Frontend Testing with Jest and Cypress

Eric Campbell

Testing Fundamentals

Testing Fundamentals

- "3" Types
  1. Unit Tests: Test one unit (e.g. a single function)
  2. Integration Tests: Tests how several different units integrate with each other (e.g. single portion of the application)
  3. End-to-End Test: Test the entirety of an application from one end to another
- Basic approach: AAA
  - **A**rrange: Set up the initial conditions of the test
  - **A**ct: Do / change something
  - **A**ssert: use assertions to guarantee that the correct thing happened (or not)
- Test Runner: a program that takes care of running our tests for us (e.g. when to run or re-run tests, how to display the results, how to find the tests, &c.)

Testing Solutions

- Unit and Small Integration Tests: Jest (https://jestjs.io/)
- Larger Integration and End-to-End: Cypress (https://www.cypress.io/)
- + some other smaller libraries to enhance each of them

Jest — Unit Tests

Jest

- Jest: test runner, also provides some tests
- `describe`, `it/test`, `expect`
- Basic form:

```
describe("The thing I'm testing", () => {
    test('should do a thing', () => {
        expect(something).toBe(the correct value)
    })
})
```

Running Jest

- Will automatically pick up `*.test.js` or `*.spec.js` files
- `npm run test` — all of them
- `npm run test PATH_TO_FILE` — will only run that one
- Watches files for changes and re-runs tests as appropriate
- By default, it will only re-run the failing ones
- Important assertions (all of these also take `.not.????` to negate them):
  - `toBe`: test equality
  - `toBeUndefined`: is it `===undefined`
  - `toMatchSnapshot`: does it match the stored snapshot? (You can pass in basically anything; the first time a snapshot is generated)
  - `toThrow`: does it throw the correct error? (needs additional function wrapper)
- (Full list: https://jestjs.io/docs/en/expect)

Example Program

Our test program is a simple React application that uses inputs to ask the user for two numbers and an operation, and displays the result.

Example — Testing a Single Function with Jest i

Let's test the functions in the `functions.js` file.

(You can checkout the `initial-function-test` of the `frontend-testing-tutorial`)

Example — Testing a Single Function with Jest ii

function.js

```
1  export const add = (a, b) => {
2      if (a === undefined || b === undefined) {
3          return undefined;
4      }
5      if (typeof a !== "number" || typeof b !== "number")
           {
6          throw new Error("You need to pass in a number!");
7      }
8      return a + b;
9  };
10
11 export const subtract = (a, b) => {
12     return a - b;
13 };
```

Example — Testing a Single Function with Jest iii

```
14
15 export const multiply = (a, b) => {
16     return a * b;
17 };
18
19 export const divide = (a, b) => {
20     return a / b;
21 };
```

Example — Testing a Single Function with Jest iv

function.test.js

```
1  import { add, subtract, multiply, divide } from "./
       functions";
2  describe("The 'add' function", () => {
3      it("Should correctly sum up two numbers", () => {
4          expect(add(2, 3)).toBe(5);
5          expect(add(2.1, 4)).toBe(6.1);
6          expect(add(-1, 1)).toBe(0);
7      });
8      it("Should throw an error if you pass in a non-
           number", () => {
9          expect(() => add("2", "3")).toThrow("You need to
               pass in a number!"); //note the additional
               function here!
10     });
```

Example — Testing a Single Function with Jest v

```
11      it("Should return `undefined` if you miss an
            argument", () => {
12          expect(add()).toBeUndefined();
13          expect(add(1)).toBeUndefined();
14      });
15  });
```

Testing DOM Elements with `react-testing-library`

- Use `react-testing-library` (https://testing-library.com/docs/react-testing-library/intro/)
- Philosophy: tests should resemble the way users interact with the site
- Uses the node virtual DOM to run applications
- Adds several commands to make assertions about the DOM
- `npm install --save-dev @testing-library/react`
- At the top of your testing file:

```
1 import { render, screen } from @testing-library/
      react
2 import @testing-library/jest-dom/extend-expect
```

Getting and Asserting on DOM Elements

- `get / find / query [All]`
- `by`
  - `Text`
  - `LabelText`
  - `Role`
  - `TestId`
- Adds some assertions:
  - `.toBeInTheDocument()`
  - `.toBeDisabled()`
  - `.toHaveAttribute()`

```
1 expect(screen.getByText(/flibbertigibbet/i)).
    toBeInTheDocument();
```

Firing Events

- Use `@testing-library/user-event` for events
- (`react-testing-library` has its own way to fire events (`fireEvent`), but this library is 'more realistic')
- (Some shortcomings, but generally what you want to use)
- `npm install --save-dev @testing-library/user-event`
- In your file: `import userEvent` from '@testing-library/user-event'
- Fairly intuitive interface:

```
1 userEvent.click(the thing to click)
2 userEvent.type(element, 'here{enter}is some{shift}
      text')
```

Example — Testing the DOM and Events i

Let's tests the DOM and some simple events!

