



TASK

React - Components

Visit our website

Introduction

WELCOME TO THE REACT - COMPONENTS TASK!

In this task, you will learn about components used in React applications. Components are the fundamental building blocks of user interfaces, providing independent and reusable UI elements. You will also gain insights into functional components, exploring their functionalities and practical applications.

WHAT IS A FUNCTIONAL COMPONENT?

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

EXAMPLES OF FUNCTIONAL COMPONENTS

Today we will create a regular application using the **create-react-application** node library to demonstrate the initialisation 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 command line in the folder of your choice to create our application:

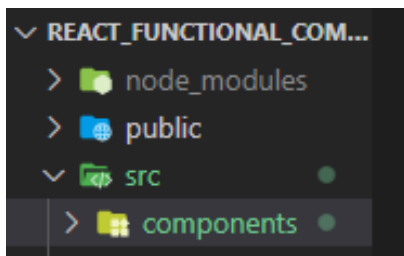
```
npx create-react-app react_functional_components
```

npx is a command-line tool that comes bundled with npm (Node Package Manager). It stands for "Node Package Execute" and is used to execute Node.js packages without the need to install them globally on your system.

The code snippet below shows the **create-react-app** command being run in the command line in the **reactexample** folder of the **C:** Drive. See just below:

```
C:\reactexample> npx create-react-app react_functional_components
```

Next, create a new folder in the **src** directory and name it **'Components'**, as indicated in the following image:



In the next step, we will create a new Javascript file called **Welcome.js**, in the **Components** directory and implement a standard functional component that will pass an object called **props** (short for properties). Props might remind you of HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, and functions. In this example, we will be passing a **'name'** prop to the component, which is a string value. The name prop will be accessed using standard [JavaScript Object notation](#). The Object is called **props** and the property we are accessing is called **name**. As such, the object notation for accessing the property of the props object is **{props.name}**

To write this component we will be using [JSX](#). JSX enables you to write HTML-like markup within a JavaScript file, keeping rendering logic and content seamlessly integrated. Sometimes, you may want to include a little JavaScript logic or reference a dynamic property inside that markup. In such cases, you can use curly braces in JSX, providing a gateway to JavaScript functionality. If you are unsure about JSX syntax, you can use a [converter](#) tool to convert HTML to JSX code.

```
//Importing React
import React from "react";

//Create the Welcome Component
function Welcome(props) {
  return (
    <div>
      {/* This h1 element uses JSX to display the name property of the props
object */}
      <h1>Hello World, {props.name}</h1>
    </div>
  );
}
export default Welcome;
```

IMPORTING AND EXPORTING COMPONENTS

You can declare many components in one file, but large files with many components can get difficult to navigate and maintain. Common practice in React applications is to create each component in its own file, export the component (as can be seen in the code example above) and then import the component into another file.

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

```
// Import CSS files for styling purposes
import './App.css';
// importing the Welcome Component
import Welcome from './components/Welcome';

function App() {
  return (
    <div>
      {/* Display an instance of the Welcome component
      passing a prop name="Joe Soap" */}
      <Welcome name="Joe Soap" />
      {/* Display a second instance of the Welcome component
      passing a prop name="John Smith" */}
      <Welcome name="John Smith" />
    </div>
  );
}

export default App;
```

Here we are displaying the name of a specific user using props. We have included the **Welcome** component in **App.js** and passed its **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, open the command line and run the command **npm run start** in the **react_functional_components folder**, which we created when we ran the **create-react-app** command. You should see the following in your web browser:

Hello World, Joe Soap

Hello World, John Smith

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

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

```
function App() {  
  return (  
    <div>  
      {/* Display an instance of the Welcome component  
        passing 2 props  
        name="Joe Soap"  
        age="39" */}  
      <Welcome name="Joe Soap" age="39" />  
      {/* Display a second instance of the Welcome component  
        passing two props  
        name="John Smith"  
        age=39 */}  
      <Welcome name="John Smith" age="52" />  
    </div>  
  );  
}
```

By now, you will start to see the advantages of components over simple React elements and how the implementation of functions can help you in future development.

STYLING REACT COMPONENTS

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 style to our components. There are several ways to style React components. Some of the more popular ways to style your components are described below.

CSS Styling: Standard [CSS styling](#) can be applied to React components using CSS selectors, with a couple of notable exceptions, the HTML “**class**” attribute is replaced in React with “**className**”. For example:

```
<div className="App">
```

When applying inline styling to components, remember that inline CSS is written in a JavaScript object. For properties with two names, such as **background-color**, you should use camel case syntax. Instead of “**background-color**”, use “**backgroundColor**”. For example:

```
<h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
```

Library of React UI components: Some common libraries include [React Bootstrap](#) and [Material-UI](#) that contain pre-styled components that can be imported and used in your React application.

