**React Tutorial A Step-by-Step Guide(2) –**[Kasun Dissanayake](https://kasunprageethdissanayake.medium.com/?source=post_page-----ba62d745ed89--------------------------------) Apr 13, 2022

In this tutorial, I am going to explain the Folder Structure of React applications, What is a React component in a detailed manner, and How we can use them.

**Folder Structure of React Application**

It is really important to understand the Files and the Folders involved and How the control flows when you run the application. Because most of the developers don't know how to explain the flow of React applications.

In your project at the root level, you can see 3 folders and four files to begin.

Graphical user interface, text, application, chat or text message

Description automatically generated

React Folder Structure

Let’s start with **Files**,

* **package.json File**— This contains dependencies and the scripts which we need in the project.
* **gitignore File**— Maintain git related files
* **README.md** File

**Folders** like,

* **node\_modules** — This is the folder in which all the dependencies are installed. It is generated when you run the **create-react-app** command or when you run the **npm install.**
* **public Folder**— This folder contains **3** files and the only special file which we need is the **index.html** file. This is the only HTML file you are going to have in your application. We are building single-page applications. The view might dynamically change in the browser. But application serves this HTML file. You are not going to add any code to this file. Maybe some changes in the head tag. But not in the body tag. You want React to control the UI. For that, we have one**“div tag”**called “**root**”.

<div id="root"></div>

At the runtime react application takes over this “div tag” and it is ultimately responsible for the UI.

* Source Folder (src)— Mostly you will be working with this folder during development. The starting point of the React application is **index.js**. In the index.js we specified the root component which is the **App Component**and the DOM element which will be controlled by React. The DOM element in our code is an element with an ID of **root**. This is the element in our index.html file. We called this root element in index.html the root DOM node because everything inside it controls by React.

Graphical user interface, text, application

Description automatically generated with medium confidence

For this application, the **App Component** is rendered inside the root DOM node. You can find the app component in the **App.js**file. The App component represents the view you see in the browser. Also, you can find the **App.css**file which we can use for styling and the **App.test.js**file for Unit testing. We can find another file called **index.css** file which applies styles to the body tag. Finally, we have the **serviceWorker.js** file which uses for progressive web apps.

**What is a React Component?**

In React a component represents a part of the User Interface. Going back to the example we can say that we have 5 components in our application.

* Root (App) component — which contains whole components
* Header Component
* SideNav Component
* Main Content Component
* Footer Component

A picture containing table

Description automatically generated

**React Nesting Components**

In React, we can nest components inside within one another. This helps in creating more complex User Interfaces. The components that are nested inside parent components are called **child components**. Import and Export keywords facilitate the nesting of the components.

Components are reusable. The same component we can use with different properties to display different information. For example **side-nav Component** can be the l**eft-side-nav**or **right-side-nav**.

How does a component translate the code in our application? Component code is placed in a **JavaScript** file. For Example, an app component is placed in **App.js**. You can also have your component files in the “**.jsx**” extension. Components consist of two types.

**Component Types**

In React we have two component types.

* **Stateless Functional Component**— JavaScript Functions. It returns HTML which explains the UI.
* **Stateful Class Component**— Regular ES6 classes that extend the Component class in the react library. They must contain a render method that returns HTML.

A picture containing text, receipt, document, screenshot

Description automatically generated

As I mentioned earlier you can find AppComponent in the app.js file. Here you can see AppComponent is a **Class component**. Class named as “**App”** and it extends Component class from the React library. It contains a render method that returns some HTML. You can have multiple components in your react application. More complex applications have more components.

**Functional Components**

Functional components are just **JavaScript functions**. They can optionally receive the object of **properties** which is referred to as **props** and returns HTML which describes the UI.

Text

Description automatically generated with low confidence

The HTML is also known as **JSX**.

