Hyperiondev

TASK

Function Components

Visit our website

Introduction

WELCOME TO THE FUNCTION COMPONENTS (REACTJS) TASK!

In this module, you will learn more about function components used in ReactJS applications—the theory behind function components in terms of their functionality and usage, and what the differences are between function components and class components.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev**, where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS A FUNCTION COMPONENT?

A function component is a JavaScript function that takes props and returns a React element. Since the roll-out of React Hooks, writing function components has been the traditional way of writing React components in advanced applications.

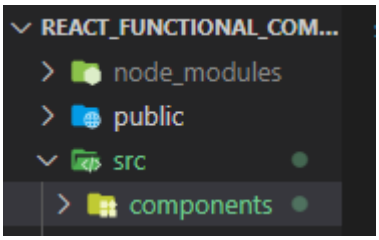
EXAMPLES OF FUNCTION COMPONENTS

Today we will create a regular ReactJS application using the create-react-application node library to demonstrate the initialization of function components and usage thereof, and how function components will help you develop modular UI components as a developer.

First, we will run the following command in the folder of your choice to create our ReactJs application:

```
create-react-app react_functional_components
```

Now, we will create a components folder in our src or source directory, as indicated in the following image:



In the next step, we will create a new Javascript file called **`Welcome.js`**, and implement a standard function component that will pass an object called props (short for properties).

```
import React from "react";
function Welcome(props) {
  return (
    <div>
      <h1>Hello World, {props.name}</h1>
    </div>
  );
}
export default Welcome;
```

To implement our newly created function component, we will import the function into our `App.js` file and initiate some examples:

```
import "./App.css";
import Welcome from "./components/Welcome";

function App() {
  return (
    <div className="App">
      <Welcome name="Joe Soap" />
      <Welcome name="John Handcock" />
    </div>
  );
}

export default App;
```

What we are doing here is we are displaying the name of a specific user using props. We have included the Welcome component in `App.js` and passed its info/name as an attribute. The Welcome component will receive this info as props which is an object containing a name field inside it, and we can use this information in the Welcome component wherever we want.

To check whether everything is up and compiling correctly, you should see the following in your web browser.

Hello World, Joe Soap

Hello World, John Handcock

Well done if everything is working! This is just one of the examples we will show you during this module on how to implement modular function components.

Now to emphasise the modularity of a function component, we will pass additional props to our function components and add event handling into our function by implementing a button that will display the age of each user's implementation. The event handler we will execute is the `onClick` function. When the button is clicked on or pressed, an alert will appear with the user's age.

First, we will implement the button into the JSX / HTML of our **`Welcome.js`** file and add a function that will alert the user of their age.

```
import React from "react";

function Welcome(props) {
  const displayAge = () => {
    alert(props.age);
  };

  return (
    <div>
      <h1>Hello World, {props.name}</h1>
      <button onClick={displayAge}>Display The User's Age</button>
    </div>
  );
}

export default Welcome;
```

Then adjust the props passes to our function components implementations by adding the age of the user.

```
function App() {
  return (
    <div className="App">
      <Welcome name="Joe Soap" age="39" />
      <Welcome name="John Handcock" age="52" />
    </div>
  );
}
```

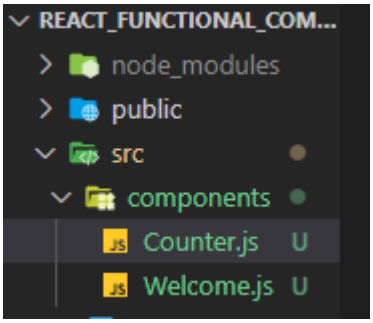
By now, you will be able to see the advantages of function components and how this function implementation can help you in future development, and see the alternatives to using conventional class components.

The following tutorial example will show you how to use a function component using React Hooks!

The first thing you might ask yourself is what React Hooks is. The answer is that React hooks are powerful functions that can be used in a function component to manipulate the state of the component without turning it into class components.

We will explain the differences between class and function components later in this module, so don't stress about them at present.

To implement our function component using React Hooks we first need to create a `Counter.js` file inside the component folder we created earlier in this tutorial, as indicated below.



Now to make this exercise fun and ensure our function looks pretty, and to reiterate the advantages of using function components, we will add some flair to our component. We are using React-Bootstrap and other CSS rules.

To Install react-bootstrap into our application, we need to run the following command.

```
npm install react-bootstrap bootstrap
```

And include the following import into our `index.js` file.

```
import "bootstrap/dist/css/bootstrap.min.css";
```

Now that react-bootstrap is installed, we can implement the following code. NB: To ensure you don't have any runtime issues with react-bootstrap, ensure you have the import mentioned above implemented into your `index.js` file.

Example of our `Counter.js` function component.

```
import React from "react";
import { Button, Stack } from "react-bootstrap";
import { useState } from "react";

function Counter() {
  const [counter, setCounter] = useState(0);

  const addToCounter = () => {
```

```
    let updateCounter = counter + 1;
    setCounter(updateCounter);
  };

  const minusFromCounter = () => {
    let updateCounter = counter - 1;
    setCounter(updateCounter);
  };

  return (
    <div>
      <h1 style={{ marginTop: "10px" }}>Counter value: {counter}</h1>
      <Stack
        direction="horizontal"
        gap={2}
        style={{
          display: "flex",
          justifyContent: "center",
          alignContent: "center",
          margin: "50px",
        }}
      >
        <Button onClick={addToCounter}>Increment</Button>
        <Button onClick={minusFromCounter}>Decrement</Button>
      </Stack>
    </div>
  );
}

export default Counter;
```

DIFFERENCES BETWEEN FUNCTION COMPONENTS AND CLASS COMPONENTS

Function Components	Class Components
A function component is a pure JavaScript function that accepts props as an argument and returns a React element(JSX).	A class component requires you to extend from React.Component and create a render function which returns a React element.
There is no render method used in function components.	It must have the render() method returning JSX (which is syntactically similar to HTML).

Function components run from top to bottom; once the function is returned, it can't be kept alive.	A class component is instantiated, and a different life cycle method is kept alive and being run and invoked depending on the phase of the class component.
Also known as Stateless components, they accept data, display it in some form, and are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in function components.	React lifecycle methods (for example, componentDidMount) can be used inside class components.

Hooks can be easily used in function components to make them Stateful.
example: const [name,SetName]= React.useState(‘ ‘)
	Different syntax is required inside a class component to implement hooks.
	example: constructor(props) {
	super(props);
	this.state = {name: ‘ ‘}
	}

Constructors are not used.	A constructor is used as it needs to store state.

Compulsory Task 1

- Find a published web page that you particularly like (not something you’ve already made). It could be anything from Netflix, Takealot, UCook, or any other website you like!
- Create a ReactJS application that’s a clone of the website. Don’t worry about adding state changes to your application.
- Using only function components and custom CSS rules, try to recreate the webpage as best you can.

- Ensure at least one function component passes props to the applicable component and renders the information suited to your webpage.

- Add the URL of the webpage you are recreating as a link at the bottom of your webpage.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.