Module 02

Purwadhika
Digital Technology School

Fullstack Web Developer

# Advanced topic & intro to TDD in React JS

Job Connector Program

## Outline

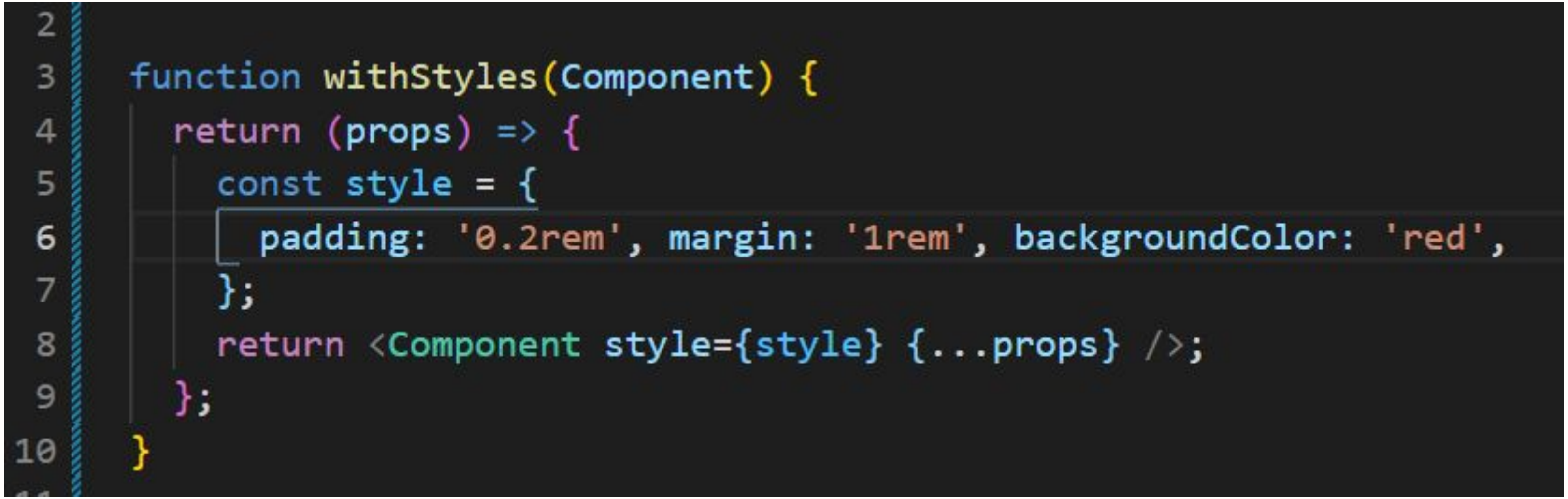
- Higher Order Component (HOC)
- Optimize performance with code splitting
- Error handling & debugging
- Build and deploy
- Add test, run test, refactor, repeat

## Higher Order Component (HOC)

A Higher Order Component (HOC) is a component that receives another component. The HOC contains certain logic that we want to apply to the component that we pass as a parameter. After applying that logic, the HOC returns the element with the additional logic.

## Higher Order Component (HOC)

Say that we always wanted to add a certain styling to multiple components in our application. Instead of creating a style object locally each time, we can simply create a HOC that adds the style objects to the component that we pass to it

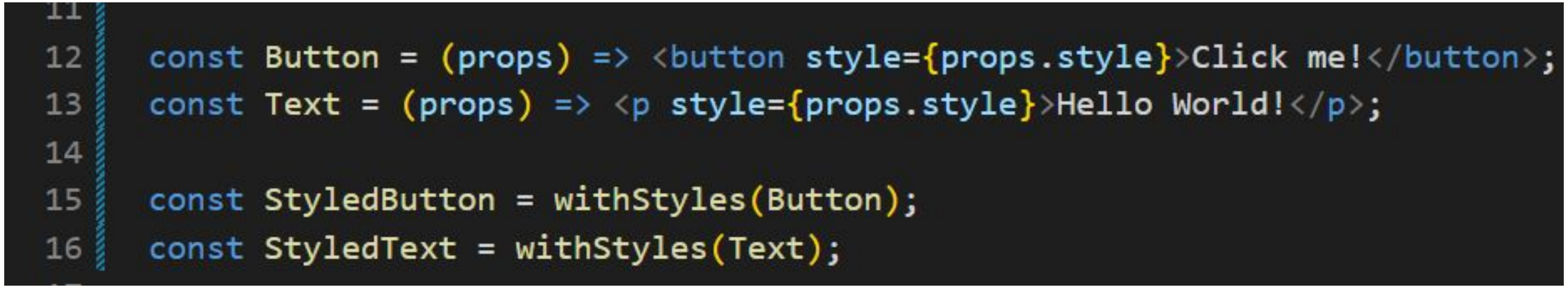


```
function withStyles(Component) {
  return (props) => {
    const style = {
      padding: '0.2rem', margin: '1rem', backgroundColor: 'red',
    };
    return <Component style={style} {...props} />;
  };
}
```

# Higher Order Component (HOC)

In this case we will apply this style into Button and Text components.

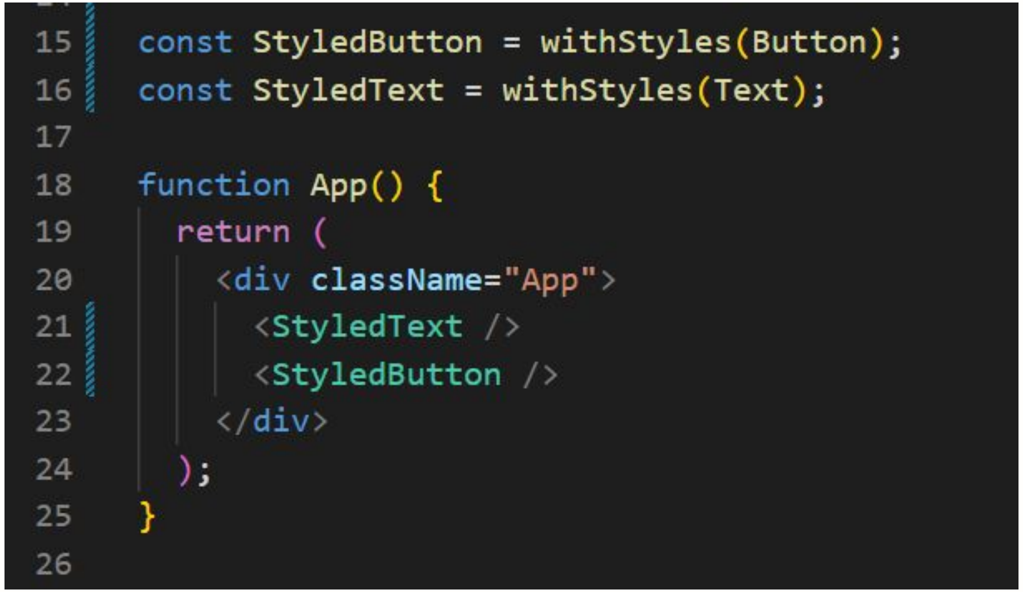
As you can see in line 15 and 16 we applies the styling into Button and Text Components



```
12  const Button = (props) => <button style={props.style}>Click me!</button>;
13  const Text = (props) => <p style={props.style}>Hello World!</p>;
14
15  const StyledButton = withStyles(Button);
16  const StyledText = withStyles(Text);
```

## Higher Order Component (HOC)

After apply the styling into the components, call that components into your page.



```
const StyledButton = withStyles(Button);
const StyledText = withStyles(Text);

function App() {
  return (
    <div className="App">
      <StyledText />
      <StyledButton />
    </div>
  );
}
```

## Optimize performance with code splitting

Code splitting is simply dividing huge code bundles into smaller code chunks that can be loaded ad hoc. Usually, when the SPAs grow in terms of components and plugins, the need to split the code into smaller-sized chunks arises. Bundlers like Webpack and Rollup provide support for code splitting.