Styled Components: [Styled Components](#) allow you to use actual CSS syntax inside your components. Styled Components is a variant of “CSS-in-JS”, which utilises tagged template literals (a recent addition to JavaScript) to style your components.

In this task, we will be concentrating on **React Bootstrap** to style our components.

To Install **react-bootstrap** into our application, we need to run the following command in the command line terminal in the **react_functional_components** folder that was created when we ran the create-react-app command. This folder contains the **package.json** file, which NPM uses to keep track of all the NPM packages installed in your application:

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

We'll then need to include the following import into our **index.js** file ([Bootstrap](#) is a frontend toolkit that can be used in standard HTML and JS applications. The **react-bootstrap** library is built as a React extension of standard Bootstrap):

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

Next, add the following **link** to the `<head> </head>` tag of the `index.html` file in the `/public/` folder of your react application. Copy-paste the stylesheet `<link>` into your `<head>` before all other stylesheets to load the Bootstrap CSS.

```
<!-- Latest compiled and minified CSS -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
```

This file links to the Bootstrap CSS stylesheet file from the cloud that contains CSS code that will style the components we import.

Many bootstrap components require the use of JavaScript to function. Specifically, they require jQuery, Popper.js, and Bootstrap's own JavaScript plugins, which are included in the `<script>` we are linking to below.

Add the following link to the Bootstrap JS Script to the `<head> </head>` tag of the `index.html` file:

```
<!-- Latest compiled JavaScript -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
```

The `index.html` file should now look like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Latest compiled and minified CSS -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap
        .min.css"
      rel="stylesheet"
    />
    <!-- Latest compiled JavaScript -->
    <script
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap
        .bundle.min.js">
    </script>
```

```

<meta charset="utf-8" />
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="theme-color" content="#000000" />
<meta name="description" content="Web site created using create-react-app"
/>
<link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
<!--
  manifest.json provides metadata used when your web app is installed on a
  user's mobile device or desktop. See
  https://developers.google.com/web/fundamentals/web-app-manifest/
-->
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<!--
  Notice the use of %PUBLIC_URL% in the tags above.
  It will be replaced with the URL of the `public` folder during the
  build. Only files inside the `public` folder can be referenced from the
  HTML.

  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
  work correctly both with client-side routing and a non-root public URL.
  Learn how to configure a non-root public URL by running `npm run build`.
-->
<title>React App</title>
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
</html>

```


Now that we have installed, linked, and imported all of the React-Bootstrap requirements, let's import some Bootstrap components and look at how they differ from standard HTML components:

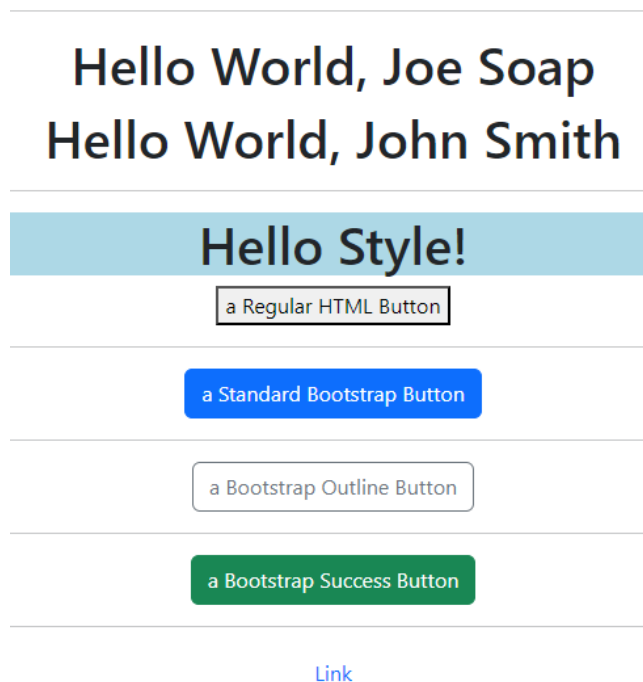
```
import './App.css';
import Welcome from './components/Welcome';
import "bootstrap/dist/css/bootstrap.min.css";
import Button from 'react-bootstrap/Button';

function App() {
  return (
    <div className="App">
      <hr />
      <Welcome name="Joe Soap" age="23" />
      <Welcome name="John Smith" age="39"/>
      <hr />
      /* Adding standard CSS styling to an HTML component in React */
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      /* A Standard HTML Button Component */
      <button> a Regular HTML Button</button>
      <hr />
      /* A Bootstrap Button Component */
      <Button>a Standard Bootstrap Button</Button>
      <hr />
      /* These are variants of the same Bootstrap button */
      <Button variant="outline-secondary">a Bootstrap Outline Button</Button>
      <hr />
      <Button variant="success">a Bootstrap Success Button</Button>
      <hr />
      <Button variant="link">Link</Button>

