**Front End Development using React JS**

Use Cases Manual

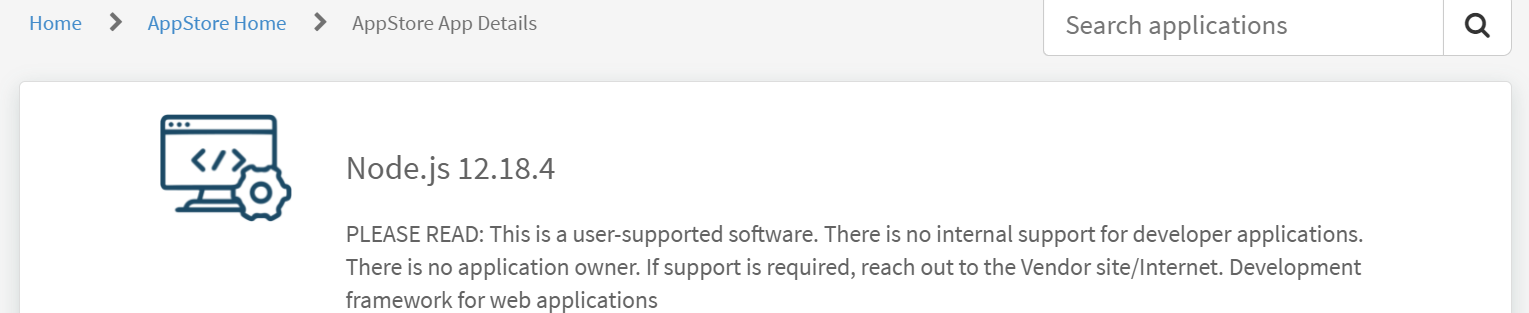
# Sandbox Link [React](https://share.percipio.com/cd/CLjSwmFlb)

# **Agenda**

|  |  |  |
| --- | --- | --- |
| S.No | Description | Page No |
| Lab-1 | Installation of ReactJS | 3 |
| Lab-2 | Introduction to the directory structure | 4 |
| Lab-3 | ReactJS Components | 6 |
| Lab-4 | Approach1 Creating Functional Component | 13 |
| Lab-5 | Approach2 (ECMAScript 6) syntax for creating a functional component | 15 |
| Lab-6 | Using Props with Class Components | 18 |
| Lab-7 | Calling Child component inside Parent component | 20 |
| Lab-8 | Passing some value from Parent to Child component | 21 |
| Lab-9 | Using Props with Functional Components | 22 |
| Lab-10 | State Example | 23 |

# **Lab-1: Installation of ReactJS**

Install [**Node.js**](http://node.js/) and [**npm**](https://www.npmjs.com/get-npm) globally on your machine.[App store link](https://optum.service-now.com/euts_intake?id=euts_appstore_app_details&appKeyId=29175)



Note :

npm config edit

registry=https://repo1.uhc.com/artifactory/api/npm/npm-virtual

To create any, React application, run the following code in your terminal:

npx create-react-app todoapp

cd todoapp

npm start

# (Note: If you’ve previously installed create-react-app globally via **npm install -g create-react-app**, we recommend you to uninstall the package using **npm uninstall -g create-react-app** to ensure that npx always uses the latest version)

# And make sure you should not use uppercase letters for your application name.

# So let’s check out our browser and we can see our application is up and running on [**http://localhost:3000**](http://localhost:3000/) locally.

# **Visual Studio Code** — as an IDE.

# **Lab-02 Introduction to the directory structure**

# Let’s walk through the directory structure that React CLI created for us.

# Open this newly created application in VS Code.

# Image of Directory Structure of React Application

# As you can see the very first directory is **node\_modules** and this is having all the library dependency of package.json.

# Now check this **public**directory.

# In this directory our important file is **index.html**. As you know React is a single page application so it is the only one Html or you can say a single file that gets rendered when we run the application.

