Testing React

Cory House

Consultant

@housecor | www.reactjsconsulting.com

Here's the plan

Testing Technologies

- Testing frameworks
- Helper libraries

We'll test React components

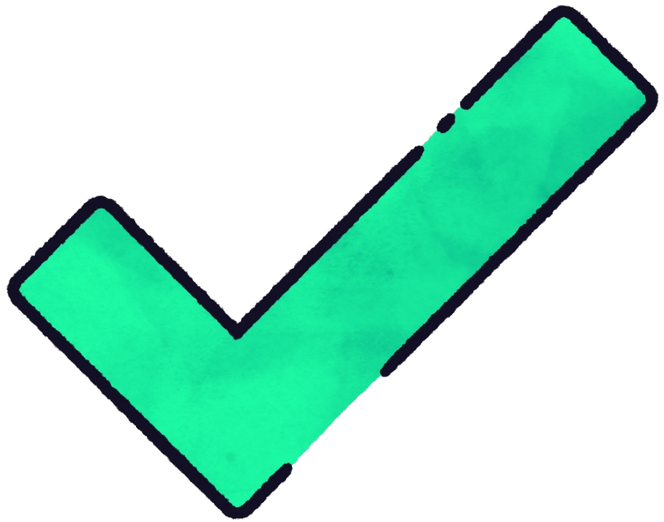
- Jest
- Enzyme
- React Testing Library

Testing Frameworks

Test Frameworks

- Jest
- Vitest
- Mocha
- Jasmine
- Tape
- AVA

Helper Libraries



React testing library

Example

```
import {render, screen} from '@testing-library/react' // (or /dom, /vue, ...)

test('should show login form', () => {
  render(<Login />)
  const input = screen.getByLabelText('Username')
  // Events and assertions...
})
```

Priority

Based on the Guiding Principles, your test should resemble how users interact with your code (component, page, etc.) as much as possible. With this in mind, we recommend this order of priority:

1. **Queries Accessible to Everyone** Queries that reflect the experience of visual/mouse users as well as those that use assistive technology.
    i. `getByRole`: This can be used to query every element that is exposed in the accessibility tree. With the `name` option you can filter the returned elements by their accessible name. This should be your top preference for just about everything. There's not much you can't get with this (if you can't, it's possible your UI is inaccessible). Most often, this will be used with the `name` option like so: `getByRole('button', {name: /submit/i})`. Check the list of roles.
    ii. `getByLabelText`: This method is really good for form fields. When navigating through a website form, users find elements using label text. This method emulates that behavior, so it should be your top preference.
    iii. `getByPlaceholderText`: A placeholder is not a substitute for a label. But if that's all you have, then it's better than alternatives.
    iv. `getByText`: Outside of forms, text content is the main way users find elements. This method can be used to find non-interactive elements (like divs, spans, and paragraphs).
    v. `getByDisplayValue`: The current value of a form element can be useful when navigating a page with filled-in values.
2. **Semantic Queries** HTML5 and ARIA compliant selectors. Note that the user experience of interacting with these attributes varies greatly across browsers and assistive technology.
    i. `getByAltText`: If your element is one which supports `alt` text (`img`, `area`, `input`, and any custom element), then you can use this to find that element.
    ii. `getByTitle`: The title attribute is not consistently read by screenreaders, and is not visible by default for sighted users
3. **Test IDs**
    i. `getByTestId`: The user cannot see (or hear) these, so this is only recommended for cases where you can't match by role or text or it doesn't make sense (e.g. the text is dynamic).

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
`getBy...`	Throw error	Return element	Throw error	No
`queryBy...`	Return `null`	Return element	Throw error	No
`findBy...`	Throw error	Return element	Throw error	Yes
Multiple Elements				
`getAllBy...`	Throw error	Return array	Return array	No
`queryAllBy...`	Return `[]`	Return array	Return array	No
`findAllBy...`	Throw error	Return array	Return array	Yes

Where to Test

Where to Test

Browser

In-memory DOM

Where Do Test Files Belong?

We'll place tests alongside the file under test.

Why?

- Easy imports. Always ./filenameUnderTest
- Clear visibility
- Convenient to open
- Move files and tests together

Our Plan

What	React components and Redux
How	Jest
Where	In-memory DOM via JSDOM
Helper	React Testing Library

Demo

Test React components

- Configure Jest
- Create Snapshot test
- Test with Jest and React Testing Library

Testing Presentation Components

Summary

Tested React components

- Framework: Jest
- Helper: React Testing Library
- Where: In-memory DOM via Node

Up Next:

Testing Redux