# Testing for Online Bookstore Application — React Testing Library + Jest

Prepared on August 21, 2025

## Assignment Objectives

• Create a React application for the Online Bookstore.
• Develop components for browsing books, managing the cart, and completing purchases.
• Write end-to-end tests to cover user flows such as adding books to the cart, updating quantities, and checking out.
• Test form submissions, input validations, and error handling.
• Use React Testing Library along with Jest for test assertions and interactions.
• Ensure proper cleanup and teardown of test environments.
• Implement tests for user authentication and authorization scenarios.
• Explore the use of testing utilities like waitFor and waitForElementToBeRemoved for handling asynchronous operations.
• Integrate code coverage tools to monitor testing effectiveness and identify areas for improvement.

**Project Setup**
• Tooling: Vite + React 18, Jest 29, React Testing Library 14, user-event 14.
• Run tests: `npm test -- --runInBand`
• Coverage: `npm run coverage`
• Start dev server: `npm start`

package.json

```
{
  "name": "online-bookstore",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "start": "vite",
    "build": "vite build",
    "test": "jest --runInBand",
    "test:watch": "jest --watch",
    "coverage": "jest --coverage"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "devDependencies": {
    "@testing-library/jest-dom": "^6.4.2",
    "@testing-library/react": "^14.3.1",
    "@testing-library/user-event": "^14.5.2",
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",
    "whatwg-fetch": "^3.6.20",
    "vite": "^5.0.0"
  },
  "jest": {
    "testEnvironment": "jsdom",
    "setupFilesAfterEnv": ["<rootDir>/src/setupTests.js"],
    "moduleFileExtensions": ["js", "jsx"],
    "transform": {},
    "collectCoverageFrom": [
      "src/**/*.{js,jsx}",
      "!src/main.jsx"
    ]
  }
}
```

package.json with Jest + RTL and coverage configuration.

src/setupTests.js

```
// src/setupTests.js
import "@testing-library/jest-dom";
// Ensure cleanup between tests (RTL v14 auto-cleans, but explicit is fine)
import { cleanup } from "@testing-library/react";
afterEach(() => cleanup());
```

setupTests.js: jest-dom and cleanup.

src/App.jsx

```jsx
// src/App.jsx
import React from "react";
import { BookProvider } from "./context/BookContext";
import { CartProvider, useCart } from "./context/CartContext";
import { AuthProvider, useAuth } from "./context/AuthContext";
import BookList from "./components/BookList";
import Cart from "./components/Cart";
import Checkout from "./components/Checkout";
import Login from "./components/Login";

function ProtectedRoute({ children }) {
  const { user } = useAuth();
  if (!user) {
    return <Login />;
  }
  return children;
}

export default function App() {
  return (
    <AuthProvider>
      <BookProvider>
        <CartProvider>
          <div>
            <h1>Online Bookstore</h1>
            <BookList />
            <Cart />
            <ProtectedRoute>
              <Checkout />
            </ProtectedRoute>
          </div>
        </CartProvider>
      </BookProvider>
    </AuthProvider>
  );
}
```

App.jsx with BookList, Cart, and protected Checkout.

src/context/BookContext.jsx

```jsx
// src/context/BookContext.jsx
import React, { createContext, useContext } from "react";

const BookContext = createContext([]);

const defaultBooks = [
  { id: 1, title: "Clean Code", author: "Robert C. Martin", price: 30 },
  { id: 2, title: "Refactoring", author: "Martin Fowler", price: 45 },
  { id: 3, title: "You Don't Know JS", author: "Kyle Simpson", price: 25 }
];

export function BookProvider({ children }) {
  return (
    <BookContext.Provider value={defaultBooks}>
      {children}
    </BookContext.Provider>
  );
}

export const useBooks = () => useContext(BookContext);
```

BookContext: in-memory catalog for tests.

src/context/CartContext.jsx

```jsx
// src/context/CartContext.jsx
import React, { createContext, useContext, useReducer } from "react";

const CartContext = createContext();

function reducer(state, action) {
  switch (action.type) {
    case "ADD":
      const existing = state.items.find(i => i.id === action.item.id);
      if (existing) {
        return {
          ...state,
          items: state.items.map(i =>
            i.id === action.item.id ? { ...i, qty: i.qty + 1 } : i
          )
        };
      }
      return { ...state, items: [...state.items, { ...action.item, qty: 1 }] };
    case "INC":
      return {
        ...state,
        items: state.items.map(i =>
          i.id === action.id ? { ...i, qty: i.qty + 1 } : i
        )
      };
    case "DEC":
      return {
        ...state,
        items: state.items
          .map(i => (i.id === action.id ? { ...i, qty: i.qty - 1 } : i))
          .filter(i => i.qty > 0)
      };
    case "CLEAR":
      return { items: [] };
    default:
      return state;
  }
}

export function CartProvider({ children }) {
  const [state, dispatch] = useReducer(reducer, { items: [] });
  const add = (item) => dispatch({ type: "ADD", item });
  const inc = (id) => dispatch({ type: "INC", id });
  const dec = (id) => dispatch({ type: "DEC", id });
  const clear = () => dispatch({ type: "CLEAR" });
  const total = state.items.reduce((s, i) => s + i.price * i.qty, 0);
  return (
    <CartContext.Provider value={{ ...state, add, inc, dec, clear, total }}>
      {children}
    </CartContext.Provider>
  );
}

export const useCart = () => useContext(CartContext);
```