Several different code splitting strategies can be implemented depending on the application structure. We will be taking a look at an example in which we implement code splitting inside an admin dashboard for better performance.

## React.lazy() for Code Splitting

React.lazy allows us to use dynamically imported components. This means that we can load these components when they're needed and reduce bundle size. As our dashboard app has several top-level routes that are wrapped inside react-router's Switch, we know that they will never need to be at once.

So, apparently, we can split these top-level components into several different bundle chunks and load them ad hoc.

## React.lazy() for Code Splitting

From :

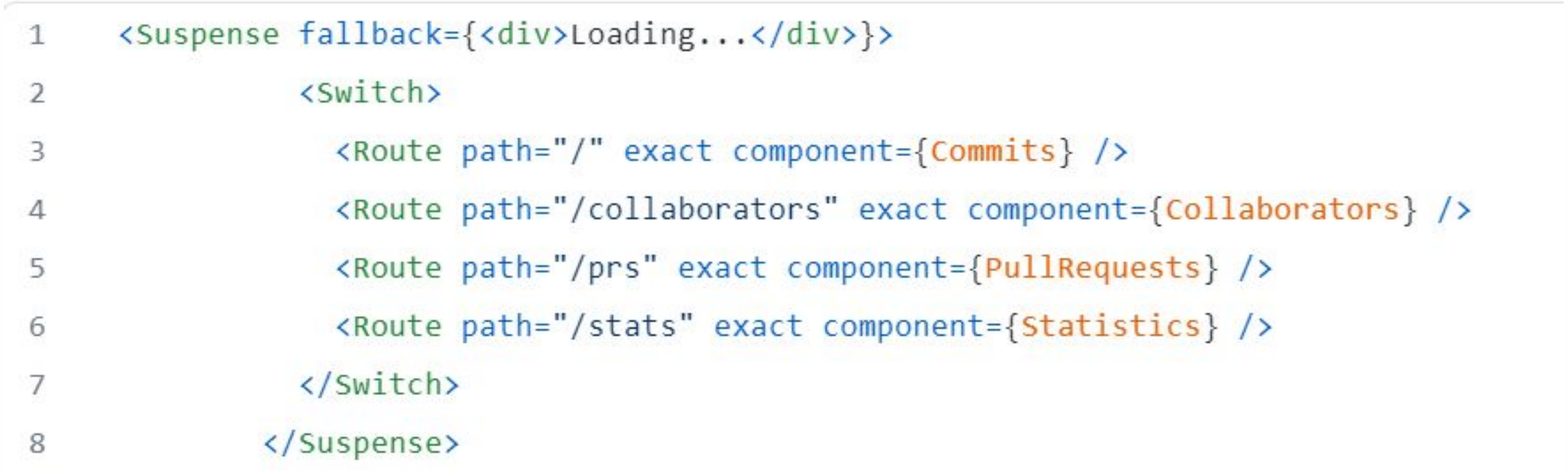
```
import Collaborators from './Collaborators';

import PullRequests from './PullRequests';

import Statistics from './Statistics';
```

To :

```
const Commits = React.lazy(() => import('./Commits'));

const Collaborators = React.lazy(() => import('./Collaborators'));

const Statistics = React.lazy(() => import('./Statistics'));
```

## React.lazy() for Code Splitting

This also requires us to implement a Suspense wrapper around our routes, which does the work of showing fallback visuals till the dynamically loading component is visible.



```
<Suspense fallback={<div>Loading...</div>}>
          <Switch>
            <Route path="/" exact component={Commits} />
            <Route path="/collaborators" exact component={Collaborators} />
            <Route path="/prs" exact component={PullRequests} />
            <Route path="/stats" exact component={Statistics} />
          </Switch>
        </Suspense>
```

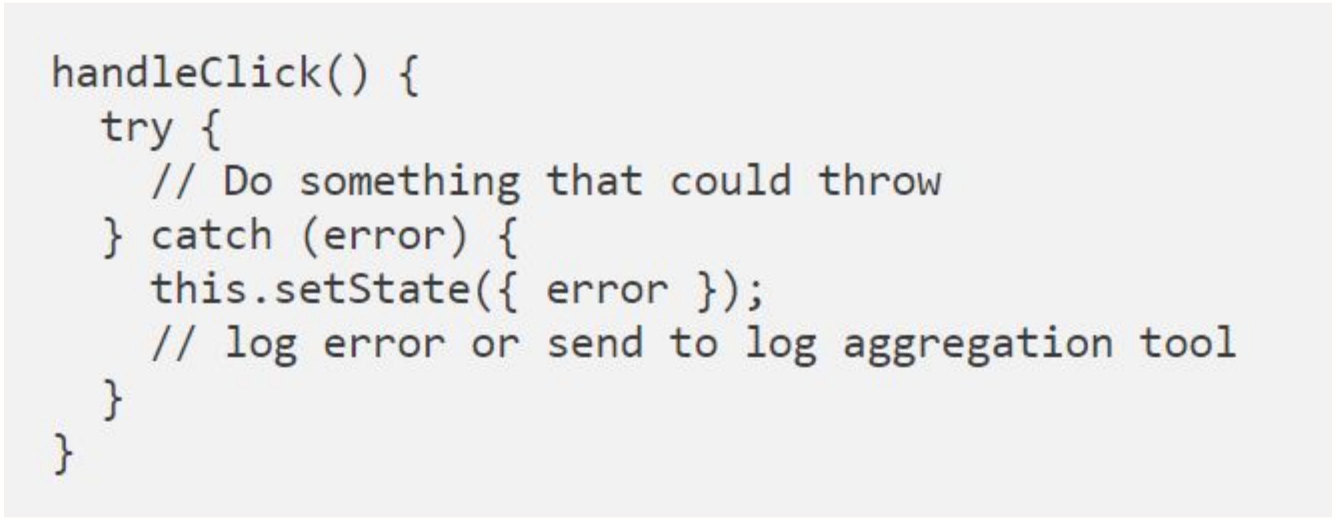
## Error Handling

**Error handling** (also called exception handling) refers to catching errors and handling them gracefully without impacting the user experience. This might include showing a blank page or error message.

