**React**

React is a component based JavaScript framework used to create expansive and maintainable single page applications (and Progressive Web Applications).

**Components**

Everything in react is a component. We create components and then render a combination of components together to form a web page. Components can also be configurable. Using components is beneficial because it promotes reusability of the code. It also incorporates separation of concerns and makes it easier to debug, maintain and update parts of application down the line.

Example of basic component:

import React from 'react';

const example = () => {

return <p>Hey, this example works :D</p>;

}

export default example;

**Two different ways of writing components**

There are two different ways to write a react component:

1. **Functional Component**: Use syntax of writing a function that returns the JSX code. These are also known as ‘presentational’, ‘dumb’ or ‘stateless’ components. ***We should use functional components wherever possible. It is the best practice***. Example 🡪 const cmp = () => { return <div>some JSX</div> }
2. **Class-based Component:** Use syntax of writing a class that extends Component class. It uses render method in React to render the JSX content. These are also known as ‘containers’, ‘smart’ or ‘stateful’ components. Example 🡪 class Cmp extends Component { render () { return <div>some JSX</div> } }

**Method - render()**

React contains a render method that it uses to render the content of a component on the web page.

***Render method is only used inside class based components.***

Everything written inside render method looks like a piece of html, but actually, it only looks similar to html tags. It has a few differences, for example, instead of ‘class=”main-app”’ we write ‘className=”main-app”’. This is because the code inside render method is actually JSX code which when transpiled transforms into pure JavaScript code. This syntactical sugar has been added to make it easier on the developers to code for html in a manner similar to html.

For example 🡪 Code Before transpiling:

render() {

return (

<div className="App">

<h1>Hi, This is a basic react app!</h1>

</div>

);

}

Code after transpiling:

render() {

return React.createElement(

'div', {className: 'App'},

React.createElement('h1', null, 'This is a random h1 tag!')

);

}

This is also one of the main reason we need to import React when creating a react component.

***One of the restrictions suffered by using the render method is that we can only have one parent element inside the render method***. If we need to render multiple elements, we need to wrap all these inside a containing parent element before we can render it using render method.

**Making Components Configurable**

To output dynamic content on components or to execute small, one-line JavaScript codes, we need to enclose the code in curly braces, i.e. ‘{}’. The JavaScript executed can only be short, one-line code. We can still execute a function in that one line that in turn calls other methods inside it.

Example (executing JS in JSX):

const dynamic = () => {

return <p>{Math.floor(Math.random() \* 30)} is a random number I generated. Isn't it cool?</p>;

}

There are two ways of making components Configurable:

**1) USING ‘props’ (Method Arguments) and ‘children’ (Reserved property of method argument)**

We can make components configurable using method arguments. These argument values are decided when using these component tags inside JSX. This gives the component more freedom for dynamic content and reusability.

We can also access the content inside opening and closing tags of a dynamic content using the property children of method arguments. ‘children’ is a reserved property in react and is used to refer to any JSX or text content written inside the component tags when using the component in JSX.

***‘props’ name can be changed since it is only the name of a function argument. The only restriction is that the method will only receive a single object as its argument while it is executed by react.***

***‘children’ property, though, is a reserved property and thus, we cannot change its name to another and then expect correct results.***

Example (configured component and use of children):

const person = (props) => {

return (

<div>

<p>Hi, I'm {props.name} and I am {props.age} years old.</p>

<p>{props.children}</p>

</div>

);

}

**2) USING ‘state’ (Reserved class property)**

We can use the ‘state’ reserved property to add dynamic content. Whenever any property inside state changes, the component is re-rendered.

This property can only be used while using Class based component, not with functional components.

Example:

class State extends Component {

state = {

persons: [

{name: 'John', age: 22},

{name: 'Jane', age: 23},

{name: 'Bob', age: 23}

]

}

render() {

return (

<div>

<p>Hi, I'm {this.state.persons[0].name} and I am {this.state.persons[0].age} years old.</p>

<p>Hi, I'm {this.state.persons[1].name} and I am {this.state.persons[1].age} years old.</p>

<p>Hi, I'm {this.state.persons[2].name} and I am {this.state.persons[2].age} years old.</p>

</div>

)

}

}

***Note: Props and State are the only two options of making components configurable with dynamic data based on input from parent component or internal logic respectively.***

Adding Output Event Handlers in JSX

Output event handlers in JSX can be added using a special syntax provided by React. This syntax is similar to html but at the same time has minor differences (like changing lowercase/Uppercase letters etc). For example: keyword for Click handler is: onClick.

