Building Web Applications with React

Chapter 3: Unit Testing with Jest and React Testing Library

Chapter Objectives

In this chapter, we will:

- Explore automated testing with Jest
- Write tests using the React Testing Library
- Test a React component

Chapter Concepts

The Jest Framework

React Testing Library

Chapter Summary

Testing Applications

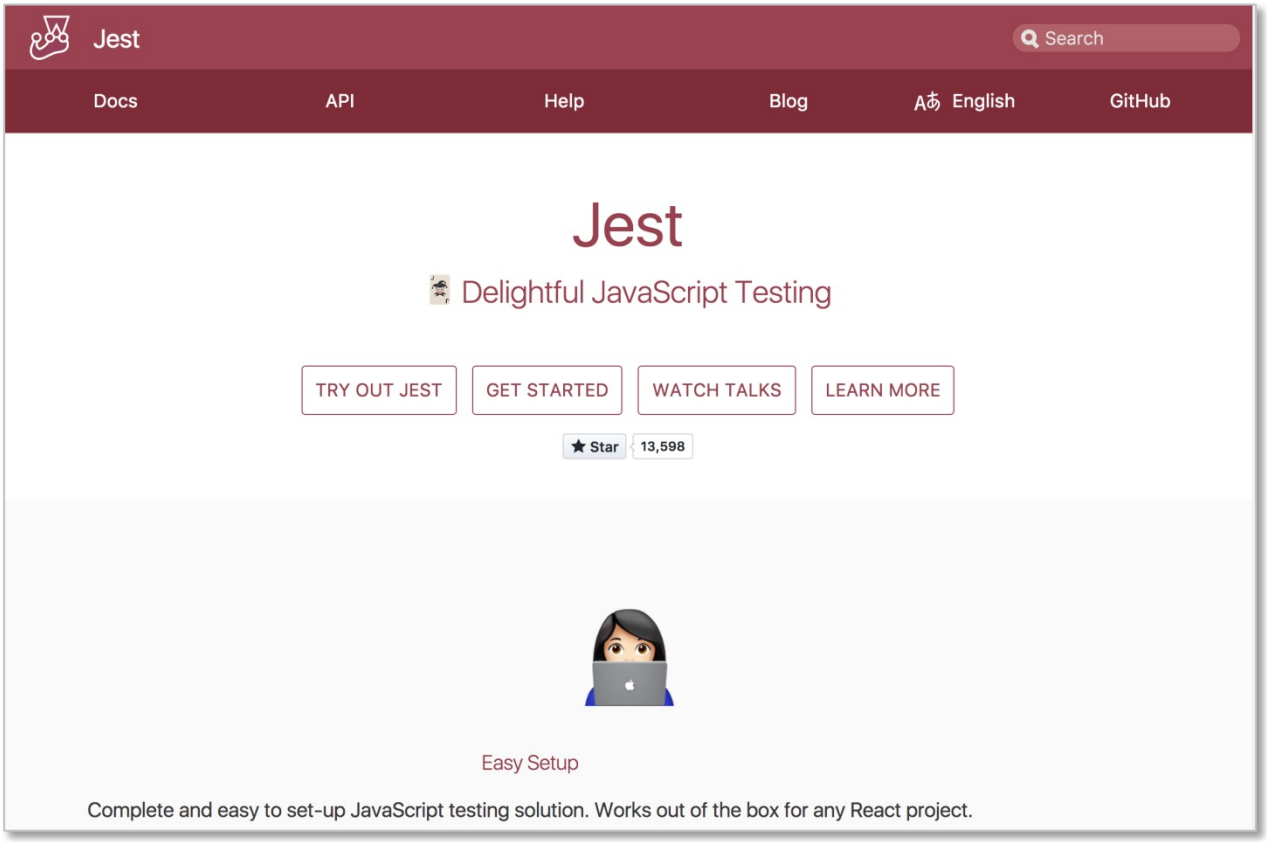
- User acceptance tests
  - Slow, expensive, and run infrequently
  - Not automated
- Integration tests
  - End-to-end test of system
    - Automated equivalent of user testing
    - Simulates user actions and records results
- Unit tests
  - Test individual components in isolation
  - Run frequently
  - Part of the development process
  - Many language-specific tools
    - Often with technology-specific extensions