Exception handling can be seen as a way of providing Resilience to our application. In other words, if you handle exceptions, your application won’t crash when it throws an exception. You ensure your application can continue to operate normally with little impact on the user experience.

## Error Handling - Try-Catch Statement

The try-catch statement is a best practice and is useful for catching event handler exceptions in React. Because event boundaries don’t catch errors inside React event handlers—such as an onClick event—you should use a try-catch statement to handle these exceptions.



```
handleClick() {
  try {
    // Do something that could throw
  } catch (error) {
    this.setState({ error });
    // log error or send to log aggregation tool
  }
}
```

## Error Handling - Sentry

Purwadhika
Digital Technology School

Sentry provides a custom error boundary component for React, automatically sending JavaScript errors within a component tree to Sentry. You can use it similarly to React's basic error boundary component.

Go to: [https://docs.sentry.io/platforms/javascript/guides/react/](https://docs.sentry.io/platforms/javascript/guides/react/)

## Debugging Code

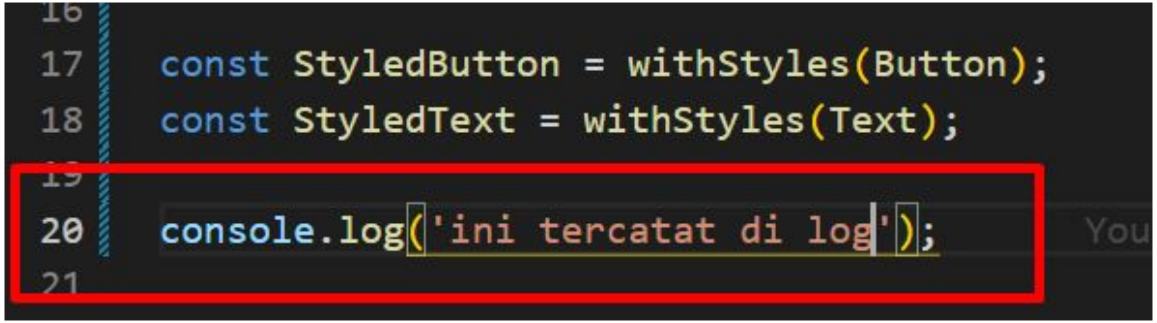
Debugging is the art of removing bugs. There are multiple ways to debug in React. In terms of React, we can have many different kinds of bugs, including:

- User interface bugs — something’s wrong in the user interface, or it just doesn’t look right
- Logic bugs — the application doesn’t behave the way we expect
- Networking bugs — although the logic is correct, a third-party service doesn’t behave as we expect
- Regressions — a feature that worked in the past doesn’t work anymore

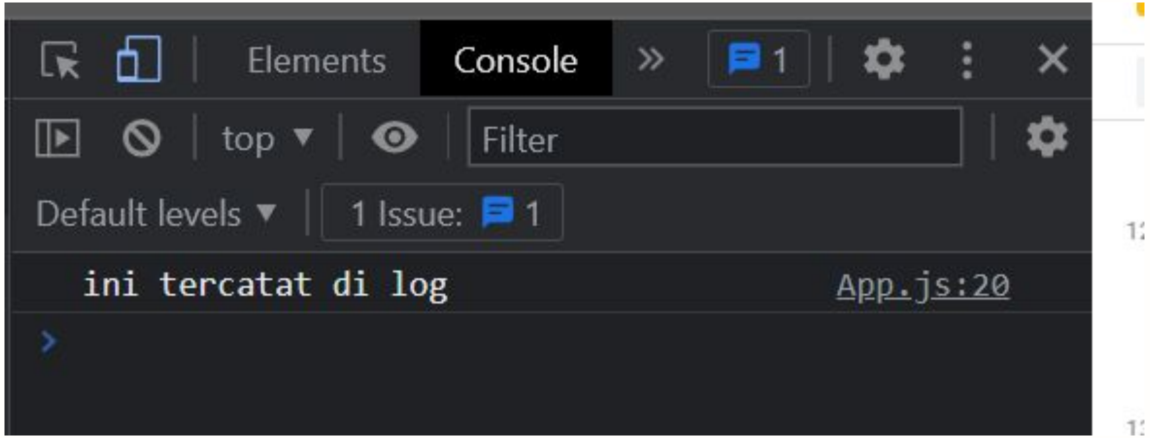
## Debug React using Console Statements

One of the classic ways to debug an application is to log it. In the web, you can achieve this by using console.log, console.warn, console.error, and similar statements. Then, to examine application state, for example, you would do console.log(count). You'll then see the result in the browser inspector. Developers use this method because it's fast to implement and it relies on your reasoning skills. That said, using a browser inspector is a more powerful approach since it gives you a better picture of what's going on.

## Debug React using Console Statements



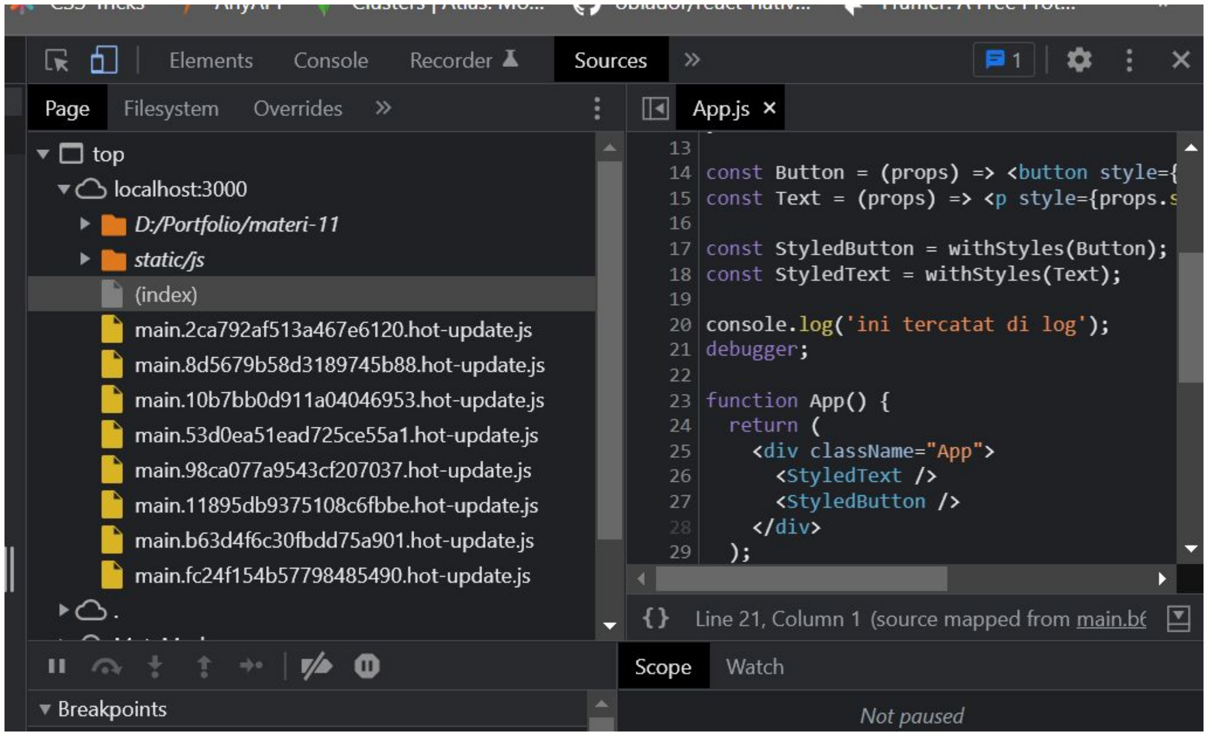
```
const StyledButton = withStyles(Button);
const StyledText = withStyles(Text);

console.log('ini tercatat di log');
```



```
ini tercatat di log                                    App.js:20
```

