Testing Components

What We Will Learn

- Introduction to Testing
- Using Jest with Test Utils from React-DOM
- Using Jest with the React Testing Library
- Using Jest with Enzyme

Lorem Ipsum is simply

Testing Components

Testing is integral to software development

A process that evaluates if a software performs as per its intended design

Test Driven Development

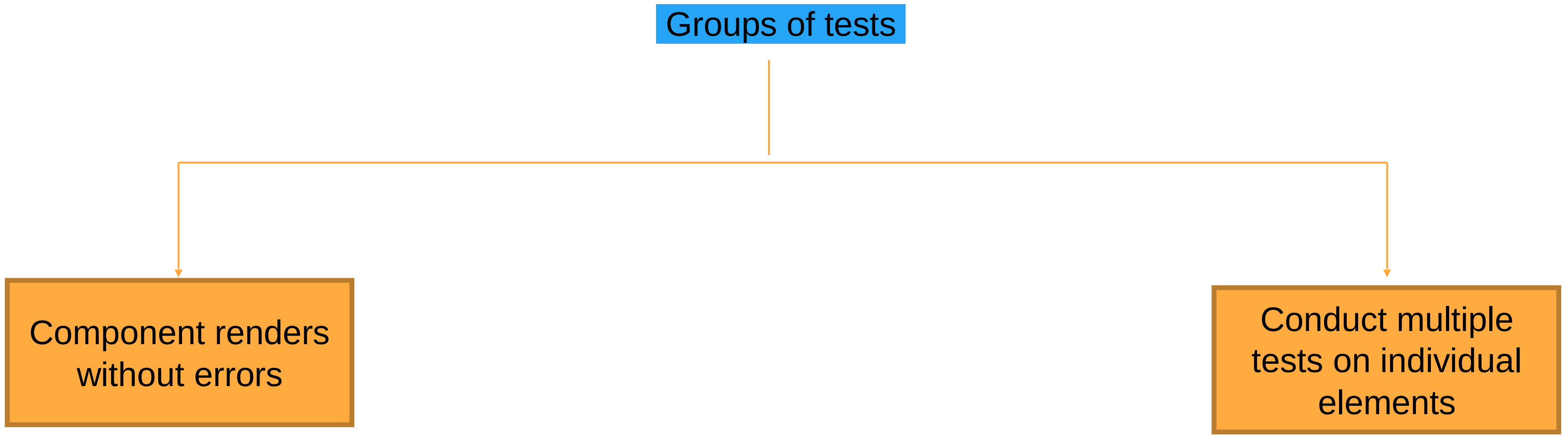
Three types of software testing strategies

