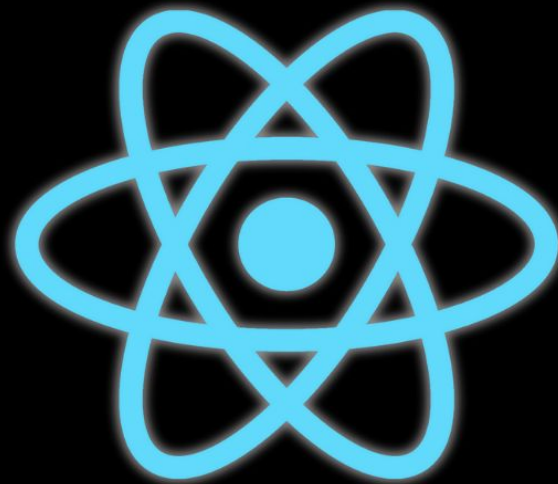


# React

# Unit testing

Master React Testing Library &  
Jest



2025 Edition

Jumana Salhab

# About This Book

Unit testing is often overlooked, yet it's essential for building **reliable and maintainable React applications**. This book is a **practical guide** to mastering **React Testing Library and Jest**, helping you write tests with confidence.

By the end of this book, you'll know **what to test, how to test, and why testing matters**. You'll learn to prevent regressions, improve code stability, and integrate testing seamlessly into your development workflow.

Whether you're a **beginner** or an **experienced React developer**, this book will give you **the skills and real-world examples** to level up your testing game.

## Copyright Notice

**Copyright © 2025 Jumana Salhab**

All rights reserved.

No part of this book may be copied, reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of the author.

This book is licensed for personal use only. Unauthorized distribution, resale, or reproduction is strictly prohibited.

# Table of contents

## [Introduction](#)

[Accessing the Code Examples](#)

## [Chapter 1: Introduction to React Testing Library and Jest](#)

[Types of Testing](#)

[Why is Testing Important?](#)

[What to Test in React Applications](#)

[What NOT to Test in React Applications](#)

[The Pyramid of Testing in React.js](#)

[Overview of the Testing Pyramid](#)

[Explanation of the Screenshot](#)

[Breakdown of Each Layer in the Testing Pyramid](#)

[Balancing the Testing Pyramid in React](#)

[Unit Testing vs Integration Testing vs E2E](#)

## [Chapter 2: Setting Up the Development Environment](#)

[1. Installing React Testing Library, Jest, and TypeScript](#)

[1.1 If You Are Using Create React App \(CRA\) \(Deprecated in React 19\)](#)

[1.2 If You Are Using Vite or Custom Setups](#)

[1.3 Explanation of Each Package](#)

[2. Configuring Jest for Non-CRA Projects](#)

[3. Running Tests for the First Time](#)

[4. Running a Specific Test File](#)

## [Chapter 3: Understanding the testing Fundamentals](#)

[How to write meaningful tests for a React component](#)

[Steps to Write Unit Tests in React](#)

[The Render Function in React Testing Library](#)

[What is the Render Function?](#)

[Why is the Render Function Important?](#)

[Best Practices for Using the Render Function](#)

[Common Mistakes to Avoid](#)

[Choosing Appropriate Selectors and Queries](#)

[Understanding Query Methods](#)

[Choosing the Best Query for Each Case](#)

[Common Mistakes to Avoid](#)

## [Arrange, Act, Assert pattern](#)

[What is the Arrange, Act, Assert Pattern?](#)

[Why use the AAA Pattern?](#)

[Applying the AAA Pattern in React Testing](#)

## [Understanding test suites and test cases](#)

[What is a Test Suite?](#)

[What is a Test Case?](#)

[Structuring Test Suites and Test Cases](#)

[Advanced: Nested Test Suites](#)

[Setting Up and Tearing Down Test Suites](#)

[Common Mistakes to Avoid](#)

## [Understanding Jest matchers](#)

[What Are Jest Matchers?](#)

[Common Jest Matchers](#)

## [Chapter 4: Testing React Router in React with RTL and Jest](#)

### [Mocking BrowserRouter in Tests](#)

[How to Mock BrowserRouter in Tests](#)

[Testing Components That Use useNavigate](#)

[Mocking Navigation in Tests](#)

[Testing Initial Route Rendering](#)

### [Testing Navigation \(useNavigate, <Link>, Redirects\)](#)

[Testing <Link> Navigation](#)

[Testing Navigation Using useNavigate\(\)](#)

[Testing Redirects \(<Navigate> and useNavigate\)](#)

