Getting Started With JavaScript Testing for WordPress

Josh Pollock

PHP & JavaScript Engineer/ Dog Enthusiast

- Lead Web Engineer 10up
- JoshPress.net
- @josh412
- Josh412.eth

Slides and Code

Slides

- 👀 View Slides
- Download Slides As PDF
- Source Code For Slides

Example Code

- Example Code For Part One
- Example Code For Part Two

Find a bug or typo? Pull requests are welcome.

What Are We Covering?

- Types of tests.
- Unit and integration tests in React apps.
- Unit and integration tests in Gutenberg blocks.
- Structuring blocks for testing.

Why Test?

- Does My Code Work?
- How would I know?

Types Of Tests

What Questions Do Tests Answer?

Types Of Tests

Unit Tests

Does A Component Work In Isolation?

Types Of Tests

Integration (Feature) Tests

Do The Components Work Together?

Types Of Tests

Acceptance (e2e) Tests

Does the whole system work together?

JavaScript Testing In And Around WordPress

Part One: Testing React Apps

Example Code For Part One

How React Works

The Short Version

How React Works

Step 1

React creates an object representation of nodes representing a user interface.

This code:

```
React.createElement("ul", { className: "inline-list" }, [
  React.createElement("li", { key: "first", className: "list-item" }, 'First Item')
]);
```

Becomes something like this:

```
 {
  ul: { className: "inline-list" }, [
    "li", { key: "first", className: "list-item" }, 'First Item'
  ]
 }
```

How React Works

Step 2

A "renderer" converts that object to a useable interface.

- ReactDOM renders React as DOM tree and appended to DOM.

```
ReactDOM.render(<App />, domElement);
```

- ReactDOMServer renders to an HTML string for server to send to client.

```
ReactDOMServer.renderToString(<App />);
```

Test Renderers

- React Test Renderer
  - Good for basic tests and snapshots. No JSDOM.
- Enzyme
  - Renders to JSDOM. Good for testing events and class components methods/ state.
- React Testing Library
  - Good for basic test, snapshots, testing events, testing hooks, etc. Uses JSDOM.

The Test Suite

- Test Runner
  - Runs the tests
  - Examples: Jest or phpunit
- Test Renderers
  - Creates and inspects output
- Assertions
  - Tests the output
  - Example: Chai

Zero-Config Testing

(and more)

- react-scripts
  - `react-scripts test`
  - Used by create-react-app
- @wordpress/scripts
  - `wordpress-scripts test`
  - Developed for Gutenberg, great for your plugins.

```
npx create-react-app
```

Let's Write Some Tests

And A Web App :)

Create A React App

```
# install create-react-app
npx create-react-app
# Run the included test
yarn test
```

Testing Included!

Create React App comes with one test.

This is an acceptance test. It tests if **anything** is broken.

Testing Included!

Test The App Renders

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
it("renders without crashing", () => {
  const div = document.createElement("div");
  ReactDOM.render(<App />, div);
  ReactDOM.unmountComponentAtNode(div);
});
```

Questions To Ask?

- How do I know the components works?
  - Answer with unit tests
- How do I know the components work together?
  - Answer with integration/ feature tests
- What is the most realistic test of the program?
  - Answer with acceptance/ e2e tests

App Spec

Create a one page app that:

- Displays a value
- Has an input to change that value

Test Spec

- Unit tests:
  - Does display component display the supplied value?
  - Does edit component display the value?
  - Does the edit component supply updated value to onChange callback?

Layout Of Our Test File

`test()` Syntax

```
//Import React
import React from "react";
//Import test renderer
import TestRenderer from "react-test-renderer";
//Import component to test
import { DisplayValue } from "./DisplayValue";

test("Component renders value", () => {});

test("Component has supplied class name", () => {});
```

Layout Of Our Test File

BDD Style

