Testing React Components

1. **React**: JavaScript UI library
2. **Jest**: JavaScript testing framework
3. **JSDOM**: DOM simulator for Node.js
4. **Testing Library**: DOM testing utility
5. **React Testing Library**: DOM testing utility for React components
6. **Storybook**: UI component rendering environment

React **JavaScript UI library**

- JavaScript library for rendering app interfaces
- Encourages the use of isolated components
- The line between an app and UI components often gets blurred

React is not necessarily:

- App framework

Jest **JavaScript testing framework**

- Great for writing unit tests of JavaScript functions and classes
- Encourages writing code as smaller isolated units
- Encourages a separation between app and library code
- Does not natively have access to the DOM since it runs on Node

Jest is not necessarily:

- A guarantee that your code works

JSDOM **DOM simulator for Node.js**

- Simulates the DOM and `window` global in Node
- Provides access to DOM API's like `document.querySelector()`
- Useful for things like parsing and transforming HTML

JSDOM is not necessarily:

- A testing tool

Testing Library **DOM testing utility**

- Abstraction of JSDOM under the hood
- Provides an excellent API for testing a simulated DOM
- Supplements a test runner like Jest

Testing Library is not necessarily:

- Test runner
- Test framework

React Testing Library

DOM testing utility for React components

- Testing Library, but for React components
- Renders isolated React components and makes them testable

Storybook **UI component rendering environment**

- Spins up an app for viewing isolated UI components
- Renders each component with any state or content
- Encourages a separation between app and components
- Provides a means for visually testing components on-demand

Storybook is not necessarily:

- Component library
- Test runner or test framework

Test a JavaScript function **With Jest**

1. Install dependencies
2. Setup and configure tooling
3. Write tests (Jest documentation)
4. Run tests

Install dependencies

```
nvm use 18

mkdir -p my-project
cd my-project

git init

npm init --yes
npm install -D jest
```

```
// package.json
{
  ...,
  "scripts": {
    "test": "jest --config=config/jest.config.js",
    "test:watch": "npm test -- --watch"
  }
}
```

Setup and configure tooling

```
// config/jest.config.js
module.exports = {
  bail: true,
  collectCoverage: true,
  collectCoverageFrom: ["src/**"],
  coverageDirectory: "coverage",
  coverageThreshold: {
    global: { statements: 100, branches: 100, functions: 100, lines: 100 },
  },
  rootDir: '../',
  testEnvironment: "node",
}
```

Write tests

- Describe the unit being tested
- Consider business requirements and user stories as criteria
- Setup conditions for the test, then make assertions or expectations
- Tests are not source code; put them in `/tests`

Run tests

- On-demand: `npm test`
- Watch for changes: `npm run test:watch`
- Git hooks: Husky pre-commit hook
- Continuous integration: Bitbucket and/or Jenkins pipeline

Test a React component

With Jest and Testing Library

1. Install dependencies
2. Setup and configure tooling
3. Write tests (Testing Library documentation)
4. Run tests

Install dependencies

- Babel for transpiling JSX to JS
- Testing Library for testing a simulated DOM
- Some plugins to connect everything

```
npm install -D react \
  react-dom \
  @babel/core \
  @babel/preset-react \
  babel-jest \
  @testing-library/react \
  @testing-library/jest-dom \
  jest-environment-jsdom
```

Setup and configure tooling

```
// config/jest.config.js
module.exports = {
  ...,
  coverageProvider: "babel",
  testEnvironment: "jsdom",
  transform: {
    "^.+.jsx?$": "babel-jest",
  },
}
```

```
// package.json
{
  ...,
  "babel": {
    "presets": ["@babel/preset-react"]
  }
}
```

View a React component story **With Storybook**

1. Install dependencies
2. Setup and configure tooling
3. Write stories (Storybook documentation)
4. Run Storybook

Install dependencies

```
npx storybook init
```

Replace `storybook` and `build-storybook` scripts and move configs files.

```
// package.json
{
  "scripts": {
    ...,
    "compile:storybook": "NODE_OPTIONS=--openssl-legacy-provider build-storybook --config-dir=config/storybook",
    "serve:storybook": "NODE_OPTIONS=--openssl-legacy-provider start-storybook --config-dir=config/storybook
--port=6060"
  }
}
```

Setup and configure tooling

```
// config/storybook/main.js
module.exports = {
  'stories': [
    '../../stories/**/*.stories.mdx',
    '../../stories/**/*.stories.@(js|jsx|ts|tsx)'
  ],
  'addons': [
    '@storybook/addon-links',
    '@storybook/addon-essentials',
    '@storybook/addon-interactions'
  ],
  'framework': '@storybook/react'
}
```

```
// config/storybook/preview.js
export const parameters = {
  actions: { argTypesRegex: '^on[A-Z].*' },
  controls: {
    matchers: {
      color: /(background|color)$/i,
      date: /Date$/,
    },
  },
}
```

Write stories

- List the stories for each component
- Consider business requirements and user stories as criteria
- Setup a template and instantiate with different args for each story
- Stories are not source code; put them in `/stories`
  - Move `.storybook` to `config/storybook`
  - In `config/storybook/main.js` update `stories` paths: `../src` to `../../stories`

Run Storybook

- Compile on-demand: `npm run compile:storybook`
- Serve and watch for changes: `npm run serve:storybook`
- Git hooks: Husky pre-commit hook
- Continuous integration: Bitbucket and/or Jenkins pipeline

Checkout the repo

github.com/paulshryock/testing-react-components