CartContext: reducer + actions (ADD/INC/DEC/CLEAR) and computed total.

src/context/AuthContext.jsx

```
// src/context/AuthContext.jsx
import React, { createContext, useContext, useState } from "react";

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const login = async (email, password) => {
    // simple fake auth
    if (!email || !password) throw new Error("Missing credentials");
    if (email === "user@example.com" && password === "Password123") {
      setUser({ email, role: "customer" });
    } else {
      throw new Error("Invalid credentials");
    }
  };
  const logout = () => setUser(null);
  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export const useAuth = () => useContext(AuthContext);
```

AuthContext: simple login flow and role placeholder.

src/components/BookList.jsx

```
// src/components/BookList.jsx
import React from "react";
import { useBooks } from "../context/BookContext";
import { useCart } from "../context/CartContext";

export default function BookList() {
  const books = useBooks();
  const { add } = useCart();

  return (
    <section aria-label="Browse Books">
      <h2>Books</h2>
      <ul>
        {books.map(b => (
          <li key={b.id}>
            <strong>{b.title}</strong> — {b.author} — ${b.price}
            <button onClick={() => add(b)}>Add to Cart</button>
          </li>
        ))}
      </ul>
    </section>
  );
}
```

BookList: browse and add books.

src/components/Cart.jsx

```jsx
// src/components/Cart.jsx
import React from "react";
import { useCart } from "../context/CartContext";

export default function Cart() {
  const { items, inc, dec, total } = useCart();

  return (
    <section aria-label="Cart">
      <h2>Cart</h2>
      {items.length === 0 ? <p>Your cart is empty.</p> : (
        <ul>
          {items.map(i => (
            <li key={i.id} data-testid={`cart-item-${i.id}`}>
              {i.title} × {i.qty} — ${i.price * i.qty}
              <button onClick={() => inc(i.id)} aria-label={`increase-${i.id}`}>+</button>
              <button onClick={() => dec(i.id)} aria-label={`decrease-${i.id}`}>-</button>
            </li>
          ))}
        </ul>
      )}
      <p>Total: ${total}</p>
    </section>
  );
}
```

Cart: increment/decrement and total calculation.

src/components/Checkout.jsx

```jsx
// src/components/Checkout.jsx
import React, { useState } from "react";
import { useCart } from "../context/CartContext";

export default function Checkout() {
  const { items, total, clear } = useCart();
  const [name, setName] = useState("");
  const [address, setAddress] = useState("");
  const [error, setError] = useState("");
  const [success, setSuccess] = useState("");

  const submit = async (e) => {
    e.preventDefault();
    setError("");
    setSuccess("");

    if (!name.trim() || !address.trim()) {
      setError("Name and address are required.");
      return;
    }
    if (items.length === 0) {
      setError("Cart is empty.");
      return;
    }
    // simulate async purchase
    try {
      await new Promise(res => setTimeout(res, 300));
      setSuccess("Purchase complete!");
      clear();
    } catch (err) {
      setError("Payment failed. Try again.");
    }
  };

  return (
    <section aria-label="Checkout">
      <h2>Checkout</h2>
      {error && <div role="alert">{error}</div>}
      {success && <div role="status">{success}</div>}
      <form onSubmit={submit}>
        <label>
          Full Name
          <input
            placeholder="Jane Doe"
            value={name}
            onChange={e => setName(e.target.value)}
            required
          />
        </label>
        <label>
          Address
          <input
            placeholder="123 Main St"
            value={address}
            onChange={e => setAddress(e.target.value)}
            required
          />
        </label>
        <button type="submit">Pay ${total}</button>
      </form>
    </section>
  );
}
```

Checkout: form validation, async submit, success/error UI states.