```
//Import React
import React from "react";
//Import test renderer
import TestRenderer from "react-test-renderer";
//Import component to test
import { DisplayValue } from "./DisplayValue";

describe("EditValue Component", () => {
  it("Has the supplied value in the input", () => {});

  it("Passes string to onChange when changed", () => {});
});
```

Unit Testing React Components

Install React Test Renderer

```
yarn add react-test-renderer
```

[Documentation]()

What We Are Testing

```
import React from 'react';
export const DisplayValue = ({value,className}) =>(
    <div className={className}>{value}</div>
);
```

Unit Testing React Components

Snapshot Testing

Renders Component To JSON

Stores JSON representation of component in file system

- Make sure your props went to the right places.
- Forces you to **commit** to changes.

Unit Testing React Components

Create A Snapshot Test

With React Test Renderer

```
test("Component renders correctly", () => {
  expect(
    TestRenderer.create(
      <DisplayValue value={"The Value"} className={"the-class-name"} />
    ).toJSON()
  ).toMatchSnapshot();
});
```

Testing Events

Testing Library

[*Documentation*]

Install React Testing Library

```
yarn add @testing-library/react
```

Testing Events

Test On Change Event

```
import { render, cleanup, fireEvent } from "@testing-library/react";
describe("EditValue component", () => {
  afterEach(cleanup); //reset JSDOM after each test
  it("Calls the onChange function", () => {
    //put test here
  });
  it("Has the right value", () => {
    //put test here
  });
});
```

Testing Events

Test On Change Event

```
test("Calling the onChange function", () => {
    const onChange = jest.fn();
    const { getByLabelText } = render(<EditValue onChange={onChange} value={""} id=
{'input-test'}  className={"some-class"}  />);
    fireEvent.change(getByLabelText("Set Value"), {
        target: { value: "New Value" }
    });
    expect(onChange).toHaveBeenCalledTimes(1);
});
```

Testing Events

Test On Change Event

```
const onChange = jest.fn();
test("Passes updated value, not event to onChange callback", () => {
    const onChange = jest.fn();
    const { getByDisplayValue } = render(<EditValue onChange={onChange} value={"Old
Value"} id="input-test" className={"some-class"}    />);
    fireEvent.change(getByDisplayValue("Old Value"), {
        target: { value: "New Value" }
    });
    expect(onChange).toHaveBeenCalledWith("New Value");
});
```

Unit Testing React Components

Snapshot Testing

With React Testing Library

```
test( 'matches snapshot', () => {
    const {container} = render(<EditValue onChange={jest.fn()} value={"Hi Roy"} id=
{'some-id'} className={"some-class"}   /> );
    expect( container ).toMatchSnapshot();
});
```

Integration Tests

Do the two components work together as expected?

Integration Tests

Does One Component Update The Other?

```
it("Displays the updated value when value changes", () => {
  const { container, getByTestId } = render(<App />);
  expect(container.querySelector(".display-value").textContent).toBe("Hi Roy");
  fireEvent.change(getByTestId("the-input"), {
    target: { value: "New Value" }
  });
  expect(container.querySelector(".display-value").textContent).toBe(
    "New Value"
  );
});
```

Test For Accesibility Errors

Using dequeue's aXe

```
# Add react-axe
yarn add react-axe --dev
# Add react-axe for Jest
yarn add jest-axe --dev
```

Test App For Accesibility Errors

Does the accessibility scanner raise errors?

This does NOT mean your app is accessible!

