React Components

- React is component based.
- A component comprises template with presentation and logic.
- A pre-defined logic and presentation which you can implement and use in any application.
- There are several 3rd Party component libraries. [Telerik, Material UI etc.]
- React allows to create custom components.
- A component comprises of
    - Presentation - HTML
    - Logic - TS/JS/JSX
    - Style - CSS
- A component is re-usable.
- Components are classified into 2 types
    - Functional Component
    - Class Component
- Components can be added to your application by using 2 techniques
    - Embedded Technique
        - Single file comprises of logic, presentation and styles.
    - Module Technique
        - Component can be separated into Logic, Styles and HTML files.

External File as Component:

- Add a new folder “src” into project
- Add a new file “hello.js”

```
ReactDOM.render(
  <p> Welcome to React.js - External Component </p>,
```

```
    document.getElementById("container")
)
```

- Go to Index.html and link the script file.

```
<!DOCTYPE html>
<html>
  <head>
    <title>React App</title>
    <script
src="../node_modules/react/umd/react.development.js"></scri
pt>
    <script src="../node_modules/react-dom/umd/react-
dom.development.js"></script>
    <script
src="../node_modules/@babel/standalone/babel.js"></script>
    <script src="../src/hello.js" type="text/jsx"></script>
  </head>
  <body>
    <h2>React Application</h2>
    <div id="container">

    </div>
  </body>
</html>
```

JSX in React Component

- JSX by default supports only single line element configuration.
  Ex:

```
<h2> Heading </h2>          // Invalid JSX Code Block
<p> Paragraph </p>          // Multi Line not allowed
```

- JSX element configuration can be defined in “( )” if it comprises of multiple lines.
- Element must be a container to hold multiple lines.
    <p> etc.
  Ex:

```
const element = (
   <div>
      <h2>React.js</h2>
      <p>Components in React.js</p>
   </div>
);
ReactDOM.render(
   element,
   document.getElementById("container")
)
```

- Adding additional container may affect the presentation in HTML.
- Hence you have to use empty container.
  <> Start
  </> End

Ex:

Hello.js

```
const element = (
   <>
      <h2>React.js</h2>
      <p>Components in React.js</p>
   </>
```

```
);
ReactDOM.render(
   element,
   document.getElementById("container")
)
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>React App</title>
      <script
src="../node_modules/react/umd/react.development.js"></script>
      <script src="../node_modules/react-dom/umd/react-
dom.development.js"></script>
      <script
src="../node_modules/@babel/standalone/babel.js"></script>
      <script src="../src/hello.js" type="text/jsx"></script>
   </head>
   <body>
      <h2>React Application</h2>
      <table border="1" width="400">
         <tr>
            <td id="container"></td>
         </tr>
```

```
    </table>
  </body>
</html>
```

JSX Expressions

- JavaScript allows embedded expression in a string representation by using “${ }”
  `<div> ${ dynamicValue } </div>`
- JSX expression is embedded by using “{ }”

Ex:

Hello.js

```
var product = {
   Name: "Nike Causlas",
   Price: 5600.55,
   InStock: true
}
const element = (
   <>
    <dl>
       <dt>Name</dt>
       <dd>{product.Name}</dd>
       <dt>Price</dt>
       <dd>{product.Price}</dd>
       <dt>Stock</dt>
```

```
        <dd>{(product.InStock==true)?"Available":"Out of
Stock"}</dd>
    </dl>
   </>
);
ReactDOM.render(
   element,
   document.getElementById("container")
)
```

assets/styles.css

```
dt {
   font-weight: bold;
   background-color: gray;
   color: white;
   width: 300px;
}
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>React App</title>
      <link rel="stylesheet" href="../assets/styles.css">
      <script
src="../node_modules/react/umd/react.development.js"></script
>
```

```
    <script src="../node_modules/react-dom/umd/react-dom.development.js"></script>
    <script src="../node_modules/@babel/standalone/babel.js"></script>
    <script src="../src/hello.js" type="text/jsx"></script>
  </head>
  <body>
    <div id="container">

    </div>
  </body>
</html>
```