### [Testing Route Parameters \(useParams\)](#)

[Testing Components That Use useParams](#)

[Mocking useParams in Tests](#)

[Handling Missing or Invalid Parameters](#)

[Testing Components That Fetch Data Using useParams](#)

### [Simulating Route Changes in Tests](#)

[Changing Routes Using MemoryRouter with initialEntries](#)

[Simulating Route Changes Using history.push\(\)](#)

[Testing Components That React to Route Changes \(useEffect\)](#)

### [Handling Protected Routes \(Authentication & Redirects\)](#)

[Basic Protected Route Setup and Testing](#)

[Testing Authenticated Access to Protected Routes](#)

[Testing Authentication with Context \(AuthProvider\)](#)

[Testing Role-Based Protected Routes](#)

[Testing Nested Routes & Layout Components](#)

[Understanding Nested Routes with Layout Components](#)

[Testing That the Layout Renders Correctly](#)

[Testing Nested Route Rendering](#)

[Testing Navigation Between Nested Routes](#)

[Handling Nested Route Redirects \(Default Child Route\)](#)

[Testing 404 & Not Found Routes](#)

[Setting Up a 404 Route in React Router](#)

[Testing 404 Route Handling](#)

[Ensuring Valid Routes Do NOT Show 404](#)

[Testing Redirects from Invalid Routes](#)

[Testing 404 Pages with Navigation Links](#)

[Debugging & Common Pitfalls in React Router Testing](#)

[Forgetting to Wrap Components with MemoryRouter](#)

[Testing Navigation Without Using `userEvent.click\(\)`](#)

[Not Using `initialEntries` When Testing Route Rendering](#)

[Forgetting to Mock `useNavigate\(\)` in Tests](#)

[Using `getByText` Instead of `queryByText` for Non-Existent Elements](#)

[Not Mocking API Calls When Testing Routes That Fetch Data](#)

[Infinite Redirects in Navigate Components](#)

[Not Handling Role-Based Redirects Properly](#)

[Chapter 5: Mocking Functions](#)

[What is Mocking?](#)

[How Mocking helps in Testing ?](#)

[When should you use Mocking?](#)

[Benefits of Mocking](#)

[Types of Mocking in Jest](#)

[Function Mocks \(`jest.fn\(\)`\)](#)

[Spying on Functions \(`jest.spyOn\(\)`\)](#)

[Module Mocks \(`jest.mock\(\)`\)](#)

[Partial Mocking \(`jest.requireActual\(\)`\)](#)

[Mocking Return Values \(`mockReturnValue`, `mockReturnValueOnce`\)](#)

[Mocking Implementations \(`mockImplementation`, `mockImplementationOnce`\)](#)

[Mocking Rejected Values \(mockRejectedValue\)](#)

[When to use each Mock Function in Jest ?](#)

[beforeEach vs AfterEach](#)

[Understanding beforeEach and afterEach](#)

[Why use beforeEach?](#)

[Why use afterEach?](#)

[When to use beforeEach and afterEach](#)

[Common Mistakes and Best Practices](#)

[beforeAll vs AfterAll](#)

[Understanding beforeAll and afterAll](#)

[Why use afterAll?](#)

[When to Use beforeAll and afterAll?](#)

[Common Mistakes and Best Practices](#)

[Chapter 6: Testing Async Operations](#)

[Testing async functions and promises](#)

[Understanding Async Functions and Promises in Testing](#)

[Best Practices for Testing Async Functions and Promises](#)

[Using waitFor](#)

[What is waitFor?](#)

[Why use waitFor?](#)

[Real-World Use Cases and Examples](#)

[Common Mistakes and How to Avoid Them](#)

[Best Practices for Using waitFor](#)

[Using waitForElementToBeRemoved](#)

[What is waitForElementToBeRemoved?](#)

[Why use waitForElementToBeRemoved?](#)

[Real-World Use Cases and Examples](#)

[Common Mistakes and How to Avoid Them](#)

[Best Practices for Using waitForElementToBeRemoved](#)

[Testing Promises and Async/Await Functions](#)

[Understanding Promises and Async/Await in Testing](#)

[Testing a Standalone Async Function](#)

[Testing an Async Function Inside a React Component](#)

[Best Practices for Testing Promises and Async/Await](#)

[Common Mistakes and How to Avoid Them](#)

[Mocking API calls and testing component behavior](#)

[Why Mock API Calls?](#)

[Ways to Mock API Calls in Tests](#)

[Mocking API Calls in an Async Function](#)

[Mocking API Calls Inside a React Component](#)