1. Can we enter values into the inputs and see the result?
2. When we change the values, do we see a corresponsing change in the output?

Example — Testing the DOM and Events ii

App.js

```
1  import React, { useState, useEffect } from "react";
2  import { Form, Text, Select } from "informed";
3  import { add, subtract, multiply, divide } from "../
       functions";
4
5  function App() {
6     const [result, setResult] = useState();
7     const handleChange = (formState) => {
8        console.log("handleChange called");
9        const first = formState.values.first;
10       const op = formState.values.op;
11       const second = formState.values.second;
12
```

Example — Testing the DOM and Events iii

```
13        console.log("in handleChange", { first, op,
              second });
14
15        switch (op) {
16            case "+":
17                setResult(add(first, second));
18                break;
19            case "-":
20                setResult(subtract(first, second));
21                break;
22            case "*":
23                setResult(multiply(first, second));
24                break;
25            case "/":
26                setResult(divide(first, second));
```

Example — Testing the DOM and Events iv

```
27                break;
28            default:
29                console.log("Error: you shouldn't be here")
                      ;
30        }
31    };
32    return (
33        <>
34            {" "}
35            <h1>The Amazing Calculator!</h1>
36            <Form>
37                {({ formState }) => {
38                    return (
39                        <>
40                            <Text
```

Example — Testing the DOM and Events v

```
41                    aria-label="first"
42                    field="first"
43                    onChange={(e) => handleChange(
                          formState)}
44                />
45                <Select
46                    aria-label="operation"
47                    field="op"
48                    defaultValue="+"
49                    onChange={(e) => handleChange(
                          formState)}
50                >
51                    <option value="+">+</option>
52                    <option value="-">-</option>
53                    <option value="*">*</option>
```

Example — Testing the DOM and Events vi

```
54                          <option value="/"></option>
55                      </Select>
56                      <Text
57                          aria-label="second"
58                          field="second"
59                          onChange={(e) => handleChange(
                                formState)}
60                      />
61                      <span>{` = ${
62                          result === undefined ? "" :
                                result
63                      }`}</span>
64                  </>
65              );
66          }}
```

Example — Testing the DOM and Events vii

```
67              </Form>
68          </>
69      );
70  }
71
72  export default App;
```

Example — Testing the DOM and Events viii

App.test.js

```
1  import React from "react";
2  import { render, screen } from "@testing-library/react
       ";
3  import "@testing-library/jest-dom/extend-expect";
4  import userEvent from "@testing-library/user-event";
5
6  import App from "./App";
7
8  describe("The application", () => {
9     test("Should render correctly", () => {
10       render(<App />);
11
12       expect(screen.getByText(/the amazing calculator/i
           )).toBeInTheDocument();
```

Example — Testing the DOM and Events ix

```
13        // screen.debug();
14    });
15    test("Should be able to use the inputs to calculate
          that 2 * 3 = 6, and that changing the inputs
          changes the result", () => {
16        render(<App />);
17
18        expect(screen.getByText(/the amazing calculator/i
              )).toBeInTheDocument();
19
20        //check that there are two inputs, and get them
              both
21        expect(screen.getAllByRole("textbox").length).
              toBe(2);
22
```

Example — Testing the DOM and Events x

```
23          expect(
24              screen.getByRole("textbox", { name: "first" })
25          ).toBeInTheDocument();
26          const first = screen.getByRole("textbox", { name:
                "first" });
27          expect(
28              screen.getByRole("textbox", { name: "second"
                    })
29          ).toBeInTheDocument();
30          const second = screen.getByRole("textbox", { name
                : "second" });
31
32          //now enter the values
33          userEvent.type(first, "2");
34          expect(first.value).toBe("2");
```

Example — Testing the DOM and Events xi

```
35
36          userEvent.type(second, "3");
37          expect(second.value).toBe("3");
38
39          //now get the select
40          expect(
41              screen.getByRole("combobox", { name: "
                    operation" })
42          ).toBeInTheDocument();
43          const select = screen.getByRole("combobox", {
                name: "operation" });
44
45          userEvent.selectOptions(select, "*");
46          expect(select.value).toBe("*");
47
```

Example — Testing the DOM and Events xii

```
48        //check - we expect to see a 6 somewhere on the
              screen
49        expect(screen.getByText(/6/)).toBeInTheDocument()
              ;
50
51        //part 2 - if we change the value, does the value
               change?
52        userEvent.clear(first);
53        userEvent.type(first, "11");
54        expect(first.value).toBe("11");
55
56        userEvent.clear(second);
57        userEvent.type(second, "9");
58        expect(second.value).toBe("9");
59
```

Example — Testing the DOM and Events xiii

```
60          userEvent.selectOptions(select, "-");
61          expect(select.value).toBe("-");
62
63          //check the the original result disappeared and
                was replaced with the new one
64          expect(screen.queryByText(/6/)).not.
                toBeInTheDocument();
65          expect(screen.getByText(/2/)).toBeInTheDocument()
                ;
66      });
67  });
```