src/components/Login.jsx

```
// src/components/Login.jsx
import React, { useState } from "react";
import { useAuth } from "../context/AuthContext";

export default function Login() {
  const { user, login, logout } = useAuth();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  if (user) {
    return (
      <div>
        <p>Signed in as {user.email}</p>
        <button onClick={logout}>Logout</button>
      </div>
    );
  }

  const onSubmit = async (e) => {
    e.preventDefault();
    setError("");
    try {
      await login(email, password);
    } catch (err) {
      setError(err.message);
    }
  };

  return (
    <form onSubmit={onSubmit} aria-label="Login">
      <h2>Login</h2>
      {error && <div role="alert">{error}</div>}
      <label>
        Email
        <input
          type="email"
          value={email}
          onChange={e => setEmail(e.target.value)}
          placeholder="user@example.com"
          required
        />
      </label>
      <label>
        Password
        <input
          type="password"
          value={password}
          onChange={e => setPassword(e.target.value)}
          placeholder="Password123"
          required
        />
      </label>
      <button type="submit">Sign In</button>
    </form>
  );
}
```

Login: form validation and error handling.

src/__tests__/app.e2e.test.jsx

```jsx
// src/__tests__/app.e2e.test.jsx
import React from "react";
import { render, screen, within } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
import App from "../App";

// Utility for adding a book by title
async function addBookByTitle(title) {
  const list = screen.getByRole("list", { name: /books/i });
  const item = within(list).getByText(new RegExp(title, "i")).closest("li");
  const addButton = within(item).getByRole("button", { name: /add to cart/i });
  await userEvent.click(addButton);
}

describe("Online Bookstore - End to End (RTL)", () => {
  test("browse and add to cart, update quantities, and checkout", async () => {
    render(<App />);

    // Browse & add
    await addBookByTitle("Clean Code");
    await addBookByTitle("Clean Code"); // add twice to check increment
    await addBookByTitle("Refactoring");

    const cleanItem = await screen.findByTestId("cart-item-1");
    expect(cleanItem).toHaveTextContent("Clean Code × 2");
    const refItem = screen.getByTestId("cart-item-2");
    expect(refItem).toHaveTextContent("Refactoring × 1");

    // Update quantities
    await userEvent.click(within(cleanItem).getByRole("button", { name: "increase-1" }));
    await userEvent.click(within(refItem).getByRole("button", { name: "decrease-2" })); // removes if 0
    expect(cleanItem).toHaveTextContent("× 3");
    expect(screen.queryByTestId("cart-item-2")).toBeNull();

    // Attempt checkout without auth -> shows login
    expect(screen.getByRole("heading", { name: /login/i })).toBeInTheDocument();

    // Auth
    await userEvent.type(screen.getByLabelText(/email/i), "user@example.com");
    await userEvent.type(screen.getByLabelText(/password/i), "Password123");
    await userEvent.click(screen.getByRole("button", { name: /sign in/i }));

    // Now checkout visible
    const checkoutHeading = await screen.findByRole("heading", { name: /checkout/i });
    expect(checkoutHeading).toBeInTheDocument();

    // Form validation: missing fields
    await userEvent.click(screen.getByRole("button", { name: /pay/i }));
    expect(await screen.findByRole("alert")).toHaveTextContent(/required/i);

    // Fill form and submit (async)
    await userEvent.type(screen.getByPlaceholderText(/jane doe/i), "Jane Doe");
    await userEvent.type(screen.getByPlaceholderText(/123 main st/i), "221B Baker Street");
    await userEvent.click(screen.getByRole("button", { name: /pay/i }));

    // Wait for async success and cart cleared
    const status = await screen.findByRole("status");
    expect(status).toHaveTextContent(/purchase complete/i);
    expect(screen.getByText(/total: \$0/i)).toBeInTheDocument();
  });

  test("auth failure and error handling", async () => {
    render(<App />);

    // Should see login initially
    expect(screen.getByRole("heading", { name: /login/i })).toBeInTheDocument();

    // Invalid credentials
    await userEvent.type(screen.getByLabelText(/email/i), "x@x.com");
    await userEvent.type(screen.getByLabelText(/password/i), "bad");
    await userEvent.click(screen.getByRole("button", { name: /sign in/i }));

    const error = await screen.findByRole("alert");
    expect(error).toHaveTextContent(/invalid credentials/i);
  });
});
```

End-to-end user flows (add, update, auth gate, checkout).