[Using msw \(Mock Service Worker\) for More Realistic API Mocking](#)

[Best Practices for Mocking API Calls](#)

[Testing Async State Updates and Side Effects](#)

[Testing Async State Updates in Components](#)

[Testing State Updates Due to User Interaction](#)

[Testing Side Effects With Timers](#)

[Testing Side Effects With Subscriptions and Cleanup](#)

[Testing State Updates With External Events](#)

[Using Jest's built-in async utilities](#)

[Using resolves and rejects to Test Promises](#)

[Why use resolves and rejects?](#)

[Controlling Timers with jest.useFakeTimers\(\)](#)

[Fast-Forwarding Time with jest.advanceTimersByTime\(\)](#)

[Extending Test Timeout with jest.setTimeout\(\)](#)

[Retrying Flaky Async Tests with jest.retryTimes\(\)](#)

[Mocking External Dependencies and API Responses in Jest](#)

[Mocking API Calls Using Jest Mocks \(jest.mock\(\)\)](#)

[Mocking fetch for Network Requests](#)

[Mocking Third-Party Libraries](#)

[Mocking Browser APIs](#)

[Mocking WebSockets or Event Listeners](#)

[Handling async errors and timeouts in tests](#)

[Testing Error Handling in Async Functions](#)

[Testing Component Behavior When an API Fails](#)

[Handling Timeouts in Async Tests](#)

[Simulating Network Errors in Tests](#)

[Testing Retry Logic for API Calls](#)

[Chapter 7: Testing Hooks and Custom React Hooks](#)

[Writing tests for custom React hooks](#)

[Testing Hooks with Side Effects](#)

[Testing Hooks with Async State Updates](#)

[Testing Hooks That Change Based on Props](#)

## [Understanding renderHook\(\) in React Testing Library](#)

[Testing Hooks with Dependencies](#)

[Testing Hooks That Use Side Effects \(useEffect\)](#)

[Testing Hooks with Props \(render\(\)\)](#)

[Testing Async Hooks \(waitForNextUpdate\(\)\)](#)

## [Understanding act\(\) in React Testing Library](#)

[Why Do We Need act\(\)?](#)

[Using act\(\) in Custom Hook Tests](#)

[Using act\(\) in Component Tests](#)

[Using act\(\) for Async State Updates](#)

[Common Mistakes When Using act\(\)](#)

[When not to use act\(\)](#)

## [Understanding rerender\(\) in React Testing Library](#)

[How rerender\(\) Works](#)

[Using rerender\(\) to Test Hooks with Dependencies](#)

[Using rerender\(\) to Test Components with Changing Props](#)

[Testing Hooks That Update Based on Prop Changes](#)

[Common Mistakes and How to Avoid Them](#)

## [Understanding waitForNextUpdate\(\) in React Testing Library](#)

[How waitForNextUpdate\(\) Works](#)

[Basic Syntax](#)

[Using waitForNextUpdate\(\) to Test Async State Updates](#)

[Using waitForNextUpdate\(\) with SetTimeout or Intervals](#)

[Testing a Hook That Fetches Data on Interval](#)

[Common Mistakes and How to Avoid Them](#)

## [Understanding toBe\(\) vs toEqual\(\) in Jest](#)

[Understanding toBe\(\)](#)

[Understanding toEqual\(\)](#)

[When to Use toBe\(\) vs toEqual\(\) in Tests](#)

[Real-World Example: Testing a Custom Hook](#)

[Common Mistakes and How to Fix Them](#)

## [Testing Hooks That Use Context \(useContext\)](#)

[Understanding useContext in Hooks](#)

[Testing Hooks That Depend on Context Updates](#)

[Testing a Component That Uses Context](#)

[Common Mistakes and How to Fix Them](#)



[When to Use useContext Testing Approaches](#)

[Chapter 8: Testing React Query with Jest and React Testing Library](#)

[Why Testing React Query is Different from Regular State Management](#)

[Mocking React Query API Calls in Tests](#)

[Approach 1: Mocking API Calls with Mock Service Worker \(MSW\)](#)

[Approach 2: Using Jest Mocks Instead of MSW](#)

[Testing React Query in a .ts File not .tsx](#)

[Why Does This Error Occur?](#)

[How to Fix This Issue](#)

[Complete Test Setup for a React Query Hook in a .ts File](#)

[Testing Query States \(Loading, Error, Success, Empty States\)](#)

[Understanding React Query States](#)

[Testing the Loading State](#)

[Testing the Success State](#)

[Testing the Error State](#)

[Testing the Empty State](#)

[Best Practices for Testing React Query States](#)