Testing Components

Three types of Testing

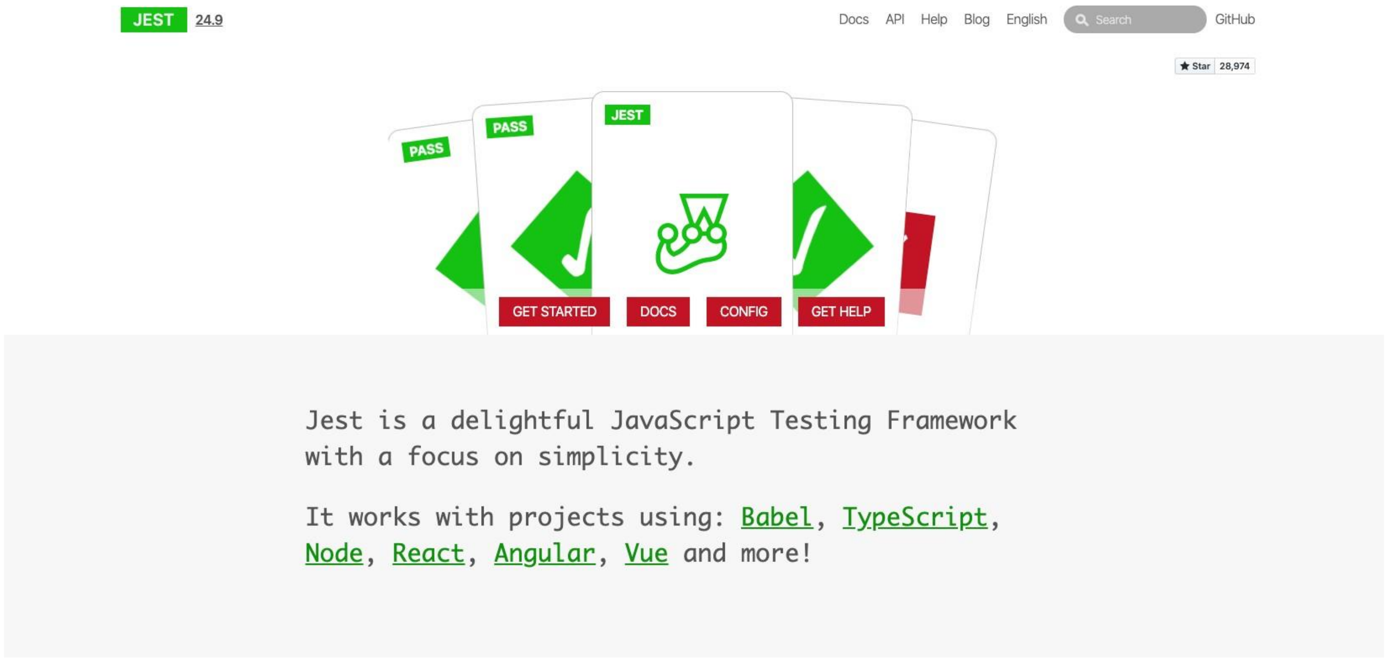
- Unit Testing
- Integration Testing
- End to End Testing

Testing Components


Groups of tests
Component renders without errors
Conduct multiple tests on individual elements

