React Testing with Jest 1-day course

# General delivery plan

This section explains generally how the course can be delivered over four days.

|  |  |
| --- | --- |
| **DAY 1** | |
| 09:30 | Agenda + Intro to React Testing with Jest |
| 10:00 | How to set up a test environment |
| 10:30 | *Break* |
| 10:45 | Jest |
| 10:55 | The What and How of Testing in React |
| 11:10 | Snapshot Testing |
| 11:45 | *Break* |
| 12:00 | Mocking Components for Testing |
| 12:10 | Testing State and Event Interactions |
| 12:15 | Mocking Functions |
| 12:30 | *Lunch* |
| 13:30 | Testing Components Asynchronously |
| 14:00 | Testing Components with Routing |
| 14:15 | Custom Hooks |
| 14:30 | *Break* |
| 14:45 | Testing Exercise |
| 15:30 | *Break* |
| 15:45 | Testing Exercise (continued) |
| 16:30 | *Finish* |

# Notes on delivering this course

This course is designed to be as practical as possible. The delegates should be familiar with Jest, but perhaps not new techniques of using it with React. The course should be completed via demonstrations and explanations throughout.

In each of the folders, there are examples of working versions of each of the topics that have been covered.

As the trainer delivering the content, it is completely at your discretion as to how you demonstrate these.

## Intro to React Testing with Jest

* Testing is crucial in application development.
  + Emphasize the role of testing in maintaining code quality.
  + Highlight testing as a practice to reduce bugs and improve user experience.
* Important to understand React testing tools, including Jest.
  + Introduce Jest as the primary testing framework for React applications.
  + Mention other useful tools and libraries that complement Jest, like Enzyme or React Testing Library.
* Discussion on unit testing components, including snapshots, dumb testing, and event testing.
  + Define unit testing and its importance in testing individual components in isolation.
  + Explain snapshots for capturing component renders over time.
  + Introduce dumb (or shallow) testing for components without external dependencies.
  + Detail event testing to simulate user interactions.
* Explanation of Jest test file structure, suites specs, setup and teardown.
  + Describe the organization of tests in files and directories.
  + Discuss the use of describe blocks for grouping related tests.
  + Explain setup (beforeEach) and teardown (afterEach) methods for preparing and cleaning up tests.
* Overview of making assertions with in-built matchers.
  + Introduce the concept of assertions to verify test outcomes.
  + List common matchers provided by Jest, like toEqual, toBe, and toHaveBeenCalled.
* Details on mocking and spying for testing functionality and component unitization.
  + Explain mocking to isolate components by simulating behaviour of dependencies.
  + Discuss spying to track calls to functions and methods.
* Highlight on the importance of test reporting, including code coverage.
  + Define code coverage and its relevance in understanding test effectiveness.
  + Show how to generate and interpret code coverage reports.
* Focus on unit testing without covering integration or end-to-end testing.
  + Clarify the scope of unit tests versus integration and E2E tests.
  + Emphasize the specific focus on React component logic and behaviour.
* Advice on what to test and what not to test, with reasons.
  + Suggest prioritizing business logic, critical paths, and user interactions.
  + Recommend avoiding tests that replicate the framework's internal functionality.

## How to set up the Test Environment

* If you're starting a new project, you can create a Vite-based React project by running.
* Change my-react-app to your desired project name. Then, navigate into your project directory.
* Jest does not come pre-configured with Vite, so you'll need to install Jest along with some additional dependencies to support React and Babel.
* Create a Babel configuration file .babelrc in your project's root directory. Add the React and modern JavaScript presets to the file.
* Set up Jest by creating a jest.config.cjs file in the root of your project with basic configuration.
* Create a jest-setup.js file (referenced in the Jest config) in your project root to include any global setup for Jest, such as importing jest-dom for additional matchers.
* Create a fileMock.js file inside a \_\_mocks\_\_ directory at the root of your project.
* Modify the scripts section of your package.json to add a command for running tests.
* Write your first test! For a component like src/App.jsx, create a test file src/App.test.js and write a simple test case.
* Execute your tests from the command line.

## Jest

* Jest described as a simple, delightful JavaScript testing framework.
  + Highlight Jest's simplicity and ease of use for beginners.
  + Mention Jest's ability to work with various JavaScript projects, enhancing its versatility.
* Compatible with various JavaScript projects and frameworks.
  + List examples of projects and frameworks Jest is compatible with, like Node.js and Vue.
* Emphasizes its independence from other frameworks and the simplicity of syntax.
  + Discuss the advantage of Jest being framework-agnostic.
  + Highlight Jest's straightforward syntax for writing tests.
* Explains the use of suites to group tests and the describe and it/test functions for defining tests.
  + Detail how describe blocks group related tests for better organization.
  + Compare the it and test functions for writing individual test cases.
* Discussion on using expect and matchers for assertions.
  + Dive deeper into the expect function for setting up test assertions.
  + Provide examples of using matchers to validate test outcomes.
* Introduction to custom matchers for specialized needs.
  + Explain how and why to create custom matchers for specific test cases.
  + Provide a simple example of a custom matcher.
* Mention of Jest's code coverage tool included in project setups using create-react-app.
  + Guide on enabling and interpreting Jest's built-in code coverage reports.
  + Discuss setting coverage thresholds to maintain high code quality.