# Debug React Using The Browser Inspector

The browser inspector contains many powerful tools, including a debugger. It can be activated from the code by using a debugger; statement. Writing debugger; is the same as if you were adding a breakpoint to your code using the browser.



Elements
Console
Recorder
Sources
Page
Filesystem
Overrides
App.js
top
localhost:3000
D:/Portfolio/materi-11
static/js
(index)
main.2ca792af513a467e6120.hot-update.js
main.8d5679b58d3189745b88.hot-update.js
main.10b7bb0d911a04046953.hot-update.js
main.53d0ea51ead725ce55a1.hot-update.js
main.98ca077a9543cf207037.hot-update.js
main.11895db9375108c6fbbe.hot-update.js
main.b63d4f6c30fbdd75a901.hot-update.js
main.fc24f154b57798485490.hot-update.js
Breakpoints
Scope
Watch
Not paused

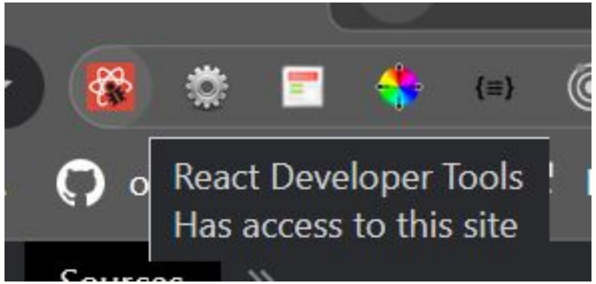
```
const Button = (props) => <button style={
const Text = (props) => <p style={props.s

const StyledButton = withStyles(Button);
const StyledText = withStyles(Text);

console.log('ini tercatat di log');
debugger;

function App() {
  return (
    <div className="App">
      <StyledText />
      <StyledButton />
    </div>
  );
```

Line 21, Column 1 (source mapped from main.b6

## Debug Using React DevTools

To debug React better, it can be a good idea to install React Developer Tools. It's an extension for Chrome-based browsers and Mozilla Firefox that implements React-specific debugging functionality. After installing the extension, open the React tab in your browser. You should notice the Elements and Profiler tabs.

Go to: https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html#installation



## Debug React and Redux

If you're using Redux with your application, Redux DevTools Extension gives you insight into the state of your application at a given time. The extension also allows you to travel in time through different states. It requires a certain amount of setup in your application. but it's all explained in the documentation of the extension.

Go to: https://extension.remotedev.io/

## Debug React Networking Issues

Usually, an application is connected to a remote backend. That brings a new source of bugs, as the data you receive from a backend might not be what you expect. Or perhaps you made a mistake in your query. To debug these sorts of issues, consider using the Networking tab of the inspector. You'll see all sent network requests there, and you'll be able to inspect their contents further.

Doing this work in tandem with another technique, such as inserting a debugger; statement into a good place, can be a decent way to debug these issues.

## Build and Deploy - Netlify

The fastest and easy way to deploy a React application is just to drag and drop the build folder in Netlify.

To create a build folder, just execute the **npm run build** or **yarn build** command from the command line from your project folder.

Once **the build folder** is created, you just need to drop the folder in the drop area under **the sites menu**