Testing Components

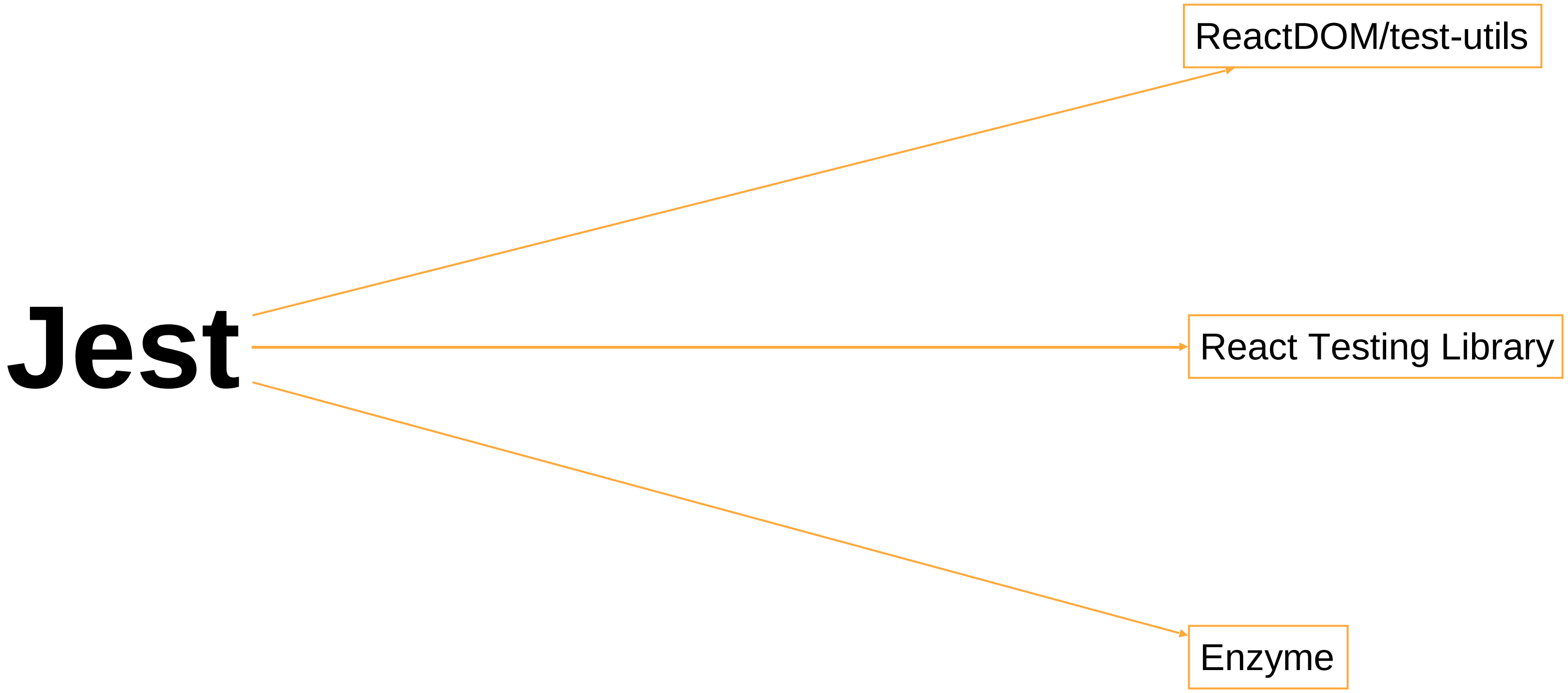
Jest



Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!

Testing Components



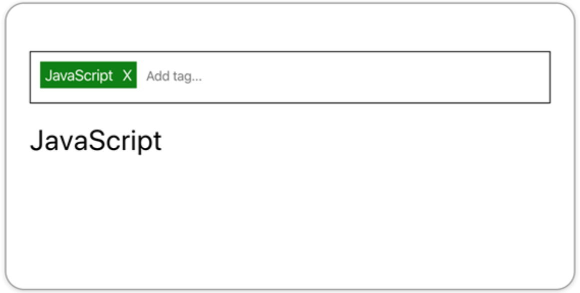
Testing Components

Code Demo

Testing Components

Snapshot Testing

Snapshot captures the rendered output of a Component on disk.



SNAPSHOT

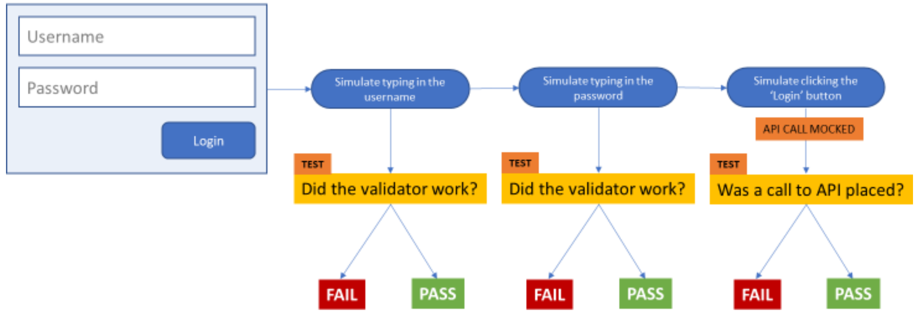
```
// Jest Snapshot v1, https://goo.gl/fbAQLP


exports[`PowerTags component renders correctly`] = `"<div class=\\"power-tags\\"><div class=\\"tag\\">JavaScript<div class=\\"del-btn\\">X</div></div><div class=\\"tag\\">CSS<div class=\\"del-btn\\">X</div></div><input type=\\"text\\" id=\\"input-tags\\" placeholder=\\"Add tag...\\" value=\\"\\"></div>"`;
```