# Also, focus on the line <div id=” root”></div>. This <div> will be replaced with the root component.

# Inside this directory, you will also find **manifest.json**. This file is for PWAs — Progressive Web Apps.

# **src**directory. This will contain all our React code.

# This is having **App.js**, which is the first component, which will get rendered when we run our application

# **index.js,** is having ReactDOM.render() method who actually rendering our App.js .

# ReactDOM.render(<App />, document.getElementById(‘root’));

# This render method will actually render the index.html’s root div.

# It will replace this root div with the app component. This is how your first component will work.

# **serviceWorker.js**. And this service worker is for PWAs — Progressive Web Apps.

# **package.json**. It contains all the dependencies of our React application.

# **Lab -3: ReactJS Components**

Basically, Components have two types:

1. Class Components
2. Functional Components

**Let us create a class-based component.**

we will create all components inside **src** directory.

* Now right-click on your component directory and create a new file and call it as **Demo.js**.
* Inside this Demo.js, we will create a class component. For this we need to follow below **steps**:

i) First, you need to **import** the React Package from the React Library

import React, {Component} from "react";

ii) Now, Create a **Class**and give it any name. This class should extend a Component interface. And now create a render() method in this class.

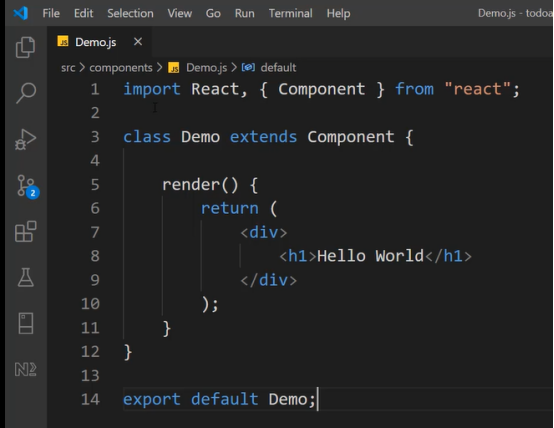
render() method will be automatically called by ReactJs when required.

class Demo extends Component {  
 render() {  
 return (  
 <div>  
 <h1>Hello World</h1>  
 </div>  
 );  
 }  
}

Whatever you want to display on your component, you need to write them inside this render() method.

iii) The last step is to **export**this Demo Component.

export default Demo;



Now we need to display this Demo Component on our browser also.

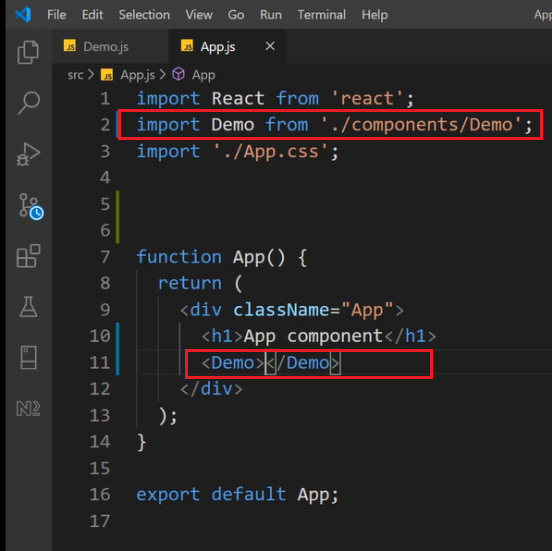
For that, we have to import/call this Demo Component on an **App.js**, because App.js is the only component that will get rendered when we run our application.

So open the **App.js** file and write a below statement:

import Demo from './components/Demo';

*Note that it is not necessary to have a name “Demo” in this import statement, you can provide any name to this.*

*And call a <Demo> </Demo> component inside the return method.*



And you can see “Hello World” on your browser.

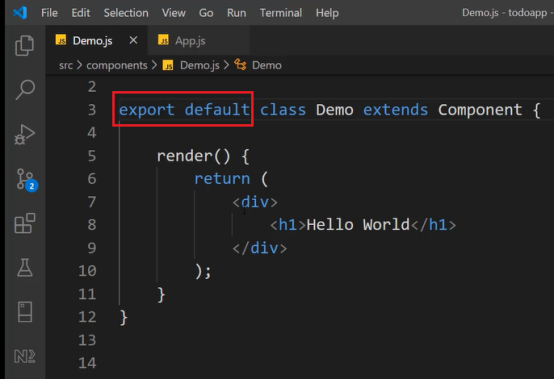
There is also another way to export this Demo component. We can write export as an inline statement too.