Go to: https://www.netlify.com/

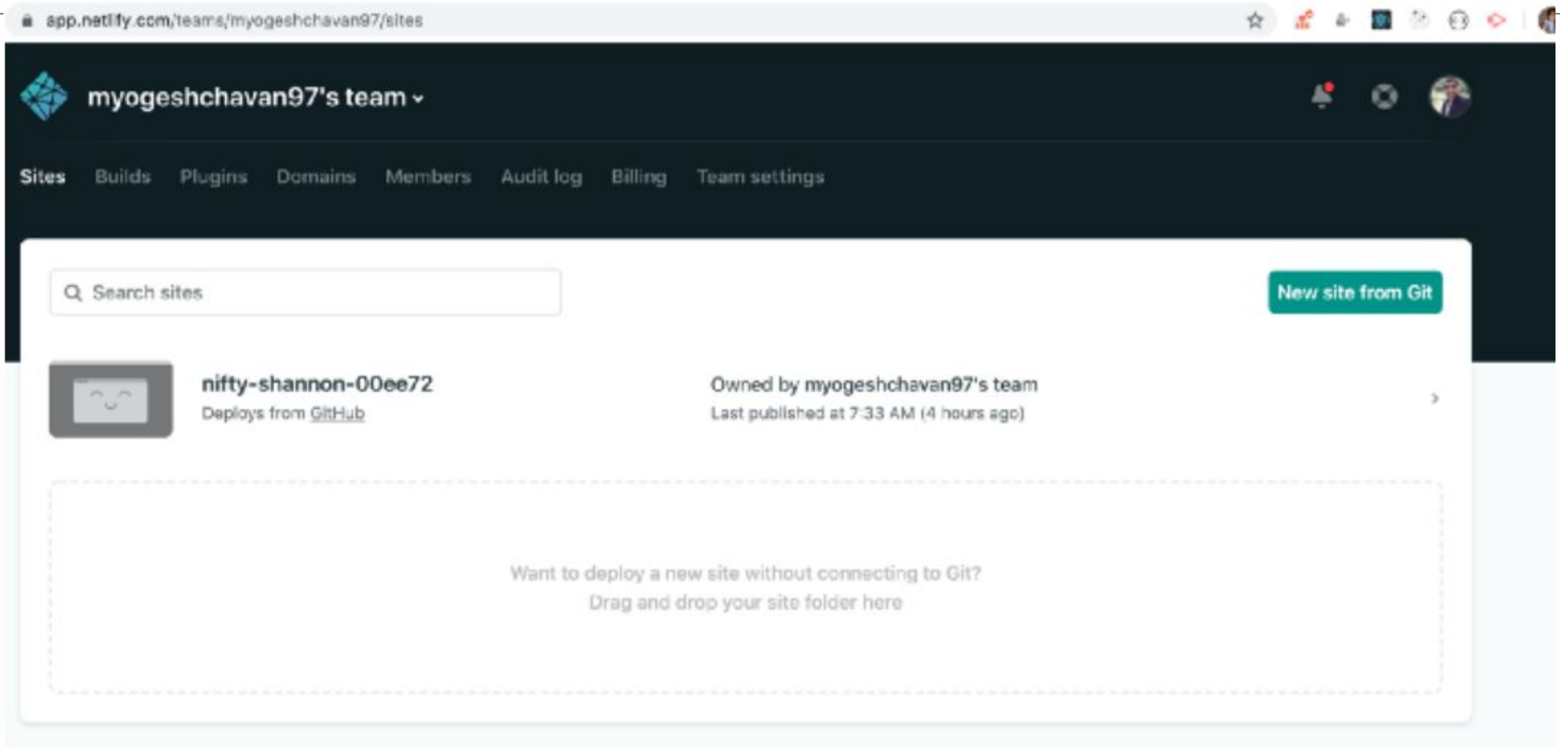
## Build and Deploy - Netlify

**Build and deploy through github** - whenever you push any changes to the GitHub repository, it will automatically be deployed to Netlify. You can also see all deployed versions and easily roll back to any previously working version of code with just a single click.

If you already have a repository pushed to GitHub, then you just need to connect it.

Login to your Netlify account. In the dashboard, click on the New site from Git button.

## Build and Deploy – Netlify


app.netlify.com/teams/myogeshchavan97/sites

myogeshchavan97's team

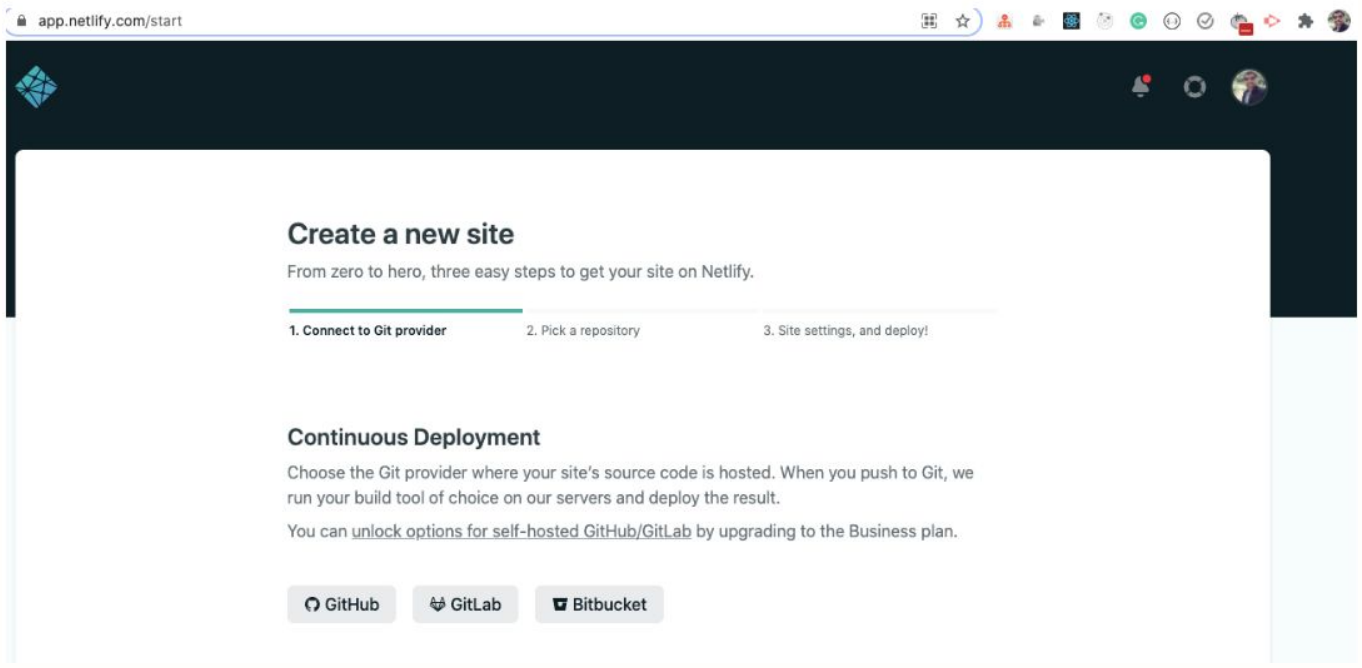
nifty-shannon-00ee72
Deploys from GitHub

Owned by myogeshchavan97's team
Last published at 7:33 AM (4 hours ago)

Want to deploy a new site without connecting to Git?
Drag and drop your site folder here

## Build and Deploy – Netlify

Click on the GitHub button to connect your GitHub repository.



app.netlify.com/start

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider
2. Pick a repository
3. Site settings, and deploy!

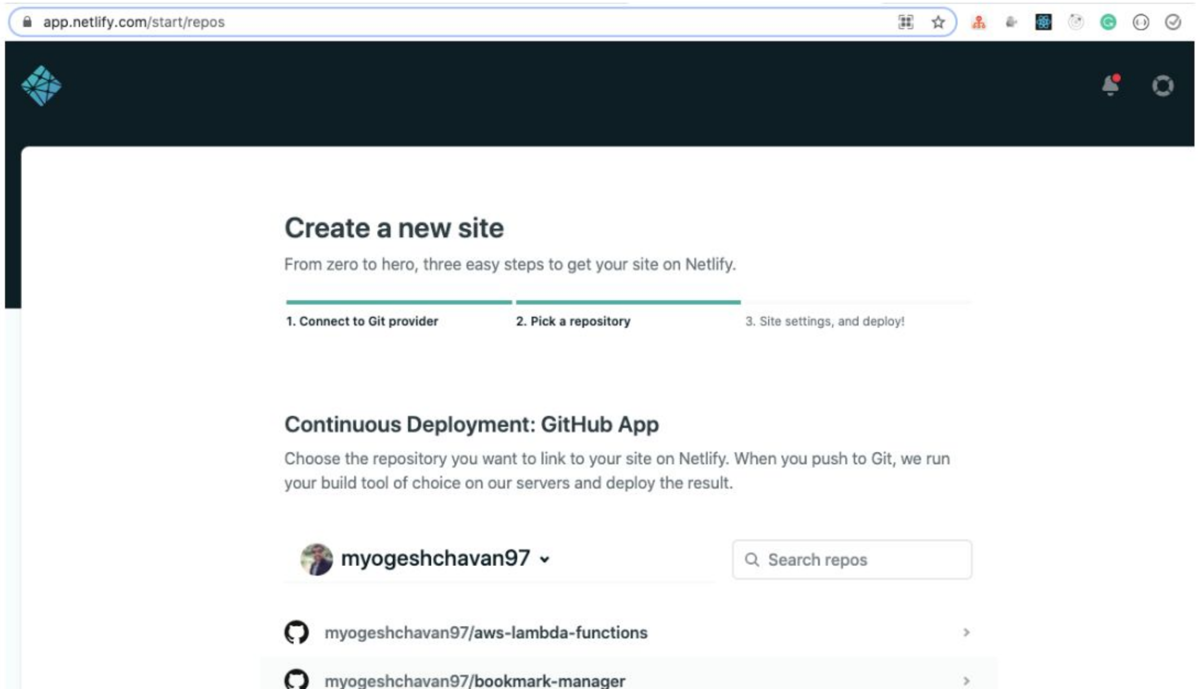
Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

You can unlock options for self-hosted GitHub/GitLab by upgrading to the Business plan.