Unit Testing React Components

- React supports automated unit testing
  - React components are just JavaScript
    - Even when using JSX
- Two things required for React unit tests
  - Testing Framework
  - React-specific utilities
- Testing framework
  - Facebook provides/uses Jest
  - Other frameworks are available
- Methods to simplify working with React components
  - React Testing Library is included with `create-react-app`
  - Focuses on testing component output

Jest

- Integrates with Node.js
- Allows JavaScript developers to define and run unit tests



Jest Methods

- Jest provides:
  - Object-mocking
  - Synchronous testing of asynchronous code
    - Including promises
  - Babel integration
  - Lightweight DOM implementation for command-line testing
- Main Jest methods
  - `describe()`
    - An optional container for tests
  - `test()`
    - An individual test
  - `expect()`
    - Checks if the test passed
      - `toEqual()`
      - `toBeGreaterThan()`
      - `toContain()`
      - More…

Jest Unit Test Code

Optional `describe()` contains tests

```
describe('Hello world test', () => {

    const a = "hello";


    test('should say hello', () => {

       expect(a).toEqual("hello");

    });

  });
```

`test()` is an individual test

`expect()` checks if the test has passed

Jest Test Files

- Best Practices:
  - Separate test file for each component
- Name test files `ComponentName.test.js`
  - Tests for `AddBookForm` inside `AddBookForm.test.js`
- This is the configuration used by `create-react-app`
  - Pre-created test for `App.js` is inside `App.test.js`
    - Both files are in the same folder

Chapter Concepts

The Jest Framework

React Testing Library

Chapter Summary

React Testing Library

- Included with `create-react-app`
- Focuses on testing DOM nodes returned by React component
- Test interacts with component as the user would
  - Entering text in inputs
  - Clicking buttons
  - Finding text in display elements
  - Locating buttons by their text
- Intention is to have test as closely resemble actual usage as possible

> The more your tests resemble the way your software is used, the more confidence they can give you

- Replaces Enzyme
  - Enzyme used to be recommended library for testing React
  - Many contributors to React Testing Library contributed to Enzyme

DOM Testing Library

- React Testing Library is an extension of DOM Testing Library
  - Re-exports all methods from DOM Testing Library
  - Adds some specific React extensions
- DOM testing library provides methods to find DOM nodes, e.g.:
  - `getByText`
  - `getByDisplayValue`
  - `getByLabelText`
- Also provides `fireEvent` method
  - Usual to also use more advanced simulations in `userEvent`
    - Must install companion library `@testing-library/user-event`

The `Render` Method

- Primary React extension in React Testing Library
- Renders a React component element into the DOM

```
test('it renders without crashing', () => {

    const { debug } = render(<App />);
    debug();

});
```

Renders `App`

Destructured return `debug` allows rendered HTML to be displayed in console when test runs

- By default, will render `all` children of the element
  - Jest mocks can be used to replace child elements
    - Not recommended in most circumstances

```
jest.mock('./components/books/BookList',
                              () => () => (<div>book list</div>));
```

Testing Rendered Content

`screen` provides the document body you can query against

```
import { render, screen } from '@testing-library/react';
import Book from './Book';


test('it should build the table row from strings passed as props',
 () => {
    const input = {
        title: 'Title 1',
        author: 'Author 1'
    };
    render(<table><tbody><Book author={input.author}
                  title={input.title} /></tbody></table>);

    const title = screen.getByText(input.title);

    expect(title).toBeInTheDocument();
});
```

Note the outer wrapping HTML to ensure valid DOM elements created

Search the `screen` for rendered text

Assert that the content is present

Testing Events

```
test("it should call the function when Add Review is clicked",
      () => {

   const onSubmit = jest.fn();

   const {getByText} = render(<ReviewForm addReview={onSubmit} />);


    fireEvent.click(getByText(/Add Review/i));


    expect(onSubmit).toBeCalled();

});
```

Replace `onSubmit` function with `jest` mock

Regex match string ignoring case

Assert that the DOM element called the correct method when the event fired

Exercise 3.1: Unit Testing React

- In this exercise, you will create automated unit tests for React components
- Please refer to the Exercise Manual

Chapter Concepts

The Jest Framework

React Testing Library

Chapter Summary

Chapter Summary

In this chapter, we have:

- Explored automated testing with Jest
- Written tests using the React Testing Library
- Tested a React component