[Verifying Cache Behavior and Background Refetching](#)

[How React Query Caches Data](#)

[Testing Cached Data Persistence](#)

[Testing Background Refetching](#)

[Testing Cache Invalidation and Query Refetching](#)

[Best Practices for Testing Cache and Background Refetching](#)

[- Use QueryClient to configure caching behavior in tests.](#)

[- Always reset the cache before each test to prevent stale data leaks.](#)

[- Use await waitFor\(\) when testing background updates.](#)

[- Mock multiple API responses to verify data changes over time.](#)

[- Use invalidateQueries\(\) to trigger refetches in controlled tests.](#)

[Testing Mutations and Query Invalidation](#)

[Understanding Mutations in React Query](#)

[Testing Mutations \(Adding Data\)](#)

[Testing Mutations \(Deleting Data\)](#)

[Testing Error Handling in Mutations](#)

[Chapter 9: Testing Redux in React with Jest and React Testing Library](#)

[Why Test Redux?](#)

[Testing Redux Reducers](#)

[Why Test Redux Reducers?](#)

[Setting Up Tests for Redux Reducers](#)

[Writing Unit Tests for Reducers](#)

[Best Practices for Testing Reducers](#)

[Testing Redux Actions and Thunks](#)

[Why Test Redux Actions and Thunks?](#)

[Testing Synchronous Redux Actions](#)

[1. Define Test Suite](#)

[2. Test Case: Creating an Add Todo Action](#)

[Testing Asynchronous Redux Thunks](#)

[Best Practices for Testing Redux Actions and Thunks](#)

[Common Mistakes to Avoid](#)

[Testing Redux Store Integration in React Components.](#)

[Why Test Redux Store Integration?](#)

[Setting Up a Redux-Connected Component for Testing](#)

[How to Test Redux-Connected Components](#)

[Best Practices for Testing Redux-Connected Components](#)

[Common Mistakes to Avoid](#)

[Mocking API Calls in Redux Async Thunks](#)

[Why Mock API Calls in Redux Thunks?](#)

[Example: Redux Async Thunk for Fetching Todos](#)

[Mocking API Calls in Redux Async Thunk Tests](#)

[Best Practices for Testing Redux Async Thunks](#)

[Common Mistakes to Avoid](#)

[Using Redux Testing Utilities](#)

[Why use Redux Testing Utilities?](#)

[Setting Up a Test Redux Store](#)

[Using Redux Test Store in Tests](#)

[Testing Async Thunks with Redux Test Store](#)

[Best Practices for Using Redux Testing Utilities](#)

[Common Mistakes to Avoid](#)

[Chapter 10: Best Practices and Tips for Effective Testing](#)

[Choosing the Right Query in React Testing Library \(RTL\)](#)

[Preferred Queries \(Use These First\)](#)

[Secondary Queries \(Use If No Labels or Roles Exist\)](#)

[Fallback Queries \(Use Only When Necessary\)](#)

[Summary: Best Practices for Selecting the Right Query](#)

[Writing Clean and Maintainable Tests](#)

- [Keep Tests Focused and Concise](#)
- [Follow the Given-When-Then Structure](#)
- [Avoid Test Duplication](#)
- [Use Meaningful Assertions](#)
- [Clean Up After Each Test](#)

[Naming Conventions and Descriptions for Tests](#)

- [General Naming Conventions](#)
- [Structuring Test Descriptions](#)
- [Recommended Naming Pattern for it Blocks](#)
- [Use Meaningful test or it Descriptions](#)
- [Naming Convention for Test Variables and Mocks](#)
- [Example of a Well-Structured Test Suite](#)
- [Common Mistakes to Avoid](#)

[Organizing Test Files and Test Suites](#)

- [General Folder Structure for Tests](#)
- [Grouping Tests Within a File Using describe](#)
- [Organizing Tests by Type](#)
- [Skipping and Running Specific Tests](#)

[The “it.each” Command in Jest](#)

- [What is it.each?](#)
- [Basic Syntax of it.each](#)
- [Using it.each for Component Testing](#)
- [Using it.each with Multiple Assertions](#)
- [Using it.each for API Calls and Error Handling](#)
- [Using it.each with Objects for Better Readability](#)
- [Common Mistakes and How to Avoid Them](#)

[Debugging and Troubleshooting Tests in Jest & React Testing Library](#)

- [Running Tests in Watch Mode](#)
- [Using only and skip to Isolate Tests](#)
- [Using console.log and debug\(\) for Debugging](#)
- [Using React Testing Library’s screen.debug\(\)](#)
- [Handling Asynchronous Test Failures](#)
- [Checking for Incorrect Queries](#)
- [Running Tests with Verbose Output](#)