```
import React from "react";
import server from "react-dom/server";
import App from "./App";
import { render, fireEvent, cleanup } from "@testing-library/react";

const { axe, toHaveNoViolations } = require("jest-axe");
expect.extend(toHaveNoViolations);

it("Raises no a11y errors", async () => {
  const html = server.renderToString(<App />);
  const results = await axe(html);
  expect(results).toHaveNoViolations();
});
```

Review App Spec

Create a one page app that:

- Displays a value
- Has an input to change that value

JavaScript Testing In And Around WordPress

Part Two: Testing Gutenberg Blocks

Example Code Part Two

Testing Gutenberg Blocks

It's React, Test It The Same Way

```
yarn add @wordpress/scripts
```

Let's Write Some Tests

And A Plugin

Spec

A block for showing some text.

- Block editor will display a value or when selected an inline editor
- Block Tutorial

What Is `@wordpress/scripts` ??

- React-scripts inspired zero-config build tool for WordPress plugins with blocks.
- Provides:
  - Compilers
  - Linters
  - Test runner
  - e2e tests
  - Local development

Setting Up Plugin For Testing

Install WordPress scripts

```
# Install WordPress scripts
yarn add @wordpress/scripts
```

Add Scripts To package.json

See README

```
{
  "scripts": {
    "build": "wp-scripts build",
    "start": "wp-scripts start",
    "test:unit": "wp-scripts test-unit-js --config jest.config.js",
  }
}
```

Jest Is The Test Runner

Testing works the same, we can use same renderers.

`@wordpress/scripts` works on top of Jest, webpack, Babel, etc.

Structuring Blocks For Testing

One file that registers the block.

Structuring Blocks For Testing

The Block

```
import { registerBlockType } from "@wordpress/blocks";
import Edit from './Edit';
import Save from './Save';

const blockConfig = require('./block.json');
registerBlockType(blockConfig.name, {
    ...blockConfig,
    edit: Edit,
    save: Save
});
```

Structuring Blocks For Testing

Edit And Save Callbacks

The edit and save callback are composed in separate files.

Edit Callback

```
import { TextControl } from '@wordpress/components';
import { __ } from '@wordpress/i18n';
import { useBlockProps } from '@wordpress/block-editor';

export const Editor = ({ value, onChange, isSelected }) => (
    <>
        {isSelected ?
            <TextControl
                value={value}
                onChange={onChange}
            /> : <p>{value}</p>
        }
    </>
);
export default function Edit({ attributes, setAttributes, isSelected }) {
    return (
        <div {...useBlockProps()}>
            <Editor isSelected={isSelected} value={attributes.content} onChange=
{(content) => setAttributes({ content })} />
        </div>
    );
}
```

Test Edit Callback

```
//Import component to test
import { Editor } from './Edit';
describe("Editor componet", () => {
    afterEach(cleanup);
    it('matches snapshot when selected', () => {});
    it('matches snapshot when not selected', () => {});
    it("Calls the onchange function", () => {});
    it("Passes updated value, not event to onChange callback", () => { });
});
```

Snapshot Test Block Editor Component

```
it('matches snapshot when selected', () => {
    const onChange = jest.fn();
    const { container } = render(<Editor
        onChange={onChange}
        value={'Tacos'}
        isSelected="true"
    />);
    expect(container).toMatchSnapshot();
});
```

Testing Events For Block Editor Component

```
it("Calls the onChange function", () => {
  const onChange = jest.fn();
  const { getByDisplayValue } = render(<Editor
      onChange={onChange}
      value={'Salad'}
      isSelected="false"
  />);
  fireEvent.change(getByDisplayValue("Salad"), {
      target: { value: "New Value" }
  });
  expect(onChange).toHaveBeenCalledTimes(1);
  expect(onChange).toHaveBeenCalledWith("Boring Water");
});
```

Save Callback

```
import { __ } from '@wordpress/i18n';
import { useBlockProps } from '@wordpress/block-editor';

export default function save({ attributes }) {
  return <div {...useBlockProps.save()}>{attributes.content}</div>;
}
```

Test Save Callback

- Don't test the framework.
- Probably better to rely on acceptance testing or manual QA.

🙆‍♀️ Any Questions? 🙆‍♀️

- Slides, And Links
- Download Slides As PDF

🙏 Thank You! 🙏

- JoshPress.net
- @josh412
- Josh412.eth