The following url contains a list of supported Events in JSX: <https://reactjs.org/docs/events.html#supported-events>

Example:

switchPersonsOrder = () => {

// DO NOT DO THIS: this.state.persons[0].name = 'abc';

let length = this.state.persons.length;

let switchedPersons = [];

if(length > 1) {

for(let index=0; index < length-1; index++) {

switchedPersons.push(this.state.persons[index+1]);

}

switchedPersons.push(this.state.persons[0]);

}

this.setState({

persons: switchedPersons

})

}

render() {

return (

<div>

<button onClick={this.switchPersonsOrder} > Switch names </button>

<label> (Clicking this would change the order of persons appearing below) </label>

<p>Hi, I'm {this.state.persons[0].name} and I am {this.state.persons[0].age} years old.</p>

<p>Hi, I'm {this.state.persons[1].name} and I am {this.state.persons[1].age} years old.</p>

<p>Hi, I'm {this.state.persons[2].name} and I am {this.state.persons[2].age} years old.</p>

<p> {this.state.randomComment} </p>

</div>

)

}

Important Considerations while using Output Events

1. ***The state property must not be mutated directly***, as it would not do anything. ***We can only use setState method provided by Component class to change the state***. This method only need those properties inside state that are modified. The properties that are not added to this method remain unchanged.
2. ***Please note the Uppercase C in onClick (used in JSX) as opposed to onclick which is used in html***.
3. ***The method is passed to the event and it does not have parentheses – () – after the method name***. If the parentheses are added, the function would be executed at the time of component rendering. Using function without parenthesis means function definition is passed to the click event and executed by React.
4. ***Handler method definition in the component should use ES-6 method of writing functions (arrow functions).*** This is because arrow functions persist the value of ‘this’ keyword. ‘this’ keyword is usually used in these event handler methods if it need to access the setState method. If we use the normal function writing method instead, the value of this keyword will change and we would not be able to access methods of App class inside that method.

**Passing method references between components**

There may often be cases where it is required that the parent component decides what action to take based on an event that occurs in one of the child components. To do this, we need to pass the method definition as a property to the child component where the child component will then execute this once event is encountered.

For Example:

Parent Component

<h3> <b> 6. <u>Passing methods from parent component to child </u> </b> </h3>

{/\* Passing method without argument \*/}

<HandlerInParent buttonName="Change Age Randomly" onChangeClick={this.changeNumber} />

{/\* Passing method with argument using bind \*/}

<HandlerInParent buttonName="Change Age to 24" onChangeClick={this.changeNumber.bind(this, 24)} />

{/\* Passing method with argument using arrow function and method call \*/}

<HandlerInParent buttonName="Change Age to 27" onChangeClick={() => this.changeNumber(27)} />

{/\* In above component, method has parentheses because it is being called (when its parent arrow function is called inside child) and its result is returned by arrow function. \*/}

<Configurable name="Paul" age={this.state.randomNumber} />

Child Component (here HandlerInParent)

const handlerInParent = (props) => {

return <button onClick={props.onChangeClick} > {props.buttonName} </button>;

}

Arguments for methods may be passed in following two ways:

1. Pass copy of handler method created using bind where we give arguments – preferred.
2. Pass arrow function and call the handler method inside that handler method.

**Adding Styles to Components**

Styles can be added to components in the following two ways:

**1) Adding <ComponentName>.css file**

In this method, we create a file (usually with the same name as component). This style should contain styles for the component. We can use normal CSS syntax in this file.

The file created needs to be imported in the component JS file.

***The styles written in this file are all global regardless of where we import the component CSS file. They are not scoped to the component.***

Example:

JS file

import './StyledComponent.css';

const styledComponent = () => {

return <p className="StyledComponent"> This paragraph is styled using CSS file </p>;

}

CSS file

.StyledComponent {

background-color: darkcyan;

color: white;

padding: 0.5em;

}

**2) Use Inline styles using ‘style’ JSX property**

In this method, we create a JS Object that contains the style properties and its values. This object is then passed to the style property inside JSX code for component. This method creates an inline style for that component while converting JSX into HTML.

***The styles applied using this method are scoped to that component itself (or element in the component on which style property is applied).***

***Since this method uses React’s style property to convert object to inline style, it has the drawback that some properties (like hover) may not be applied correctly.***

Example:

const inlineStyledComponent = () => {

const style = {

backgroundColor: 'red',

color: 'white',

padding: '0.5em'

}

return <p style={style}> This paragraph is styled using CSS file </p>;

}