    </div>
  );
}

export default App;
```

If you open up the React Application in your browser, it should look similar to this:




The first Button is a standard HTML button without styling.

Next, we have a Bootstrap Button component.

The next three Bootstrap buttons are the same component with specific styling applied by using the “**variant**” keyword.

Bootstrap Grid: Bootstrap’s grid system uses a series of containers, rows, and columns to lay out and align content. It’s built with **Flexbox** and is fully responsive. Below is an example of how the grid comes together.



Extra resource

New to or unfamiliar with Flexbox?
[Read this CSS Tricks flexbox guide](#) for background, terminology, guidelines, and code snippets.

First, let’s add a class called **red-border** to our **App.css** file to make the rows and columns easier to see:

```
/* red-border Class */
.red-border {
  border: 1px solid red;
  padding: 5px;
  align-self: center;
  margin: 15px
}
```

Next, we'll create a `LayoutExample.js` component in the **Components** folder as seen in the code below.

To use the React Bootstrap grid system we need to **import** `Container`, `Row`, and `Col`:

```
import Container from 'react-bootstrap/Container';
import Row from 'react-bootstrap/Row';
import Col from 'react-bootstrap/Col';

// Create the LayoutExample component
function LayoutExample() {
  return (
    // Create an instance of the Container component which will
    // automate the layout
    <Container>
      {/* The Container component can have standard HTML elements as children */}
      <h2>Layout Example</h2>
      <hr />
      <h3> Rows and Columns Auto Layout </h3>
      {/* Using the Row & Col components we can create a Grid Layout */}
      <Row>
        <Col className="red-border">Row 1 : Column 1 of 2</Col>
        <Col className="red-border">Row 1 : Column 2 of 2</Col>
      </Row>
      <Row>
        <Col className="red-border">Row 2 : Column 1 of 3</Col>
        <Col className="red-border">Row 2 : Column 2 of 3</Col>
        <Col className="red-border">Row 2 : Column 3 of 3</Col>
      </Row>
      <h3> Rows and Columns Specified sizes </h3>
      <Row>
        {/* We specify the size of the first column */}
        <Col xs={3} className="red-border">Row 1 : Column 1 of 2</Col>
        {/* The second column fills up the remaining space */}
        <Col className="red-border">Row 1 : Column 2 of 2</Col>
      </Row>
      <Row>
```

```

        {/* Each of the Columns has a fixed size */}
        <Col xs={2} className="red-border">Row 2 : Column 1 of 3</Col>
        <Col xs={3} className="red-border">Row 2 : Column 2 of 3</Col>
        <Col xs={4} className="red-border">Row 2 : Column 3 of 3</Col>
      </Row>
    </Container>
  );
}

export default LayoutExample;

```

Let's take a look at Bootstrap [Stacks](#). Stacks are shorthand helpers that build on top of Bootstrap's flexbox utilities to make component layout faster and easier. Stacks are vertical by default and stacked items are full-width by default. Use the **gap** prop to add space between items. Use **direction="horizontal"** for horizontal layouts. Stacked items are vertically centred by default and only take up their necessary width.

Create a file called **StackExample.js**, as below, in the **Components** folder:

```

import Stack from 'react-bootstrap/Stack';

function StackExample() {
  return (
    <div>

      <h3 > Vertical Stack</h3>
      {/* Stacks allow for a quick Grid Layout
      without the need for multiple Row & Col components */}

      {/* a Stack component with a standard vertical layout */}
      <Stack className="red-border" gap={2}>
        <div className="red-border">First item</div>
        <div className="red-border">Second item</div>
        <div className="red-border">Third item</div>
      </Stack>

      <h3>Horizontal Stack</h3>
      {/* {/* a Stack component with a horizontal layout */}
      <Stack className="red-border" direction="horizontal" gap={2}>

```

```

    <div className="red-border">First item</div>
    <div className="red-border ">Second item</div>
    <div className="red-border ">Third item</div>
  </Stack>
</div>
);
}

export default StackExample;

```

Now let's import these components into the **App.js** file to display them:

```

import "../App.css";
import "bootstrap/dist/css/bootstrap.min.css";
// Importing the components
import LayoutExample from "../components/LayoutExample.js";
import StackExample from "../components/StackExample";

function App() {
  return (
    <div className="App">
      <hr />
      {/* display the LayoutExample component */}
      <LayoutExample />
      <hr />
      {/* display the StackExample component */}
      <StackExample />
    </div>
  );
}

export default App;