Testing Components

Jest Features

Functional Tests - Testing user interaction and functionality

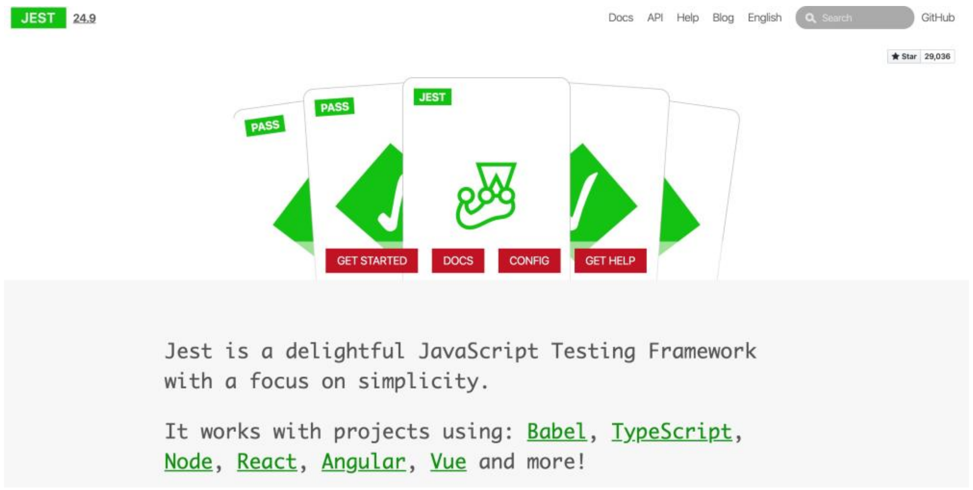
Functional Testing a Login Form Component


Username
Password
Login
Simulate typing in the username
Simulate typing in the password
Simulate clicking the 'Login' button
API CALL MOCKED
TEST
Did the validator work?
TEST
Did the validator work?
TEST
Was a call to API placed?
FAIL
PASS
FAIL
PASS
FAIL
PASS

- √ Render the component in a simulated environment
- √ Perform interaction such as clicking buttons, filling up forms
- √ Evaluate if the Component performed as intended

Using Jest with React-DOM Test Utils

Testing Components



Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!

- Maintained by open source contributors & employees from Facebook
- Can be used to write tests for React, Angular, Vue, Node and more!
- Ships with JSDom, a JavaScript based headless browser
- Provides us with tools to write test suites to define expectations and matcher functions.

Testing Components

Code Demo

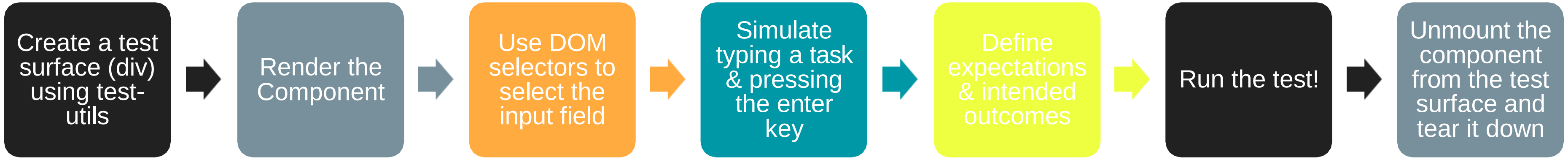
Testing Components

<AddToDo onAdd={task => {}**} />**

What is the test going to evaluate?