In **Demo.js,** remove that entire line ‘export default app;’ and write ‘export default’ before the ‘class’ keyword.

export default class Demo extends Component {  
.........

}

And this will work the same as before.



let me tell you why this ‘**default**’ is required.

Remember we can provide any name to import this component in **App.js**?

This **default**keywordis responsible for that. If we use this **default**with our class declaration then we can import this component with any name in **App.js**.

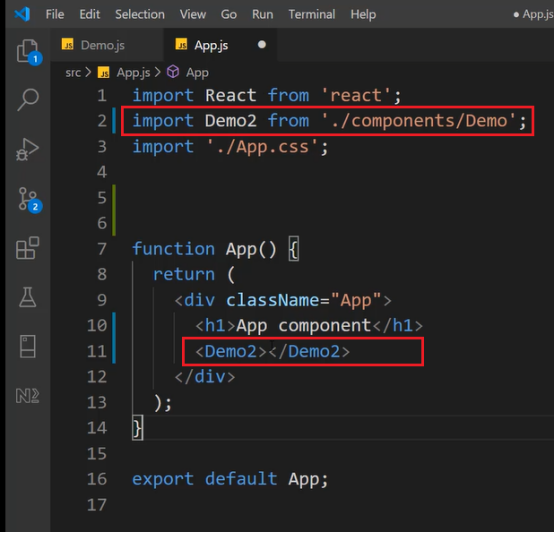
For example, try to import this with any other name except ‘Demo’, like this,

For example, try to import this with any other name except ‘Demo’, like this,

import **Demo2** from './components/Demo';

And inside return() method,

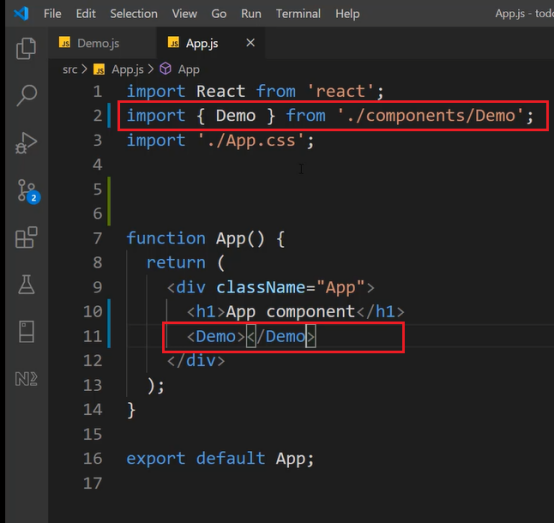
function App() {  
 return (  
 <div className=”App”>  
 <h1>App component</h1>  
 **<Demo2></Demo2>** </div>  
 );  
}



Now save this and run the application and you can see the same ‘Hello World’ as before.!

*Now if you remove this ‘****default****’ keyword and re-run this application*

*So to import a component who does not have this****default****keyword, we need to use curly braces****{ }****in the import statement. Inside this curly braces, we have to write what we are exporting from Demo.js.*



And you can see the same “Hello World” as before.

# **Lab-4 Approach1 Creating Functional Component**

Let us create a functional component.

Create a new file as **Demo1.js** inside the component directory.

Now inside this **Demo1.js,**we first need to **import** the React Package from the React Library.

import React {Component} from 'react';

Now make a Demo1 method using function keyword as below and return some HTML text to display on browser.

function Demo1() {  
 return (  
 <div>  
 <h1>Hello From function Component</h1>  
 </div>  
 );  
}

And now **export**this Demo1 Component.

export default Demo1;

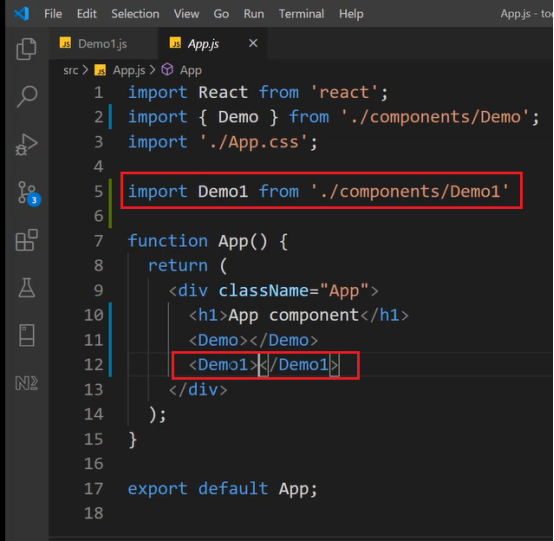
*(Note: the working of ‘default’ keyword is the same for both the types of components)*