[Debugging Mock Functions](#)

[Flaky Test Troubleshooting Checklist](#)

[Chapter 11: Enforcing Best Practices with ESLint Rules](#)

[Why Linting Matters in Testing](#)

[Essential ESLint Rules for Writing Better Tests](#)

[Configuring ESLint for Jest and React Testing Library](#)

[Common Mistakes ESLint Can Catch in Test Files](#)

[Automating Test Best Practices with ESLint](#)

[Chapter 12: Test Coverage and Reporting](#)

[What Is Test Coverage, and Why Does It Matter?](#)

[Understanding Coverage Metrics](#)

[Generating a Coverage Report Using Jest](#)

[Improving Test Coverage While Avoiding Unnecessary Tests](#)

[Using Coverage Reports to Identify Untested Code](#)

[Access the Code Examples](#)

# Introduction

When I first started learning **unit testing**, I struggled to find **comprehensive resources** that covered all aspects of testing in a **clear and practical** way. Most tutorials were either too **theoretical** or lacked **fully working examples**. At the beginning of my career, I worked at companies where **unit testing wasn't a priority at all**. Writing tests felt like an **optional** or **unnecessary** step—until I joined a company that took **unit testing seriously**.

At first, it was **overwhelming**. I had **so many questions**:

- What should I test?
- Should I test everything?
- How does testing actually work?

I quickly realized that **unit testing is more than just a "nice-to-have"—it's a crucial part of building reliable applications**. But without the right guidance, **getting started can feel intimidating**. That's exactly why I wrote this book—to give developers like you a **clear, structured, and practical approach to unit testing in React**.

## Why This Book?

Software development has evolved rapidly, making **reliability, scalability, and maintainability** more important than ever. React has become one of the most popular libraries for **building interactive user interfaces**, but as applications grow in complexity, so does the risk of **bugs, regressions, and unexpected failures**. **Unit testing** plays a vital role in preventing these issues and ensuring that your application remains **stable and maintainable** over time.

This book is designed to be a **practical, hands-on guide to unit testing in React**. Whether you're **just getting started** or looking to **level up your skills**, you'll find everything you need to confidently write **effective** and **meaningful** tests. We'll focus on **React Testing Library** and **Jest**, covering everything from **fundamental concepts** to **advanced testing strategies**—all with real-world examples and **fully working projects** that you can use and even add to your portfolio.

## Why Testing Matters

Testing is often overlooked or treated as an afterthought in software development. Many developers hesitate to write tests, perceiving them as **time-consuming** or **unnecessary for small projects**. However, investing in tests early in the development cycle **saves time**, prevents costly **production bugs**, and increases **developer confidence** when making changes.

In this book, we'll break down why testing is **crucial** for React applications and how it contributes to:

- **Code stability and reliability** – Catching errors before they reach production.
- **Maintainability** – Making refactoring easier without the fear of breaking functionality.
- **Developer confidence** – Deploying new features with fewer surprises.
- **Faster debugging** – Pinpointing issues quickly when things go wrong.

By learning to write **meaningful and effective tests**, you'll gain the ability to build **robust** and **maintainable** React applications with confidence.

## Who This Book Is For

This book is designed for **React developers of all levels** who want to improve their testing skills:

- **Beginners** – Learning how to write tests for the first time.
- **Experienced Developers** – Refining testing strategies for professional projects.

## How to Use This Book

Each chapter includes:

- **Clear explanations** – Breaking down concepts in a beginner-friendly way.
- **Practical examples** – Real-world use cases to demonstrate testing techniques.
- **Downloadable example projects** – Fully functional applications with unit tests, available for download via a dedicated **Notion page**. You can explore, modify, and use them to practice writing tests and even add them to your portfolio.

This is **not just a theoretical book**—it's a **practical roadmap** to mastering **unit testing in React**. Whether you're building personal projects or working on large-scale applications, the skills you gain here will help you **write high-quality, bug-free, and maintainable React code**.

Let's dive in and make testing an **integral part** of your development workflow!

## Accessing the Code Examples

To help you put everything into practice, I've included **fully working projects with unit tests** that demonstrate the concepts covered in this book. These projects are available for **download and hands-on experimentation**.

You can find them in the **final chapter**, where I provide a **detailed guide on accessing and running them**. The examples are structured to give you a hands-on approach, allowing you to modify the tests and experiment with different scenarios.

 **Make sure to check the last chapter to download and explore the example projects!**