1. Type a task and press enter
2. The task String should be returned in the onAdd prop
3. The input field should be cleared out

Testing Components


Create a test surface (div) using test-utils
Render the Component
Use DOM selectors to select the input field
Simulate typing a task & pressing the enter key
Define expectations & intended outcomes
Run the test!
Unmount the component from the test surface and tear it down

Testing Components

<StatusButton status={false**} onDone={**val => {}**} />**

1. When the status prop is 'false' and the component is clicked, the onDone prop function should return true.

2. When the status prop is 'true' and the component is clicked, the onDone prop function should return false.

3. Test conditional overloading of the CSS class

Testing Components

<App />

1. UI consistency using snapshot testing

2. Render the app so it fetches tasks from a mocked API (not the live API server!) & take a snapshot of the rendered UI

3. Add a new task and again take a snapshot

Testing Components

```
import React from "react";
import { render, unmountComponentAtNode } from "react-dom";
import { act, Simulate } from "react-dom/test-utils";
import AddToDo from "../components/AddToDo";


let div = null;


beforeEach(() => {
  div = document.createElement("div");
  document.body.appendChild(div);
});


afterEach(() => {
  unmountComponentAtNode(div);
  div.remove();
  div = null;
});


describe("Testing AddToDo.js component", () => {
  it("Returns the contents of the input field using the onAdd prop", async () => {
    const onAddFn = jest.fn();
    await act(async () => {
      render(<AddToDo onAdd={onAddFn} />, div);
    });


    const inputFld = document.querySelector("input");


    await act(async () => {
      await Simulate.change(inputFld, {
        target: { value: "This is a test task" }
      });
      await Simulate.keyUp(inputFld, { key: "Enter", keyCode: 13 });
    });
```

Using Jest with React Testing Library

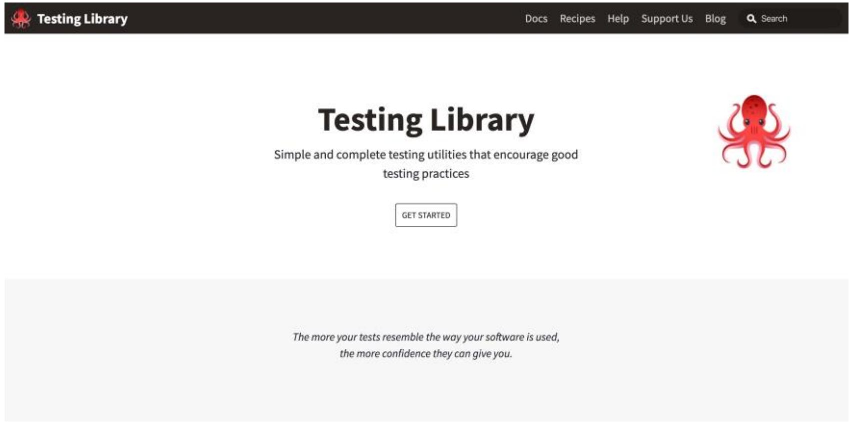
Testing Components

```
import REACT from "REACT";
import { render, unmountComponentAtNode } from "REACT-DOM";
import { ACT, SIMULATE } from "REACT-DOM/TEST-UTILS";
import AddToDo from "../components/AddToDo";


let div = null;


BEFOREEACH(() => {
  div = document.CREATEELEMENT("div");
  document.BODY.APPENDCHILD(DIV);
});


AFTEREACH(() => {
  unmountComponentAtNode(div);
  div.remove();
  div = null;
});


describe("Testing AddToDo.js component", () => {
  it("Returns the contents of the input field using the onAdd prop", ASYNC () => {
    const onAddFn = jest.fn();
    AWAIT ACT(ASYNC () => {
      render(<AddToDo onAdd={onAddFn} />, div);
    });


    const inputFld = document.querySelector("input");


    AWAIT ACT(ASYNC () => {
      AWAIT SIMULATE.CHANGE(INPUTFLD, {
        TARGET: { VALUE: "This is A test TASK" }
      });
      AWAIT SIMULATE.KEYUP(INPUTFLD, { key: "Enter", keyCode: 13 });
    });
```

The Setup

Using DOM selectors

Testing Components



Testing Library

Simple and complete testing utilities that encourage good testing practices

The more your tests resemble the way your software is used, the more confidence they can give you.