## Build and Deploy - Netlify

It will open a new tab. Make sure the popup is enabled in your browser.



app.netlify.com/start/repos

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider
2. Pick a repository
3. Site settings, and deploy!

Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

myogeshchavan97

Search repos

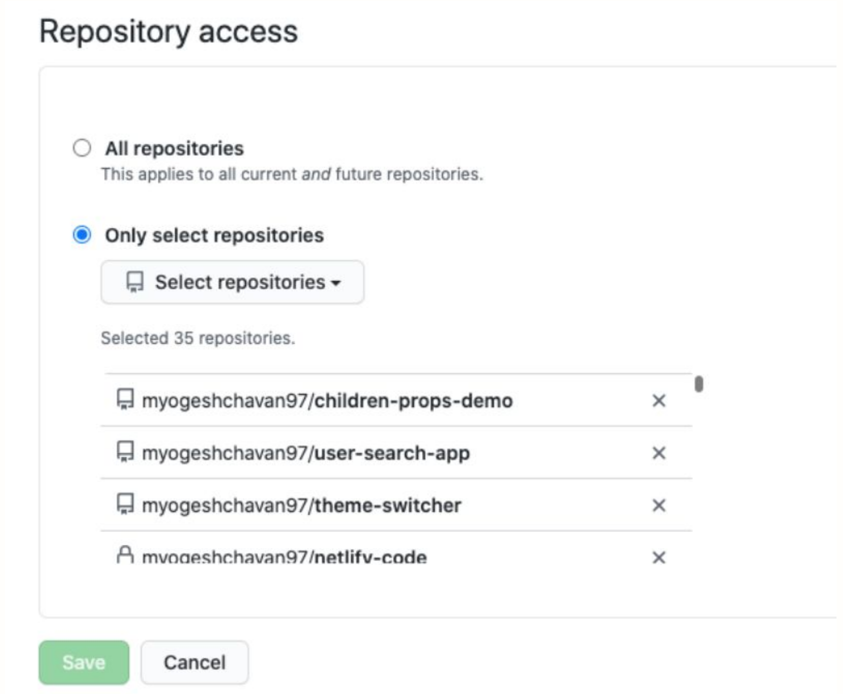
myogeshchavan97/aws-lambda-functions

myogeshchavan97/bookmark-manager

## Build and Deploy - Netlify

Search for the GitHub repository in the Search repos search box. If your repository is not getting displayed then click on the Configure the Netlify app on GitHub button at the bottom of the page.

Once clicked, scroll down on the page and click on the Select repositories dropdown and search for your repository and click on the Save button.



Repository access

○ All repositories
This applies to all current *and* future repositories.

● Only select repositories

Select repositories ▾

Selected 35 repositories.

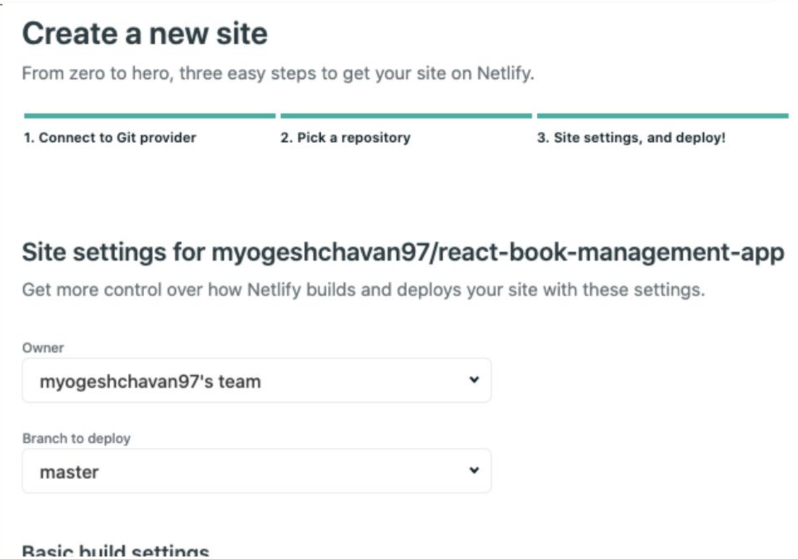
- myogeshchavan97/children-props-demo ×
- myogeshchavan97/user-search-app ×
- myogeshchavan97/theme-switcher ×
- myogeshchavan97/netlify-code ×

## Build and Deploy - Netlify

You will be redirected to the previous page showing all the available repositories.

Search for the repository you want to deploy.

Once you select the repository, you will see the following screen:



Create a new site

From zero to hero, three easy steps to get your site on Netlify.

Site settings for myogeshchavan97/react-book-management-app

Get more control over how Netlify builds and deploys your site with these settings.

Owner: myogeshchavan97's team

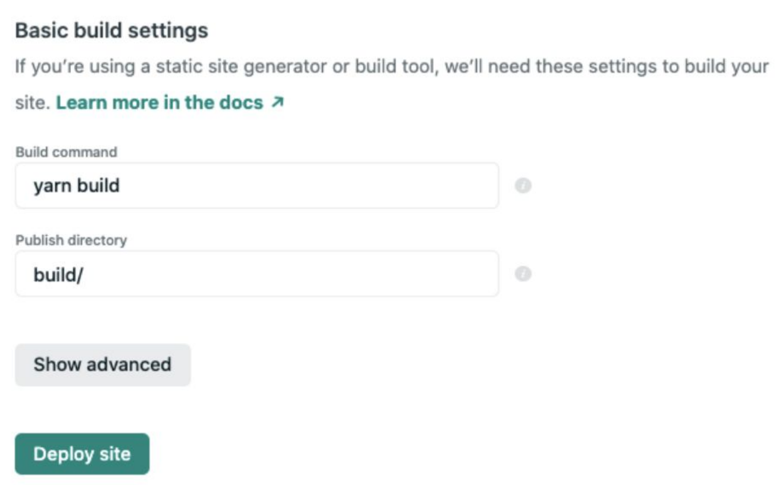
Branch to deploy: master

Basic build settings

## Build and Deploy - Netlify

Your Build command and Publish directory will be automatically populated. Make sure to enter these fields if you have a different command in package.json to build your app or those fields are not auto-populated.

Now, click on the Deploy site button. Once clicked, you will see the Site deploy in progress message.


Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site. Learn more in the docs ↗

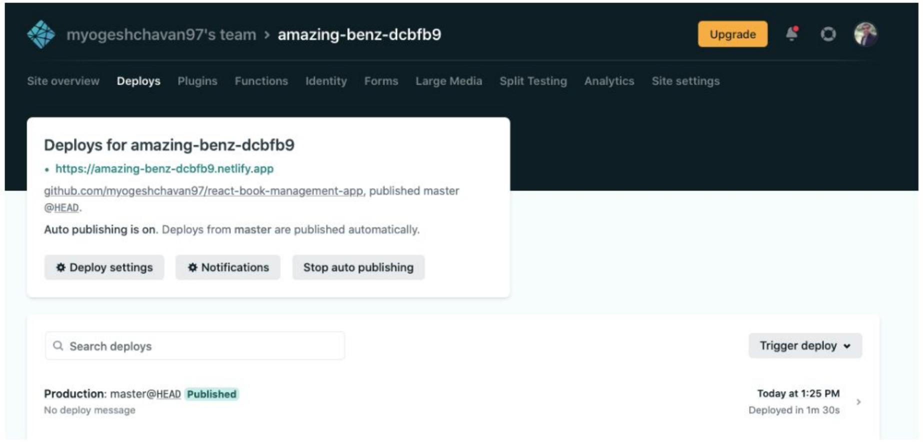
Build command: yarn build

Publish directory: build/

## Build and Deploy – Netlify

You'll have to wait a little bit while it's deploying. Once deployment is completed, you will see the following screen:

Open the link in the new tab and you will se your application deployed live.



Deploys for amazing-benz-dcbfb9

https://amazing-benz-dcbfb9.netlify.app

github.com/myogeshchavan97/react-book-management-app, published master @HEAD.

Auto publishing is on. Deploys from master are published automatically.

Production: master@HEAD Published
No deploy message

Today at 1:25 PM
Deployed in 1m 30s

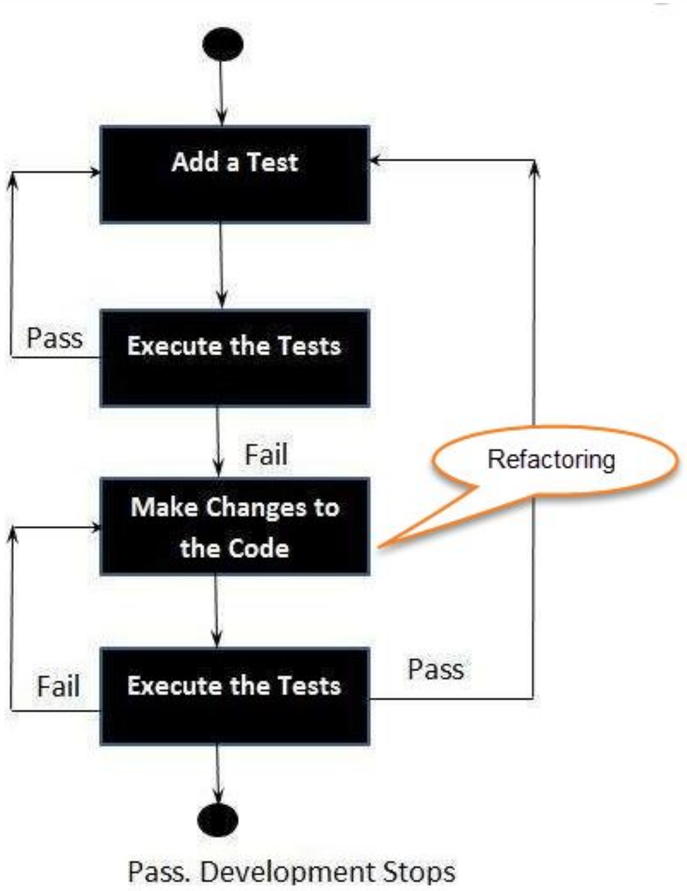
## Intro to TDD in React JS

Test-Driven Development (TDD) is a development method that utilizes repetition of a short development cycle called **Red**-**Green**-**Refactor**.

Process:

- Add a test
- Run all tests and see if the new test fails (red)
- Write the code to pass the test (green)
- Run all tests
- Refactor
- Repeat

## Intro to TDD in React JS



## Intro to TDD in React JS

Pros:

- Design before implementation
- Helps prevent future regressions and bugs
- Increases confidence that the code works as expected

Cons:

- Takes longer to develop (but it can save time in the long run)
- Testing edge cases is hard
- Mocking, faking, and stubbing are all even harder

## Test Config - Jest & Enzyme

For testing, we'll use Jest, a full-featured testing solution that comes with Create React App, and Enzyme, a powerful set of testing utilities for React.

Add Enzyme:

```
$ npm i -D enzyme
```

Enzyme requires react-test-renderer for React apps version 15.5 or greater:

```
$ npm i -D react-test-renderer @wojtekmaj/enzyme-adapter-react-17
```

## Test Config - Jest & Enzyme

Add a new file in the "src" directory titled setupTests.js:

```
import { configure } from 'enzyme';

import Adapter from '@wojtekmaj/enzyme-adapter-react-17';

configure({ adapter: new Adapter() });
```

Create React App runs the setupTests.js file before each test, so it will execute and properly configure Enzyme.

## Test Config - Jest & Enzyme

**Shallow Rendering Tests**

Shallow render tests are useful to keep yourself constrained to testing the component as a unit and avoiding indirectly testing the behavior of child components. You can find more information on shallow rendering in the Enzyme docs.

Go to: https://enzymejs.github.io/enzyme/docs/api/shallow.html

We'll start out building each component by first adding a corresponding test file and then a shallow render test using Enzyme.

## Test Config – Jest & Enzyme

src/components/App/App.spec.js

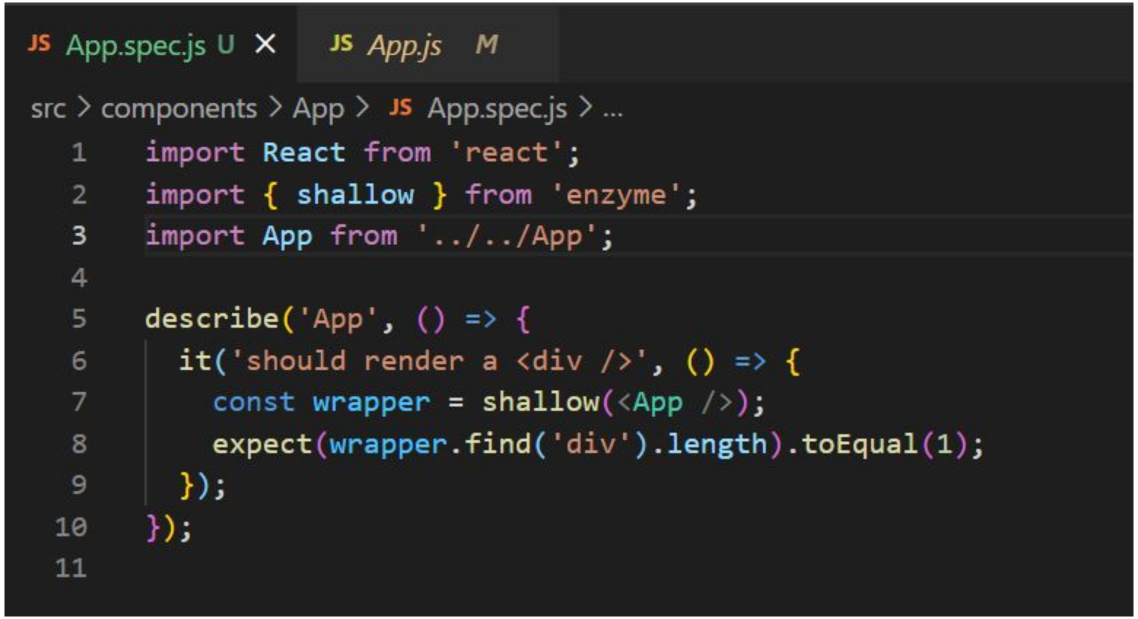
```
import React from 'react';
import { shallow } from 'enzyme';
import App from './App';

describe('App', () => {
  it('should render a <div />', () => {
    const wrapper = shallow(<App />);
    expect(wrapper.find('div').length).toEqual(1);
  });
});
```