Let's create a functional component first. Go to your project and remove all HTML code except the outer **div tag** in the **App.js**file. Now let’s create our own component. I am going to output a message from this newly created component. The component is nothing but a JavaScript file. Create a new folder called “**components**”. Inside that folder create a file called **Hello.js**.

Within the file first import React. Next, create a new function and return HTML elements.

import ***React*** from "react";  
function Hello() {  
 return <h1>  
 Hello Kasun Dissanayake   
 </h1>  
}

You have created your first functional component. But here “**Hello Kasun Dissanayake**” is not going to be rendered in the browser because the Hello component is not connected with our application. We have to **export** our Hello function from **Hello.js** and **import** it into **App.js** and then include it in the App component.

export default Hello; //Hello.js   
import Hello from "./components/Hello"; //App.js

To include the **Hello component** in the **App component** we simply specify the component in the custom HTML tags. There is no content between the tags.

function App() {  
 return (  
 <div className="App">  
 <Hello />  
 </div>  
 );  
}

Now run the application and you should be able to see Hello Kasun Dissanayake in the browser.

Text

Description automatically generated

Functional Component show in the Browser

Your first functional component is up and running.

Let’s rewrite this Hello function with **Arrow functions**syntax.

**An arrow function** expression is a compact alternative to a traditional [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function), but is limited and can’t be used in all situations.

There are differences between *arrow functions* and *traditional functions*, as well as some limitations:

* Arrow functions don’t have their own bindings to [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) or [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), and should not be used as [methods](https://developer.mozilla.org/en-US/docs/Glossary/Method).
* Arrow functions don’t have access to the [new.target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target" \t "_blank) keyword.
* Arrow functions aren’t suitable for [call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), [apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) and [bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) methods, which generally rely on establishing a [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope).
* Arrow functions cannot be used as [constructors](https://developer.mozilla.org/en-US/docs/Glossary/Constructor).
* Arrow functions cannot use [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield), within its body.

const Hello = () => <h1> Hello Kasun Dissanayake</h1>

Now the output still works as expected.

**Class Components**

Class components basically **ES6 classes**. Similar to functional components Class components also can optionally receive **props** as input and return **HTML**. Apart from the props Class components can also maintain a private internal state. It can maintain some information that is private to that component and use that information to describe the User Interface.

Let's create **Hello Component** using Class components. Create a new file called **Welcome.js**. When we create a class component we need to import two things.

* Import React
* Import “Component” Class from React

import ***React***, {Component} from "react";

Now create the class Welcome. To make this class a React component there are 2 simple steps.

* Extends the component class from React.
* The class has to implement a render method that will return null or some HTML.

class Welcome extends Component {  
 render() {  
 return (  
 <div>  
 Welcome Kasun Dissanayake  
 </div>  
 );  
 }  
}

Now you have created your own Class component. Export this component and import it into App.js to connect with our application.

export default Welcome; //Welcome.js  
import Welcome from "./components/Welcome"; //App.js//inside the App.js function add Welcome component  
function App() {  
 return (  
 <div className="App">  
 <Hello />  
 <Welcome />  
 </div>  
 );  
}

Run the application and you will be able to see the Welcome component in the browser.

Graphical user interface, text, application

Description automatically generated

Now let's find what is the advantage of one over the other and when exactly should we use which component type.

**Functional Components vs Class components**

Functional Components are,

* Simple functions accepting props, returning the function declaration.
* Use functional components as much as possible.
* Absence of “this” keyword.
* Solution without having a state — If you have a number of components each having its own state maintenance and debugging your application is kind of difficult.
* Mainly responsible for the User Interface.
* Also known as Stateless/ Dumb/ Presentational components

Class Components are,

* More feature-rich components.
* Can also maintain their own private data called States.
* Can contain complicated UI logic.
* Provide lifecycle hooks.
* Also known as Stateful/ Smart/ Container components.

I hope you got a good understanding of the **Folder Structure** of React applications, What is a React component and How we can use them. We will discuss **Hooks and JSX** in the next article.