JSX will Not allow Void Element

- Void element usually will not have end tag.
  Ex: <img>
- Every element in JSX must have and End tag. Or define as self ending
  <img> </img>
  <img />
  <img> invalid

  Ex:
  Hello.js

```
var product = {
   Name: "Nike Causlas",
   Price: 5600.55,
   InStock: true,
   Photo: "../assets/shoe.jpg"
}
```

```
const element = (
  <>
   <dl>
      <dt>Name</dt>
      <dd>{product.Name}</dd>
      <dt>Price</dt>
      <dd>{product.Price}</dd>
      <dt>Stock</dt>
      <dd>{(product.InStock==true)?"Available":"Out of
Stock"}</dd>
      <dt>Preview</dt>
      <dd>
        <img src={product.Photo} width="100" height="100"/>
      </dd>
   </dl>
  </>
);
ReactDOM.render(
  element,
  document.getElementById("container")
)
```

Note:

- JSX supports binding of dynamic values only to properties of HTML elements.
- You can't bind dynamic values to attributes of Elements.
  <table width="" height="" class="effects"> Width and Height are attributes
  var table = document.createElement("table");
  table.width = "400";
  table.height = "100"; // invalid - table element doesn't have height property

table.className = “effects”;

Ex:

Hello.js

```
var styles = "effects";
const element = (
   <>
     <h1 className={styles}>Welcome to React.js</h1>
   </>
);

ReactDOM.render(element,
document.getElementById("container"));
```

Styles.css

```
.effects {
   background-color: red;
   color:white;
   text-align: center;
   padding: 10px;
}
```

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>React App</title>
      <link rel="stylesheet" href="../assets/styles.css">
      <script
src="../node_modules/react/umd/react.development.js"></scri
pt>
      <script src="../node_modules/react-dom/umd/react-
dom.development.js"></script>
```

```
        <script
    src="../node_modules/@babel/standalone/babel.js"></script>
        <script src="../src/hello.js" type="text/jsx"></script>
      </head>
      <body>
        <div id="container">

        </div>
      </body>
    </html>
```

Note: <h1 class={styles}> is invalid as "class" is attribute not a property.

Render Complex Data

- Complex data is like a Array, JSON, Map etc.

Ex: Rendering Array Elements in Ordered List – Traditional Map method

Hello.js

```
var categories = ["Electronics", "Footwear", "Fashion"];


const element = (
  <>
    <h2>Categories List</h2>
    <ol>
      {
        categories.map(function(category){
```

```
            return <li>{category}</li>
          })
        }
    </ol>
  </>
);
ReactDOM.render(element,
document.getElementById("container"));
```

Ex: Using LAMBDA [Arrow Function]

Hello.js

```
var categories = ["Electronics", "Footwear", "Fashion"];
const element = (
  <>
    <h2>Categories List</h2>
    <ol>
      {
        categories.map(category=>
          <li>{category}</li>
        )
      }
    </ol>
  </>
```

```
);
ReactDOM.render(element,
document.getElementById("container"));
```

Ex: Rendering a Table Dynamically

Hello.js

```
var data = [
   {Name: "JBL Speaker", Price: 4500.55, Photo:
"../assets/speaker.jpg"},
   {Name: "Nike Casuals", Price: 6000.55, Photo:
"../assets/shoe.jpg"},
   {Name: "Shirt", Price: 2000.33, Photo: "../assets/shirt.jpg"}
];
const element = (
   <>
    <h2>Products Table</h2>
    <table width="400" border="1" align="center">
     <thead>
        <tr>
           <th>Name</th>
           <th>Price</th>
           <th>Preview</th>
        </tr>
     </thead>
     <tbody>
```

```
        {
           data.map(product=>
             <tr>
                <td>{product.Name}</td>
                <td>{product.Price}</td>
                <td><img src={product.Photo} width="60" height="60"
/></td>
             </tr>
           )
        }
     </tbody>
   </table>
  </>
);


ReactDOM.render(element,
document.getElementById("container"));
```

Ex: Render with Bootstrap components

- Install bootstrap for your project
  > npm install bootstrap
- Link bootstrap.css in “Index.html”
  <link rel="stylesheet"
  href="../node_modules/bootstrap/dist/css/bootstrap.css">
- Hello.js

