

# JavaScript Testing In And Around WordPress

Josh Pollock



 **Hi I'm Josh** 

**JavaScript & PHP Developer: CalderaWP**



- [JoshPress.net](https://joshpress.net)
- [CalderaForms.com](https://calderaforms.com)
- [@josh412](https://twitter.com/josh412)

# Slides/ Code

👁️ JSWP-Testing.Caldera.dev 👁️

- [View Slides](#)
- [Download Slides As PDF](#)
- [Source Code For Slides](#)
- [Related Blog Post](#)

## Example Code

- [Example Code For Part One](#)
- [Example Code For Part Two](#)

Find a bug or typo? Pull requests are welcome.

# 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

**Everything In Context....**

# Step 1

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

- It does not produce HTML.

- `React.createElement( 'div', {className: 'alert'}, 'Something Happened' );`

## Step 2

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

```
const app = React.createElement( 'div', {className: 'alert'}, 'Something Happened' );
```

- 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.

## 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?

# Test Spec

- Integration Tests:
  - Does the display value change with the input?

# 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", () => {});
```



## BDD Style

```
describe("EditValue Component", () => {  
  //Shared mock onChange function  
  let onChange = jest.fn();  
  beforeEach(() => {  
    //Reset onChange mock before each test.  
    onChange = jest.fn();  
  });  
  
  it("Has the supplied value in the input", () => {});  
  
  it("Passes string to onChange when changed", () => {});  
});
```

# Install React Test Renderer

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

# Unit Testing React Components

## Find Props

```
//Probably don't do this
test("Component renders value", () => {
  const value = "The Value";
  const testRenderer = TestRenderer.create(<DisplayValue value={value} />);
  //Get the rendered node
  const testInstance = testRenderer.root;
  //find the div and make sure it has the right text
  expect(testInstance.findByType("div").props.children).toBe(value);
});
```

**Do This For Every Prop?**

**That Is Testing React, Not Your Application**

# **Snapshot Testing**

## **Renders Component To JSON**

Stores JSON in file system

# Snapshot Testing

- Snapshots Accomplish Two Things:
  - Make sure your props went to the right places.
  - Force you to **commit** to changes.

## Create A Snapshot Test

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



# Testing Events

React testing library is best for this. Enzyme is an alternative.

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

## 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
  });
});
```

## Test On Change Event

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

## Test On Change Event

```
const onChange = jest.fn();
const { getByTestId } = render(
  <EditValue
    onChange={onChange}
    value=""
    id="input-test"
    className="some-class"
  />
);
fireEvent.change(getByTestId("input-test"), {
  target: { value: "New Value" }
});
expect(onChange).toHaveBeenCalledWith('New Value');
```

# Snapshot Testing

With React Testing Library

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

# Integration Tests

Do the two components work together as expected?

# Integration Test

```
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 Accessibility 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 Accessibility Errors

## Does the accessibility scanner raise errors?

This does NOT mean your app is accessible!

```
const { axe, toHaveNoViolations } = require("jest-axe");
expect.extend(toHaveNoViolations);
import React from "react";
import server from "react-dom/server";
import App from "../App";
import { render, fireEvent, cleanup } from "@testing-library/react";

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](#)

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

# Let's Write Some Tests

And A Plugin

# Spec

A block for showing some text.

- The components for the app should be reused.
- The block preview and rendered content should be identical.
- The control for the value should appear in the block's inspector controls.

# Test Spec

## **Integration Test**

Will Gutenberg be able to manage our component's state?

# Test Spec

## **e2e Test**

Does our plugin activate without errors?

Does our block appear in the block chooser?

# 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:e2e": "wp-scripts test-e2e --config e2e/jest.config.js",
    "test:unit": "wp-scripts test-unit-js --config jest.config.js",
    "env:start": "bash start.sh"
  }
}
```

# 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 Easy Testing

The file that builds the block to do nothing but build the block.

# The Block

```
import { registerBlockType } from "@wordpress/blocks";
import { Editor } from "../components/Editor";
import { Save } from "../components/Save";
const blockConfig = require("../block.json");
const { name, title, attributes, category, keywords } = blockConfig;

registerBlockType(name, {
  title,
  attributes,
  category,
  keywords,
  edit: props => <Editor {...props} />,
  save: props => <Save {...props} />
});
```

## **Edit And Save Callbacks**

The edit and save callback are composed in separate files, importing components built for the app.

### **Responsibility: Map Props**

# Edit Callback

```
import React, { Fragment } from "react";
import { EditValue } from "../app/EditValue";
import { DisplayValue } from "../app/DisplayValue";
import { InspectorControls } from "@wordpress/block-editor";
export const Editor = ({ attributes, setAttributes, className, clientId }) => {
  //Change handler
  const onChange = value => setAttributes({ value });
  //current value
  const { value } = attributes;
  return (
    <Fragment>
      <InspectorControls>
        <EditValue
          className={` ${className}-editor` }
          id={clientId}
          value={value}
          onChange={onChange}
        />
      </InspectorControls>
      <DisplayValue value={value} className={className} />
    </Fragment>
  );
};
```

## Test Edit Callback

```
describe("Editor componet", () => {
  afterEach(cleanup);
  it("matches snapshot", () => {
    const attributes = { value: "Hi Roy" };
    const setAttributes = jest.fn();
    expect(
      render(
        <Editor
          {...{
            attributes,
            setAttributes,
            clientId: "random-id",
            className: "wp-blocks-whatever"
          }}
        />
      )
    ).toMatchSnapshot();
  });
});
```



## Save Callback

```
import React from "react";
import { DisplayValue } from "../app/DisplayValue";
export const Save = ({ attributes, className }) => {
  return <DisplayValue value={attributes.value} className={className} />;
};
```

## Test Save Callback

```
describe("Save componet", () => {
  afterEach(cleanup);
  it("matches snapshot", () => {
    const attributes = { value: "Hi Roy" };
    expect(
      render(
        <Save
          {...{
            attributes,
            clientId: "random-id",
            className: "wp-blocks-whatever"
          }}
        />
      )
    ).toMatchSnapshot();
  });
});
```

# End To End Testing Gutenberg Blocks

- Assuming that all of the components work, does the program function as expected.
- Test as a user.
- [Introductory Post](#)
- [Documentation](#)

# How To Setup Up WordPress End To End Tests

To make things easier, add the WordPress e2e test utilities:

```
# Add e2e test utilities  
yarn add @wordpress/e2e-test-utils
```

# Configure Jest

A separate Jest config is needed to make sure it does NOT run unit tests.

- [Jest Config For Unit Tests](#)
- [Jest Config For e2e Tests](#)

```
const defaultConfig = require("../node_modules/@wordpress/scripts/config/jest-unit.config.js");
module.exports = {
  //use the default from WordPress for everything...
  ...defaultConfig,
  //Except test ignore, where we need to ignore our e2e test directory
  testPathIgnorePatterns: ["/.git/", "/node_modules/", "<rootDir>/e2e"]
};
```

This is based on [WordPress core's e2e tests](#)

# Uses Puppeteer To Automate Chrome

Easiest if you have WordPress running locally in Docker [like core does](#)

[Copy my copy of core's local development](#)

**Test That Block Works**

# Use Helpers

Import helper functions from `@wordpress/e2e-test-utils`

```
import {  
  insertBlock,  
  getEditedPostContent,  
  createNewPost,  
  activatePlugin  
} from "@wordpress/e2e-test-utils";
```



## Test Adding Block

```
describe("Block", () => {  
  beforeEach(async () => {  
    await activatePlugin("josh-jswp/josh-jswp.php");  
  });  
  it("Can add block", async () => {  
    await createNewPost();  
    await insertBlock("Josh Block");  
    expect(await getEditedPostContent()).toMatchSnapshot();  
  });  
});
```

# Do NOT e2e Test Everything

e2e tests ensure that the system works together.

They are a compliment to less expensive unit/ integration tests.

# Recomendations

- Assume your components will be reused.
  - Test in isolation.
- Start with unit tests on new projects.
  - Makes refactoring faster and safer.
- For legacy projects, start with acceptance tests.
  - Covers more, makes refactoring for unit-testablity safer.
- Do Not Forget To Test For Accesibility errors
  - [Good Post On Reporting a11y Errors In Console](#)
- Follow and learn from [Kent C. Dodds](#)
  - Author of React Testing Library and many great posts and videos about React, React testing.

# Any Questions?

- [Slides, And Links](#)
- [Download Slides As PDF](#)

 **Thank You!** 

- [JoshPress.net](#)
- [CalderaForms.com](#)
- [@josh412](#)