**Shallow Rendering Tests**

Begin by adding the first failing test (red) for the App component, and then write the code for it to pass (green).

## Test Config - Jest & Enzyme

src/components/App/App.spec.js


App.spec.js U × App.js M

src > components > App > App.spec.js > ...

```
import React from 'react';
import { shallow } from 'enzyme';
import App from '../../App';

describe('App', () => {
  it('should render a <div />', () => {
    const wrapper = shallow(<App />);
    expect(wrapper.find('div').length).toEqual(1);
  });
});
```

**Shallow Rendering Tests**

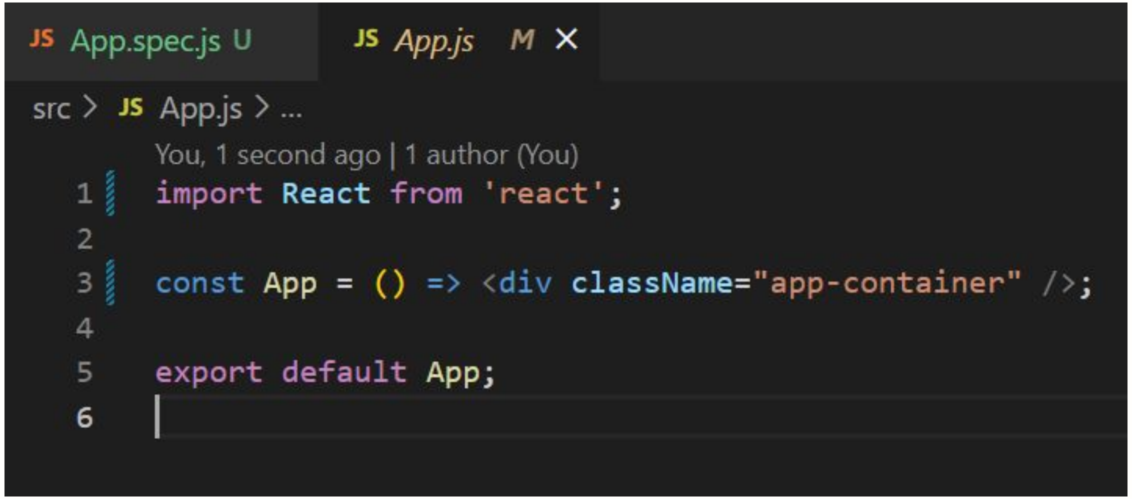
Begin by adding the first failing test (red) for the App component, and then write the code for it to pass (green).

**← Create test**

- If you run the test it will failing the test (red)

## Test Config - Jest & Enzyme

src/App.js


JS App.spec.js U    JS App.js M ×

src > JS App.js > ...

You, 1 second ago | 1 author (You)

```
import React from 'react';

const App = () => <div className="app-container" />;

export default App;
```

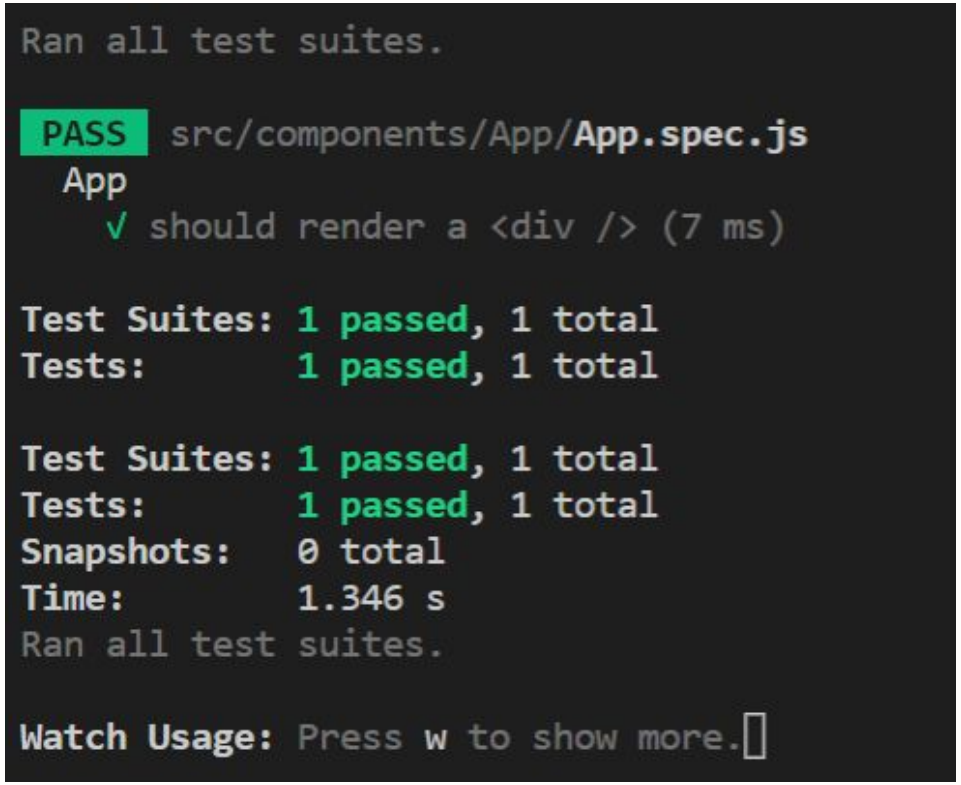
### Shallow Rendering Tests

Begin by adding the first failing test (red) for the App component, and then write the code for it to pass (green).

← **Create components based on test case**

Now, when you run the test it would pass (green).

## Test Config - Jest & Enzyme



```
Ran all test suites.

PASS  src/components/App/App.spec.js
  App
    √ should render a <div /> (7 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.346 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Run the test:

**npm test**

When the test run and return same as expected result, it would show up as the pictures.

## Thank You!

Purwadhika
Digital Technology School