- Built with DOM Testing Library
- Framework agnostic and can be used with Angular, Vue and React using bindings
- React Binding == React Testing Library
- Provides utility functions that let you focus on writing maintainable tests rather than spend hours on implementation details.

https://testing-library.com/

Testing Components

Guiding Principle : **The more your tests resemble the way your software is used, the more confidence they can give you.**

- React Testing Library is NOT a test runner or framework.
- Is a set of helper utilities that simplify the process of writing & maintaining tests

Testing Components

Code Demo

Testing Components

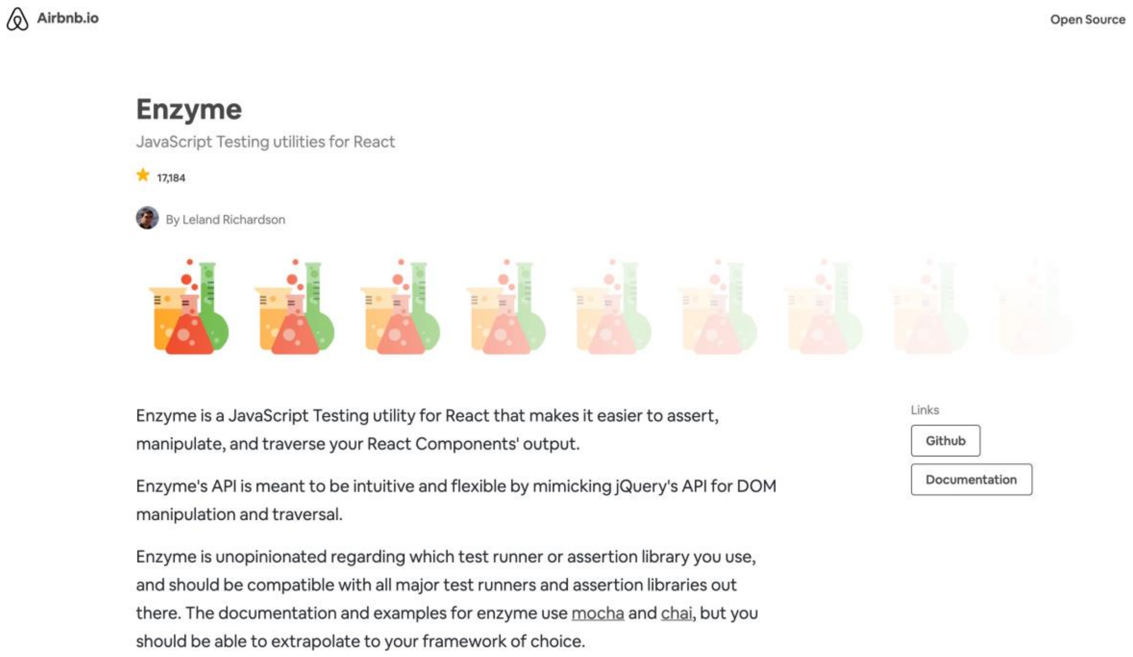
```
import React from "react";
import { render, unmountComponentAtNode } from "react-dom";
import { act, Simulate } from "react-dom/test-utils";
import AddToDo from "../components/AddToDo";


let div = null;


beforeEach(() => {
  div = document.createElement("div");
  document.body.appendChild(div);
});


afterEach(() => {
  unmountComponentAtNode(div);
  div.remove();
  div = null;
});


describe("Testing AddToDo.js component", () => {
  it("Returns the contents of the input field using the onAdd prop", async () => {
    const onAddFn = jest.fn();
    await act(async () => {
      render(<AddToDo onAdd={onAddFn} />, div);
    });


    const inputFld = document.querySelector("input");


    await act(async () => {
      await Simulate.change(inputFld, {
        target: { value: "This is a test task" }
      });
      await Simulate.keyUp(inputFld, { key: "Enter", keyCode: 13 });
    });
```

```
import React from "react";
import { render, fireEvent, cleanup } from "@testing-library/react";
import AddToDo from "../components/AddToDo";


afterEach(cleanup);


describe("Testing AddToDo.js component", () => {
  it("Returns the contents of the input field using the onAdd prop", async () => {
    const onAddFn = jest.fn();
    const { getByPlaceholderText } = render(<AddToDo onAdd={onAddFn} />);
    const taskInput = getByPlaceholderText(/Add a task/i);


    fireEvent.change(taskInput, {
      target: { value: "This is a brand new task" }
    });
    fireEvent.keyUp(taskInput, { key: "Enter", keyCode: 13 });


    expect(taskInput.value).toBe("")
    ;
    expect(onAddFn).toBeCalledWith({
      done: false,
      title: "This is a brand new task"
    });
  });
});
```

