm tra xem thông báo đã hiện ra chưa*

*cy.contains('Thông báo').should('be.visible');*

## Lệnh .viewport()

* **Chức năng**: cho phép điều khiển kích thước và định hướng của cửa sổ trình duyệt trong quá trình chạy test
* **Ví dụ cách dùng**:

*describe('Ứng dụng của tôi', () => {*

*it('phải hiển thị đúng trên màn hình nhỏ', () => {*

*cy****.viewport('iphone-5')****;*

*cy.visit('/');*

*cy.get('.mobile-menu').should('be.visible');*

*});*

*it('phải hiển thị đúng trên màn hình lớn', () => {*

*cy****.viewport(1920, 1080)****;*

*cy.visit('/');*

*cy.get('.desktop-menu').should('be.visible');*

*});*

*});*

## Lệnh .visit()

* **Chức năng**: được sử dụng để điều hướng đến một URL cụ thể trong quá trình chạy test
* **Ví dụ cách dùng**:

*// Truy cập vào một trang với tham số query*

*cy****.visit****('https://example.com?page=2');*

*// Truy cập vào một file HTML trong thư mục fixtures*

*cy****.visit****('cypress/fixtures/login.html');*

## Lệnh .wait()

* **Chức năng**: được sử dụng để tạm dừng việc thực thi test cho đến khi một điều kiện nhất định được đáp ứng
* **Ví dụ cách dùng**:

*// Chờ 2 giây*

*cy****.wait(2000)****;*

*// Chờ cho một yêu cầu API có alias là 'getUser' hoàn thành*

*cy.intercept('GET', '/users').as('getUser');*

*cy.get('.button').click();*

*cy.wait('@getUser');*

## Lệnh .within()

* **Chức năng**: được sử dụng để giới hạn phạm vi của các lệnh Cypress tiếp theo vào bên trong một phần tử cụ thể
* **Ví dụ cách dùng**:

*cy.get('form')****.within****(() => {*

*cy.get('input[name="username"]').type('myusername');*

*cy.get('input[name="password"]').type('mypassword');*

*});*

* + cy.get('form'): Tìm đến form
  + .within(() => { ... }): Bắt đầu một khối lệnh mới, tất cả các lệnh bên trong khối này sẽ chỉ tìm kiếm các phần tử con của form
  + cy.get('input[name="username"]'): Tìm đến input có name là "username" **bên trong form**

## Lệnh .wrap()

* **Chức năng**: cho phép "bao bọc" một giá trị bất kỳ vào một đối tượng Cypress Chainable(một chuỗi các lệnh Cypress được liên kết với nhau để tạo thành một dòng thực thi test), từ đó mở rộng khả năng sử dụng các lệnh Cypress cho giá trị đó 🡺 **Bao bọc các giá trị và tạo chuỗi lệnh**
* **Ví dụ cách dùng**:

*// Ví dụ 1: Bao bọc một mảng*

*const numbers = [1, 2, 3];*

*cy****.wrap(numbers)****.should('have.length', 3);*

*// Ví dụ 2: Bao bọc một đối tượng*

*const user = { name: 'John Doe', age: 30 };*

*cy****.wrap(user)****.its('name').should('eq', 'John Doe');*

*// Ví dụ 3: Bao bọc kết quả của một hàm*

*cy****.wrap(getUserData())****.its('email').should('include', '@example.com');*

## Lệnh .writeFile()

* **Chức năng**: được sử dụng để ghi dữ liệu vào một tệp trên hệ thống
* **Ví dụ cách dùng**:

*// Ghi một chuỗi vào tệp*

*cy****.writeFile****('cypress/fixtures/data.txt', 'Hello, world!');*

*// Ghi một đối tượng JSON vào tệp*

*const userData = {*

*name: 'John Doe',*

*age: 30,*

*email: 'johndoe@example.com'*

*};*

*cy****.writeFile****('cypress/fixtures/user.json', userData);*

## Lệnh .only()

* **Chức năng**: được sử dụng để chỉ định một test case duy nhất hoặc một nhóm test case cụ thể để thực thi, trong khi bỏ qua tất cả các test case khác trong một tập tin hoặc một suite test 🡺 **Tập trung vào một test case cụ thể**
* **Ví dụ cách dùng**:

*it.only('This test will run', () => {*

*// Các lệnh test ở đây*

*});*

*describe('My test suite', () => {*

*it('This test will be skipped', () => {*

*// Các lệnh test ở đây*

*});*

*it.only('Only this test will run', () => {*

*// Các lệnh test ở đây*

*});*

*});*

* + it.only() được sử dụng để đánh dấu một test case sẽ được thực thi duy nhất
  + Các test case không có .only() sẽ bị bỏ qua

## Lệnh .skip()

* **Chức năng**: được sử dụng để tạm thời bỏ qua một test case hoặc một nhóm test case
* **Ví dụ cách dùng**:

*it.skip('This test will be skipped', () => {*

*// Các lệnh test ở đây*

*});*

*describe('My test suite', () => {*

*it.skip('This test will be skipped', () => {*

*// Các lệnh test ở đây*

*});*

*it('This test will run', () => {*

*// Các lệnh test ở đây*

*});*

*});*

* + it.skip() được sử dụng để đánh dấu một test case sẽ bị bỏ qua
  + Các test case không có .skip() sẽ được thực thi bình thường