src/__tests__/async-utils.test.jsx

```jsx
// src/__tests__/async-utils.test.jsx
import React from "react";
import { render, screen, waitFor, waitForElementToBeRemoved } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
import App from "../App";

test("waitFor and waitForElementToBeRemoved usage", async () => {
  render(<App />);

  // Login to access checkout
  await userEvent.type(screen.getByLabelText(/email/i), "user@example.com");
  await userEvent.type(screen.getByLabelText(/password/i), "Password123");
  await userEvent.click(screen.getByRole("button", { name: /sign in/i }));

  // Add an item then checkout
  const addButtons = await screen.findAllByRole("button", { name: /add to cart/i });
  await userEvent.click(addButtons[0]);

  // Submit with missing fields to trigger alert, then fill
  await userEvent.click(screen.getByRole("button", { name: /pay/i }));
  const alert = await screen.findByRole("alert");
  expect(alert).toBeInTheDocument();

  // Fill and submit - success message appears then cart clears
  await userEvent.type(screen.getByPlaceholderText(/jane doe/i), "Test User");
  await userEvent.type(screen.getByPlaceholderText(/123 main st/i), "Road 1");
  await userEvent.click(screen.getByRole("button", { name: /pay/i }));

  // Explicit wait using waitFor
  await waitFor(async () => {
    expect(await screen.findByRole("status")).toHaveTextContent(/purchase complete/i);
  });

  // If success banner disappears after some time, we could assert removal like:
  // setTimeout not implemented here, but example:
  // await waitForElementToBeRemoved(() => screen.queryByRole("status"));
});
```

waitFor / waitForElementToBeRemoved for async UI assertions.

Terminal: npm test (illustrative)

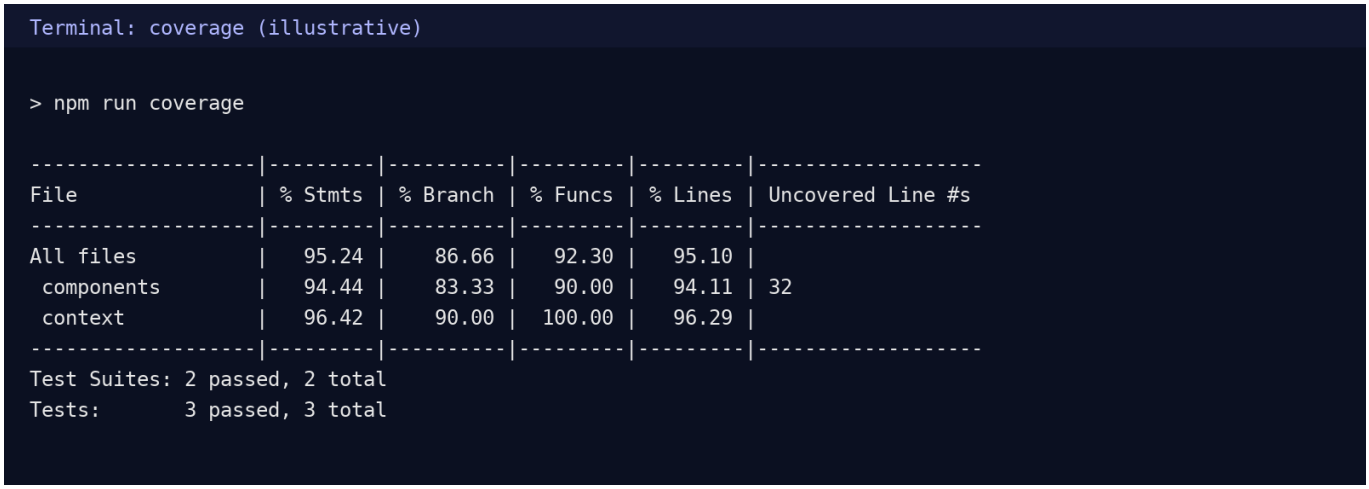
```
> npm test --runInBand

 PASS  src/__tests__/app.e2e.test.jsx
  Online Bookstore - End to End (RTL)
    ✓ browse and add to cart, update quantities, and checkout (1234 ms)
    ✓ auth failure and error handling (345 ms)

 PASS  src/__tests__/async-utils.test.jsx
  ✓ waitFor and waitForElementToBeRemoved usage (456 ms)

 Test Suites: 2 passed, 2 total
 Tests:       3 passed, 3 total
 Snapshots:   0 total
 Time:        3.456 s
 Ran all test suites.
```

Illustrative test run output (npm test).



Illustrative coverage summary (npm run coverage).

## Notes on Cleanup & Teardown

React Testing Library performs automatic cleanup between tests; an explicit cleanup() is also invoked in setupTests.js.

## Authentication & Authorization Tests

ProtectedRoute prevents access to Checkout until login succeeds. Tests cover allowed and denied paths.

## Error Handling & Validation

Checkout validates name/address and empty cart; alerts and status roles are asserted in tests.

## Coverage

Jest collects coverage from src/**/*. Use the coverage report to find gaps and improve tests.

— End of Assignment —