```
var data = [
   {Name: "JBL Speaker", Price: 4500.55, Photo:
"../assets/speaker.jpg"},
   {Name: "Nike Casuals", Price: 6000.55, Photo:
"../assets/shoe.jpg"},
   {Name: "Shirt", Price: 2000.33, Photo: "../assets/shirt.jpg"}
];
const element = (
   <>
    <h2 className="text-center bg-primary text-white mt-
3">Products Table</h2>
    <table className="table table-hover">
     <thead>
        <tr>
           <th>Name</th>
           <th>Price</th>
           <th>Preview</th>
        </tr>
     </thead>
     <tbody>
        {
           data.map(product=>
             <tr>
                <td>{product.Name}</td>
                <td>{product.Price}</td>
                <td><img src={product.Photo} width="60"
height="60" /></td>
             </tr>
           )
        }
     </tbody>
    </table>
   </>
```

```
);

ReactDOM.render(element,
document.getElementById("container"));
```

Ex: **Rendering a Card**

Hello.js

```
var data = [
  {Name: "JBL Speaker", Price: 4500.55, Photo:
"../assets/speaker.jpg"},
  {Name: "Nike Casuals", Price: 6000.55, Photo:
"../assets/shoe.jpg"},
  {Name: "Shirt", Price: 2000.33, Photo: "../assets/shirt.jpg"}
];
const element = (
  <>
   <h2 className="text-center bg-primary text-white mt-
3">Products Catalog</h2>
   <div className="row row-cols-3">
      {
         data.map(product=>
            <div className="card">
               <div className="card-header">
                  <h3>{product.Name}</h3>
               </div>
               <div className="card-body">
```

```
                <img src={product.Photo} width="200" height="200" />
              </div>
              <div className="card-footer">
                <h4>{product.Price}</h4>
              </div>
            </div>
          )
      }
    </div>
  </>
);

ReactDOM.render(element,
document.getElementById("container"));
```

JSX allows to create element and configure the properties dynamically:

- Elements are created dynamically and added to virtual DOM by using "**React.createElement()**"
  Syntax:
  React.createElement(
  "elementName",
  {
  Property: value,
  Property: value
  },

“Default Content”
)

FAQ: What is difference between?

a) document.createElement() : Adding a new Element into actual DOM
b) React.createElement() : Adding a new Element into virtual DOM

Ex:

Hello.js

```
const element = React.createElement(
   "div",
   {
      className: "container",
      align: "center"
   },
   "Welcome to React.js"
);
ReactDOM.render(
   element,
   document.getElementById("container")
);
```

Hello.js

```
const element = React.createElement(
```

```
  "img",
  {
    width: '200px',
    height: '200px',
    src: '../assets/shoe.jpg'
  }
);
ReactDOM.render(
  element,
  document.getElementById("container")
);
```

Creating Nested Elements and Rendering into Virtual DOM:

Ex:

Hello.js

```
const element = React.createElement(
  "div",
  {
    className: "container",
    align: "center"
  },
  "Welcome to React.js",
  React.createElement(
    "h2",
```

```
      {
        align: "center"
      },
      "JSX Introduction"
    ),
    React.createElement(
      "p",
      null,
      "This is a Paragraph"
    ),
    React.createElement(
      "img",
      {
        src: "../assets/shoe.jpg",
        width: "200",
        height: "200"
      }
    )
);
ReactDOM.render(
   element,
   document.getElementById("container")
);
```

You can configure virtual DOM element in JSON style

- JSON comprises of Key and Value
- The keys required for creating virtual DOM element
    - type: It defines element type
    - props: The properties for element

Syntax:

```
const element = {
   type: "div",
   props: {
      className: "container",
      align: "center",
      children: ["Welcome to React"]
   }
}
```

React Components

- Component comprises of logic, presentation and styles.
- **Logic** is defined with JSX
- **Presentation** is defined with HTML
- **Styles** are defined with CSS
- **React Components can be designed**
    - By using JavaScript Function
    - By using JavaScript Class
- Component designed by using JavaScript function is called as **Functional Component**
- Component designed by using JavaScript class is called as **Class Component**

Functional Components

- A functional component is JavaScript anonymous function.
- It uses a reference name.

Ex:

Hello.js

