📘 Chapter 1 — Introduction to Testing & React Testing Library

---

1️⃣ What is React Testing Library (RTL)?

React Testing Library is not just a library — it's a philosophy.

When we say *"opinionated"*, we mean:

RTL doesn't only give you tools — it **pushes you toward a specific way of testing**.

The Core Philosophy

- ✅ **Test the app the way a user uses it**
- ❌ Don't test internal implementation details

Examples of RTL Practices

- Interact with the UI like a user (click, type, read text)
- Find elements by **accessibility markers**:
    - role
    - labelText
    - placeholder
- Avoid data-testid unless there's no better option

👉 If a user can't see or interact with something, **your test shouldn't either**.

2️⃣ React Testing Library vs Jest / Vitest

React Testing Library

RTL is responsible for:

- Creating a **simulated DOM**
- Providing utilities to:
    - Render components
    - Query elements

    - Assert UI behavior

RTL **does NOT** run tests.

Jest / Vitest

Jest and Vitest are **test runners**.

They handle:

- Finding test files
- Running tests
- Reporting pass/fail results

RTL + Vitest/Jest = complete testing setup

3️⃣ Why Vitest in This Course?

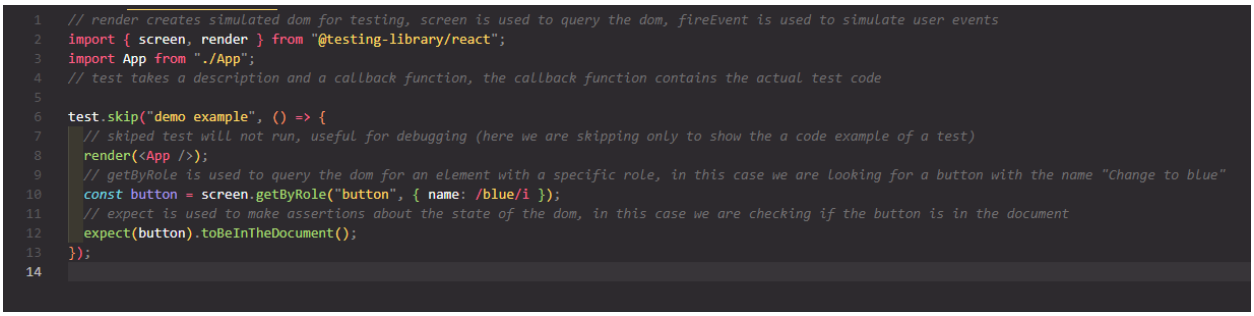
Vitest is used instead of Jest because:

- ⚡ **Much faster** (≈ 3–5x)
- 🧠 Designed to work smoothly with **Vite**
- 😵 Jest is harder to configure with Vite projects

Important Note

- Syntax in this course is **almost identical** between Jest and Vitest
- Differences are mainly:
    - Setup
    - Some advanced features (not used here)

- Code snippet example :



```
// render creates simulated dom for testing, screen is used to query the dom, fireEvent is used to simulate user events
import { screen, render } from "@testing-library/react";
import App from "./App";
// test takes a description and a callback function, the callback function contains the actual test code

test.skip("demo example", () => {
  // skiped test will not run, useful for debugging (here we are skipping only to show the a code example of a test)
  render(<App />);
  // getByRole is used to query the dom for an element with a specific role, in this case we are looking for a button with the name "Change to blue"
  const button = screen.getByRole("button", { name: /blue/i });
  // expect is used to make assertions about the state of the dom, in this case we are checking if the button is in the document
  expect(button).toBeInTheDocument();
});

```

4️⃣ Assertions: How Tests Decide Pass or Fail

Assertions describe **what you expect to be true**.

Basic Assertion Structure

expect(subject).matcher(expectedValue);

- **Subject** → what you're testing
- **Matcher** → how you test it
- **Expected value** → what you expect

Examples

expect(linkElement).toBeInTheDocument();

expect(element.textContent).toBe("hello");

expect(elementsArray).toHaveLength(7);

Matchers refine *how* the subject is checked.

5️⃣ Test-Driven Development (TDD)

TDD is **not a library**, it's a **development philosophy**.

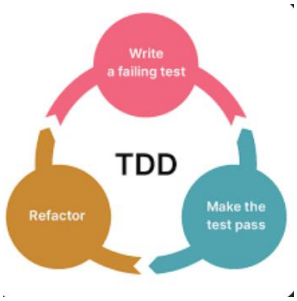
How TDD Works

1. Write the test first ❌ (fails — red)
2. Write code to satisfy the test ✅ (passes — green)

This is why it's often called **Red–Green Testing**.

Why TDD Matters

- Testing becomes **part of coding**, not a chore at the end
- More efficient:
    - Tests re-run automatically
    - Immediate feedback after changes



6️⃣ Types of Tests

Unit Tests

- Test **one unit** in isolation
- Mock dependencies
- Test internal logic/state

Integration Tests

- Test how **multiple units work together**

Functional Tests (RTL's Focus)

- Test **behavior**, not implementation
- Focus on what the user can do and see

End-to-End (E2E) Tests

- Use a real browser & server

- Tools: Cypress, Selenium
- Closest to real user experience

7️⃣ Unit Testing vs Functional Testing

Unit Testing

Pros

- Easy to pinpoint failures
- Fast and isolated

Cons

- Far from real user behavior
- Break easily during refactoring
- Tests internals, not experience

Functional Testing

Pros

- Very close to real user interaction
- More confidence in behavior

Cons

- Harder to debug
- More components involved

👉 RTL encourages **functional testing**.

8️⃣ TDD vs BDD

You might ask:

“If RTL focuses on behavior, shouldn’t we call this BDD?”

Answer: No.

BDD (Behavior-Driven Development):

- Involves collaboration between:
    - Developers
    - QA
    - Business stakeholders

In this course:

- We are **developers only**
- So we follow **TDD**, not BDD