## The What and How of Testing in React

* Focus on testing parts of applications that render data.
  + Identify components that directly interact with data for priority testing.
  + Consider testing components for correct data handling and rendering.
* Different strategies for testing React apps, including snapshot testing and testing user interactions.
  + Snapshot Testing: Ensure UI does not change unexpectedly. Ideal for components with stable output.
  + User Interactions: Simulate clicks, inputs, and other user actions to test components' responses.
* Importance of mocking for testing components with dependencies.
  + Utilize mocking to simulate external dependencies, ensuring components can be tested in isolation.
  + Discuss strategies for mocking APIs, components, and functions.
* Discussion on testing hooks and routing within React apps.
  + Hooks: Focus on custom hooks that manage state or side effects. Test hooks for expected behaviour.
  + Routing: Test navigation and route-based rendering without focusing on the router's internal functionality.
* Emphasis on writing tests that resemble how software is used for increased confidence.
  + Advocate for testing scenarios that closely mimic real-user interactions.
  + Highlight the importance of accessibility and usability tests.

## Snapshot Testing

* Snapshot testing ensures UI consistency.
  + Discuss the benefits of snapshot testing for catching unintended UI changes.
  + Provide guidelines for when to update snapshots versus when to investigate changes further.
* Use of react-test-renderer for rendering components as JavaScript objects.
  + Detail the process of using react-test-renderer for creating snapshots.
  + Mention alternatives and their use cases, such as enzyme-to-json for Enzyme integration.
* Process of creating, comparing, and updating snapshots.
  + Step-by-step guide on creating the first snapshot and subsequent comparisons.
  + Discuss strategies for reviewing and updating snapshots after intentional changes.

## Testing Components with Props

* Testing for the rendering of props, excluding default props and prop types.
  + Highlight the importance of testing dynamic props for correct rendering.
  + Discuss strategies for testing components with various prop combinations.
* Use of the test renderer object to find elements and assert based on props.
  + Demonstrate how to use selectors to find elements rendered by props.
  + Provide examples of assertions based on prop values and types.

## Mocking Components for Testing

* Unit testing challenges with components rendering other components.
  + Discuss the complexity of testing components with nested dependencies.
  + Introduce the concept of shallow rendering versus full DOM rendering for isolating components.
* Use of Jest's mock function to simulate components for isolated testing.
  + Provide examples of how to mock child components, services, and modules.
  + Discuss the use of jest.mock and practical examples of its application.

## Testing State and Event Interactions

* Importance of testing state changes and event-triggered re-renders.
  + Explain the need to test stateful components for expected behaviour upon state changes.
  + Discuss how to simulate events using tools like @testing-library/user-event.
* Use of the act function for simulating user actions in tests.
  + Describe how act ensures test fidelity by wrapping updates and assertions.
  + Provide examples of using act to test event handlers and state updates.

## Mocking Functions

* Mocking functions passed as props for testing components independently.
  + Discuss how to isolate component behaviour by mocking callback functions passed as props.
  + Provide examples of asserting that mocked functions are called with the correct arguments.
* Use of Jest's functionality to mock function calls and arguments.
  + Explore Jest's API for spying on and asserting function calls and interactions.
  + Detail the use of .toHaveBeenCalled and .toHaveBeenCalledWith matchers.

## Testing Components Asynchronously

* Challenges of testing components with asynchronous data fetching.
  + Address common issues and strategies for testing asynchronous behaviour.
  + Discuss using async/await in tests to handle promises and async functions.
* Use of mocking for asynchronous calls, e.g., with axios.
  + Demonstrate how to mock HTTP requests using jest.mock or libraries like nock.
  + Provide examples of testing loading states, successful responses, and error handling.

## Testing Components with Routing

* Routing components tested by wrapping in a MemoryRouter for isolation.
  + Explain the use of MemoryRouter for testing components that interact with routing.
  + Discuss how to test navigational changes without affecting the global router state.
* Explanation of testing components that use router components without needing mocks or stubs.
  + Detail approaches for testing components that include <Link>, <Route>, and other routing components.
  + Discuss the importance of context providers in testing routing-related behaviour.

## Testing Custom Hooks

* Testing custom hooks directly using the React hooks testing library.
  + Introduce the React hooks testing library for direct hook testing.
  + Provide examples of testing state, effects, and custom logic within hooks.
* Example of testing a custom hook with state and effect updates.
  + Walk through a practical example of testing a hook's internal state and side effects.
  + Discuss how to use renderHook and act from the React hooks testing library.

## Testing Exercise

Unit test the QA Estate Agents website using techniques covered during training, this will take the remainder of the afternoon and it is unlikely that they will finish.   
  
This is more of an opportunity for them to try out different techniques that have been covered and ask questions.

Link to the QA Estate Agents repo:  
<https://github.com/AndySmithQA/QAEstate>