```
const HeaderComponent = function(){
   return (
      <>
       <h1 className="bg-primary text-white text-center mt-
3">Amazon Shopping</h1>
      </>
   )
}
const NavigationComponent = () => (
   <>
     <ul>
        <li>Home</li>
        <li>About</li>
        <li>Contact</li>
     </ul>
   </>
)
const FooterComponent = () => (
   <>
```

```
      <div className="bg-dark text-white text-center">&copy;
copyright 2021</div>
    </>
)

const MainComponent = () => (
    <>
      <HeaderComponent/>
      <div className="row" style={{height: "400px"}}>
        <div className="col-2">
          <NavigationComponent />
        </div>
        <div className="col-10">
          <p>Online - Shopping</p>
        </div>
      </div>
      <FooterComponent />
    </>
)

ReactDOM.render(
    <MainComponent />
    ,
    document.getElementById("container")
```

)

Rules for configuring a custom component:

- The custom components must not collide with HTML DOM elements.
  <header>: not good – It is HTML DOM element
- Event component name must be in **Pascal Case** or in **lowercase**.
- For actual DOM we use lowercase.
- For Virtual DOM we use Pascal Case [Every word first letter must be a capital letter].
  Ex: **FooterComponent, LoginComponent**

Class Component in React

- Class Component is just a JavaScript class that extends "React.Component" base class.
  Syntax:
  class LoginComponent extends React.Component
  {
  }
- JavaScript class comprises only
  - Properties
  - Methods
  - Accessors
  - Constructor
- Data is stored in properties
- Functionality is defined with methods
- Component class can render virtual DOM elements by using "render()" method.
- "render()" method renders virtual DOM.

Syntax:

```
class Login extends React.Component
{
    render(){
      return “JSX”
    }
}
```

FAQ:

- Can we define a variable in class?
  No
- Can we define a function in class?
  No

Ex:

Hello.js

```
class HeaderComponent extends React.Component {
  render(){
    return (
      <>
        <h1 className="bg-primary text-center text-white p-2 mt-2">Amazon Shopping</h1>
      </>
    )
  }
}
```

```
class NavComponent extends React.Component {
   render(){
      return(
         <>
          <ul>
             <li>Home</li>
             <li>About</li>
             <li>Contact</li>
          </ul>
         </>
      )
   }
}
class FooterComponent extends React.Component
{
   render(){
      return(
         <>
          <div className="bg-primary text-white text-center p-
2">&copy; copyright 2021</div>
         </>
      )
   }
}
```

```
class MainComponent extends React.Component
{
   render(){
      return(
         <>
          <div className="container-fluid">
           <HeaderComponent />
           <div className="row mt-3" style={{height: '400px'}}>
             <div className="col-3">
               <NavComponent />
             </div>
             <div className="col-9">
               <p>Shopping - Home</p>
               <p>Visit for latest offers.</p>
             </div>
           </div>
           <FooterComponent />
          </div>
         </>
      )
   }
}
ReactDOM.render(
   <MainComponent />,
```

```
  document.getElementById("container")
)
```

Functional Component vs Class Component

Function Component:

- Functional components are referred as “Stateless” components.
- Functional because they are designed by using JavaScript function.
- Stateless because they don’t hold or manage state.
- **It is Difficult to transport data from one component to another.**
- **It uses lot of “round trips”.**
- **It will be slow in access.**
- **Tightly coupled**
- **Not easy to extend**
- It is more secured.
- It is discreet in access.
- It manages memory.
- It reduces the burden on server.
- Always choose functional components when you don’t need regular extensions.

Class Component

- It is referred as “Stateful”
- It is a ES6 Class
- It comprises of data and logic
- It can hold and manage memory.
- It is good for transporting data across components [Requests]
- It can have a local state [Context]
- Loosely coupled
- Extensible [Easy to extend]

- Maintainability and Testability
- Uses more memory
- Not secured – Not discreet