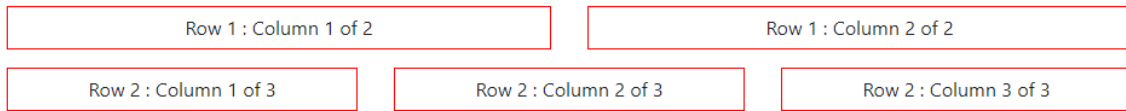
```

Now run the React application with the **npm run start** command in your terminal and open the application in your browser.

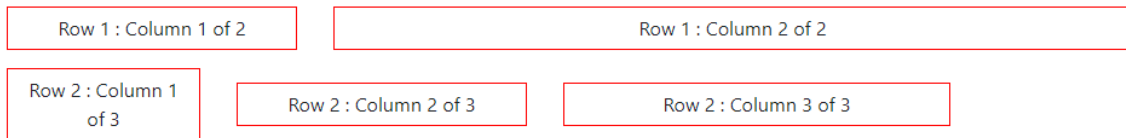
The result should look similar to this:

Layout Example

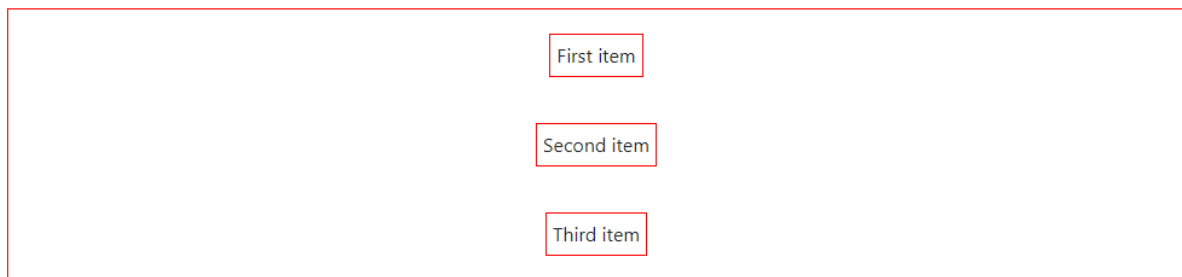
Rows and Columns Auto Layout



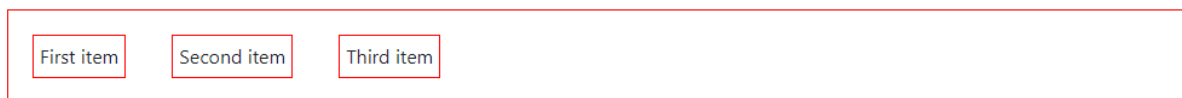
Rows and Columns Specified sizes



Vertical Stack



Horizontal Stack



DIFFERENCES BETWEEN FUNCTIONAL COMPONENTS AND CLASS COMPONENTS



Take note:

Class components were a type of component that was widely used in older versions of the React library. However, with the introduction of React Hooks, functional components have become the preferred way of writing components due to their simplicity and better code organisation. As a result, class components have been **deprecated** and are no longer receiving updates. However, if you encounter them while working with legacy React projects, you can refer to [this link for more information](#).

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 that 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.	The class component is instantiated and a different lifecycle method is kept alive and is 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: <code>const [name, setName]= React.useState('')</code>	Different syntax is required inside a class component to implement hooks. example: <code>constructor(props) { super(props); this.state = {name: ''} }</code>
Constructors are not used.	A constructor is used as it needs to store the state.

Instructions

In this assignment, you will demonstrate your understanding of functional components in React and the use of custom CSS rules to recreate an existing website of your choice. You have the creative freedom to choose any website you find interesting and engaging.

Compulsory Task 1

Follow these steps:

- **Choose a website:** Find a published web page that you particularly like, such as Netflix, Twitter, Instagram, Takealot, UCook, [HyperionDev](#), or any other website that inspires you. Ensure that the website you choose is feasible to **recreate** within the scope of this task. Keep in mind that you are tasked to recreate **a web page**, not the entire website.
- **Create a clone:** Create a React application, using `create-react-app`, that serves as a clone of the chosen website. Don't worry about adding state changes to your application.
- **Functional components:** Analyse the chosen website and identify its major components and sections. Implement functional components for different sections of the website, such as headers, navigation bars, content sections, and footers.
- **Pass props:** Choose at least one functional component to pass props to another applicable component. This will allow you to render specific information tailored to your webpage.
- **Styling:** Apply custom CSS rules and possibly Bootstrap components to style your website and make it visually similar to the original website.
- **Link to website:** Include the URL of the website you are recreating as a link at the bottom of your webpage, to provide a reference to the original source for comparison.



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.