Using Jest with Enzyme

Testing Components

Enzyme



Enzyme

JavaScript Testing utilities for React

★ 17,184

By Leland Richardson

Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.

Enzyme's API is meant to be intuitive and flexible by mimicking jQuery's API for DOM manipulation and traversal.

Enzyme is unopinionated regarding which test runner or assertion library you use, and should be compatible with all major test runners and assertion libraries out there. The documentation and examples for enzyme use mocha and chai, but you should be able to extrapolate to your framework of choice.

Testing Components

Three kinds of renderers

Testing Components

Shallow Renderer

shallow(<Component />);

- Useful for testing components in isolation
- Renders components one level deep
- Doesn’t let component affect behavior of child components resulting in pure isolation
- Currently, doesn’t support the useState() hook

Testing Components

Full DOM Renderer

mount(<Component />);

- Components and their child tree rendered using jsDOM – a headless browser
- Full DOM APIs available
- Component is rendered into the DOM
- Supports useState() and other hooks

Testing Components

Static Renderer

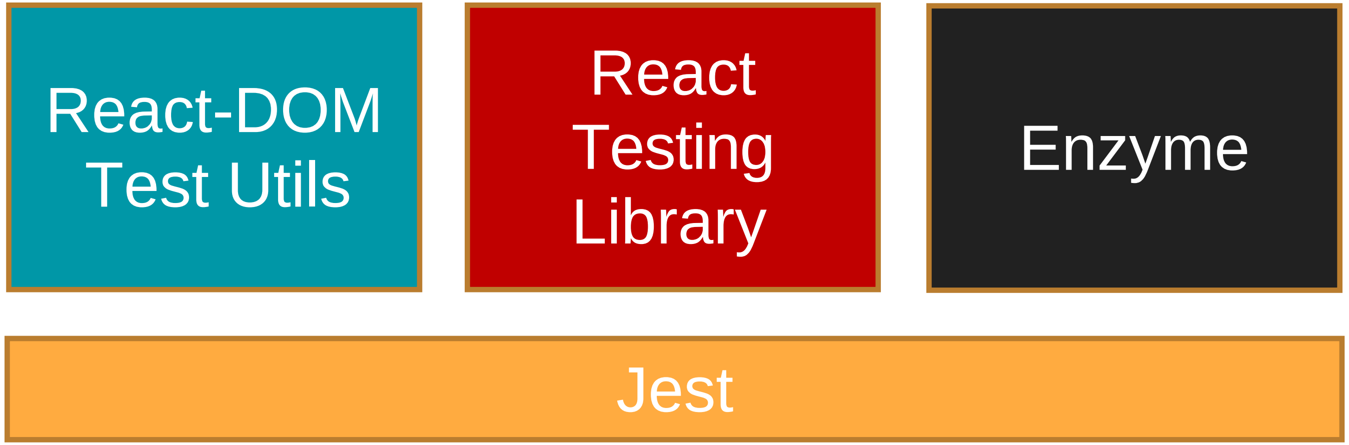
render(<Component />);

- Renders a static markup
- You can traverse and parse the HTML using Cheerio, which comes built-in.

Testing Components

Code Demo

Testing Components



Test Driven Development is key to writing error free and maintainable code

thank you!