Now on **App.js,**we need to import this newly created **Demo1**component.

open the **App.js** file and write a below statement:

import Demo1 from './components/Demo1';

And also call this with <Demo1></Demo1> in return () method.



Now run the application. You can see this “Hello from function component” is there.

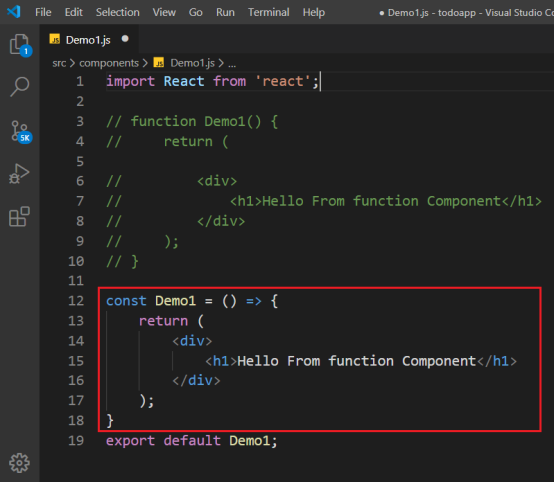
we are having 3 ways to create a functional component and we have just done it in one way.

# **Lab-5 -Approach2 (ECMAScript 6) syntax for creating a functional component.**

Now the **second way**is to use ES6 (ECMAScript 6) syntax for creating a functional component.

For this we have to use the arrow function as below:

const Demo1 = () => {  
 return (  
 <div>  
 <h1>Hello From function Component</h1>  
 </div>  
 );  
}

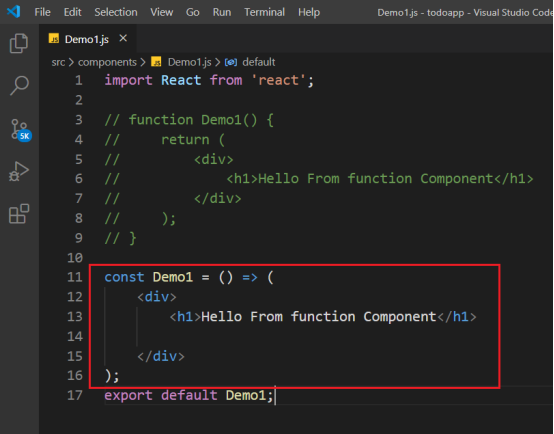


Now save this and check the browser and you can see it is working the same as before.

Now let’s see the **third way**of creating a functional component.

Here also we will use ES6 but we will modify our arrow function.

const Demo1 = () => (  
 <div>  
 <h1>Hello From function Component</h1>  
 </div>  
}



You can check your browser now and it is showing the same text as before.

# **ReactJS Components Communication**

1. Using Props with Class Components
2. Using Props with Functional Components

# Lab-6 :Using Props with Class Components

Create 2 (Class) Components: One as a Parent and another one as a Child.

**Parent.js**

import React, {Component } from 'react’;

export default class Parent extends Component {  
render() {  
 return (  
 <div>  
 <h1>I am parent Component</h1>  
 </div>  
 );  
 }  
}

**Child.js**

import React, { Component } from 'react';

export default class Child extends Component {  
render() {  
 return (  
 <div>  
 <h3>I am Child Component</h3>  
 </div>  
 );  
 }  
}

Now we will do the following tasks:

We will call the Child component inside Parent component

We will Pass some value from Parent component to Child component

# **Lab-7.Calling Child component inside Parent component:**

For this, we need to **import**this Child component on **Parent.js**

import Child from './Child';

And now just call this <child></child> inside render () method.

render() {  
 return (  
 <div>  
 <h1>I am parent Component</h1>  
 **<Child></Child>**  
 </div>  
 );  
}

Now run this app and we can see the contents of both Parent and Child component.