Cypress — Integration and End-to-End Testing

Cypress

- Full-featured test runner (basically does everything) (Homepage)
- Browser-based tests based on Mocha, Chai, &c. (Full list)
- `npm install cypress` (large download)
- Your server needs to be running for the tests to run
- Tests (`*.spec.js`) placed in the `cypress/integration` directory (from project root)
- Config: `cypress.json`
- Environment variables: `cypress.env.json`. They can also be stored in `cypress.json`...

cypress-testing-library

- The default Cypress tests often rely on things like attributes or other things that the user can't directly use
- `@testing-library/cypress` allows us to use the same DOM queries in Cypress
- Getting it to work requires a few more steps, but nothing too onerous (already installed for our project)

Cypress — Querying, Asserting, and Actions

- Chain off the base `cy` object
- Can use the `@testing-library` queries
- Cypress-native:
  - `cy.visit(url)` — probably the first one
  - `cy.contains(TEXT)` — get the element with contents TEXT
  - `cy.get(QUERY SELECTOR, options)` — use CSS selectors
  - `.should('eq', 6)` — the assertions are based on Mocha and Chai — ('exist') is also useful
  - `.click()`
  - `.type(WORDS)`

Example — Cypress Testing i

let's recreate our earlier Jest test of our system using Cypress!

1. Can we enter values into the inputs and see the result?
2. When we change the values, do we see a corresponsing change in the output?

Example — Cypress Testing ii

App.js

```
1  import React, { useState, useEffect } from "react";
2  import { Form, Text, Select } from "informed";
3  import { add, subtract, multiply, divide } from "../
       functions";
4
5  function App() {
6     const [result, setResult] = useState();
7     const handleChange = (formState) => {
8        console.log("handleChange called");
9        const first = formState.values.first;
10       const op = formState.values.op;
11       const second = formState.values.second;
12
```

Example — Cypress Testing iii

```
13        console.log("in handleChange", { first, op,
              second });
14
15        switch (op) {
16            case "+":
17                setResult(add(first, second));
18                break;
19            case "-":
20                setResult(subtract(first, second));
21                break;
22            case "*":
23                setResult(multiply(first, second));
24                break;
25            case "/":
26                setResult(divide(first, second));
```

Example — Cypress Testing iv

```
27                break;
28            default:
29                console.log("Error: you shouldn't be here")
                      ;
30        }
31    };
32    return (
33        <>
34            {" "}
35            <h1>The Amazing Calculator!</h1>
36            <Form>
37                {({ formState }) => {
38                    return (
39                        <>
40                            <Text
```

Example — Cypress Testing v

```
41                          aria-label="first"
42                          field="first"
43                          onChange={(e) => handleChange(
                                formState)}
44                      />
45                      <Select
46                          field="op"
47                          aria-label="operation"
48                          defaultValue="+"
49                          onChange={(e) => handleChange(
                                formState)}
50                      >
51                          <option value="+">+</option>
52                          <option value="-">-</option>
53                          <option value="*">*</option>
```

Example — Cypress Testing vi

```
54                                <option value="/"></option>
55                            </Select>
56                            <Text
57                                field="second"
58                                aria-label="second"
59                                onChange={(e) => handleChange(
                                      formState)}
60                            />
61                            <span>{` = ${
62                                result === undefined ? "" :
                                      result
63                            }`}</span>
64                        </>
65                    );
66                }}
```

Example — Cypress Testing vii

```
67              </Form>
68          </>
69      );
70  }
71
72  export default App;
```

Example — Cypress Testing viii

example.spec.js

```
1  describe("Our main application", () => {
2      it("Should take an input, display the result, and
           then display a new result when the inputs change
           ", () => {
3          //make the initial visit
4          cy.visit("http://localhost:3000/");
5          cy.findByText(/the amazing calculator/i).should("
               exist");
6
7          //now muck about with the options
8          cy.findByRole("textbox", { name: "first" })
9              .type(6)
10             .should("have.value", 6);
11         cy.findByRole("textbox", { name: "second" })
```

Example — Cypress Testing ix

```
12          .type(5)
13          .should("have.value", "5");
14      cy.findByRole("combobox", { name: "operation" })
15          .select("*")
16          .should("have.value", "*");
17      cy.findByText(/30/).should("be.visible");
18
19      //now change the options and see that the result
            changes as well
20      cy.findByRole("textbox", { name: "first" })
21          .clear()
22          .type(5)
23          .should("have.value", 5);
24      cy.findByRole("textbox", { name: "second" })
25          .clear()
```

Example — Cypress Testing x

```
26              .type(9)
27              .should("have.value", "9");
28          cy.findByRole("combobox", { name: "operation" })
29              .select("-")
30              .should("have.value", "-");
31          cy.findByText(/30/).should("not.exist");
32          cy.findByText(/-4/).should("be.visible");
33      });
34  });
```

THE END

!