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?
- (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. Let's test the functions in the `functions.js` file.

Example — Testing a Single Function with Jest i

word

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
```

Firing Events

- Use `@teseting-library/user-event` for events
- (`react-testing-library` has its own way to fire events, but this library is 'more realistic')
- `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

Cypress — Integration and End-to-End Testing

Cypress

- Cypress