# **Lab-8.Passing some value from Parent to Child component:**

On **Parent.js**, we will declare a property as ‘Title’ for Child and assign some value to it.

import React, {Component } from 'react';  
**import Child from './Child';**

export default class Parent extends Component {  
render () {  
 return (  
 <div>  
 <h1>I am parent Component</h1>  
 **<Child Title="I am text from Parent Component"></Child>**  
 </div>  
 );  
 }  
}

And now on **Child.js**, we will simply fetch this ‘**Title**’ set by our Parent component. This **Title**attribute is accessed using **this.props.Title**.

To do this we need to use JSX syntax and as you know for this we need to use { } to print the value.

import React, {Component } from 'react';

export default class Child extends Component {  
render() {  
 return (  
 <div>  
 <h3>I am Child Component</h3>  
 **<h3>{this.props.Title}</h3>**  
 </div>  
 );  
 }  
}

Check the browser and you can see that “I am text from Parent Component”, which was sent by Parent to Child component.

# **Lab-9:Using Props with Functional Components**

On **Parent.js,** let us call our functional component **Demo1.js**.

Import and render the Demo1 component here as below:

import React, {Component } from 'react';  
import Child from './Child';\  
import Demo1 from './Demo1'

export default class Parent extends Component {  
 render() {  
 return (  
 <div>  
 <h1>I am parent Component</h1>  
 <Child Title="I am text from Parent Component"></Child>  
 <Demo1 Title="I am Text for function Component" ></Demo1>  
 </div>  
 );  
 }  
}

You can see that we have set **Title**property with this Demo1 component too. And like the previous example, we will also get this **Title**on Demo1 (functional) component.

Remember that in the functional component we do not have **this**keyword to fetch the value of prop. So here we have pass **props**as an argument to our functional component. And then we can use **props. Title**to get the value.

**Demo1.js**

import React from 'react';

const Demo1 = (props) => {  
 return (  
 <div>  
 <h1>Hello From function Component</h1>  
 <h2>{props.Title}</h2>  
 </div>  
 );  
}export default Demo1;

And run and check the output.

# **Lab-10 : State Example**

Let’s understand **State**with an example:

We will create a class component and name it as a **Sample**component.

In this, firstly we will simply create a variable and then display its value.

we will have a Button and by clicking on this Button we will change the value of this variable.

**Sample.js**

import React, {Component } from "react";

export default class Sample extends Component {  
state = {  
 a: 'Hello'  
};

render() {  
 return (  
 <div>  
 <h1>{this.state.a}</h1>  
 </div>  
 );  
 }  
}

Now run it and check on the browser, you will see “Hello” is there.

Now let us have a Button and we will have its onClick method.

by clicking this we will change the text from “Hello” to some other in our <h1> tag.

**Sample.js**

import React, {Component} from "react";

export default class Sample extends Component {  
state = {  
 a: 'Hello'  
};

handleButtonClick = () => {  
 console.log("Inside Button Click");  
 this.state.a = "You pressed Button";  
};render() {  
 return (  
 <div>  
 <h1>{this.state.a}</h1>  
 <button type="button" onClick={this.handleButtonClick} >Click Me</button>  
 </div>  
 );  
 }  
}

You can see that we have created ***handleButtonClick()***method of the button and inside that event, we are changing the variable text with the state.

Please notice that when you pressed that button the variable text is not updating but it is printing the console.!

So why does this happen?

For the solution read the warning that your console is showing.

It is saying that you can not mutate the state of a variable directly and you need to use **setState()** for this.

So to update the state, we’ll use **this.setState( )**.This is a built-in method to modify the state.

**Sample.js**

import React, {Component} from "react";

export default class Sample extends Component {  
state = {  
 a: 'Hello'  
};handleButtonClick = () => {  
 console.log("Inside Button Click");  
 // this.state.a = "You pressed Button";  
 this.setState({  
 a: "you clicked Button"  
 });  
};render() {  
 return (  
 <div>  
 <h1>{this.state.a}</h1>  
 <button type="button" onClick={this.handleButtonClick} >Click Me</button>  
 </div>  
 );  
 }  
}

Now run the app and click that button: