**React**

Facebook, a modern web application has over 1.6 billion daily users and handles around 150000 messages and 500000 likes every minute. Updating each like/photo in the UI using JavaScript will have significant overhead and thus, data may load very slowly and the page will become unresponsive. Developing and maintaining applications of this size is also difficult.

React JS library helps us to build the user interface of such large applications. ReactJS, a JavaScript library for creating user interfaces,makes the development of UI components easy and modular.

React JS was created by Jordan Walke, a software engineer at Facebook and open sourced to the world by Facebook and Instagram.

Angular is one of the popular frameworks used for UI development. Let us compare React and Angular in brief:

|  |  |
| --- | --- |
| ***React*** | ***Angular*** |
| React is a small view library | Angular is a full framework |
| React covers only the rendering and event handling part | Angular provides the complete solution for front-end development |
| Presentation code in JavaScript powered by JSX | Presentation code in HTML embedded with JavaScript expressions |
| React's core size is smaller than Angular, so bit fast | Angular being a framework contains a lot of code, resulting in longer load time |
| React is very flexible | Angular has less flexibility |
| Great performer, since it uses Virtual DOM | Angular uses actual DOM which affects its performance |

**Rendering Elements to DOM**

Observe the file src/index.js

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**);​​​​​​​**

ReactDOM.render is used to render an element or component to the virtual DOM.

* The first argument specifies what needs to be rendered
* The second argument specifies where to render.

The root element is present inside index.html

**<body>**

**<noscript>You need to enable JavaScript to run this app.</noscript>**

**<div id="root"></div>**

**</body>**

You can render elements also using the ReactDOM.render method as shown below

**ReactDOM.render(<h1>Hello React!<h1/>, document.getElementById('root'));**

**StrictMode**

React.StrictMode: It is a helper component which helps the developers in identifying the problems in the application.The React.StrictMode, component does not render any visible UI

StrictMode helps with the below :

* Identifying components with unsafe lifecycles
* Warning about legacy string ref API usage
* Warning about deprecated findDOMNode usage
* Detecting unexpected side effects
* Detecting legacy context API

**Rendering Elements to DOM**

Observe the file src/index.js

**ReactDOM.render(<App />, document.getElementById('root'));**

ReactDOM.render is used to render an element or component to the virtual DOM.

- The first argument specifies what needs to be rendered

- The second argument specifies where to render.

The root element is present inside index.html

**<body>**

**<noscript>You need to enable JavaScript to run this app.</noscript>**

**<div id="root"></div>**

**</body>**

You can render elements also using the render method as shown below.

**ReactDOM.render(<h1>Hello React!<h1/>, document.getElementById('root'));**

* **StrictMode**

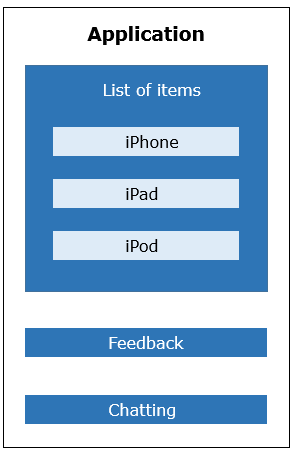
*React.StrictMode* : It is a helper component which helps the developers in identifying the problems in the application.The React,StrictMode component does not render any visible UI

StrictMode helps with the below :

* Identifying components with unsafe lifecycles
* Warning about legacy string ref API usage
* Warning about deprecated findDOMNode usage
* Detecting unexpected side effects
* Detecting legacy context API

**Why Components?**

Let's have look at the below application where you have different sections as a list of items, feedback, and chatting.



You have the following challenges in developing this application:

1. Entire page will get re-rendered even when a section of the page (eg. feedback section) undergoes changes

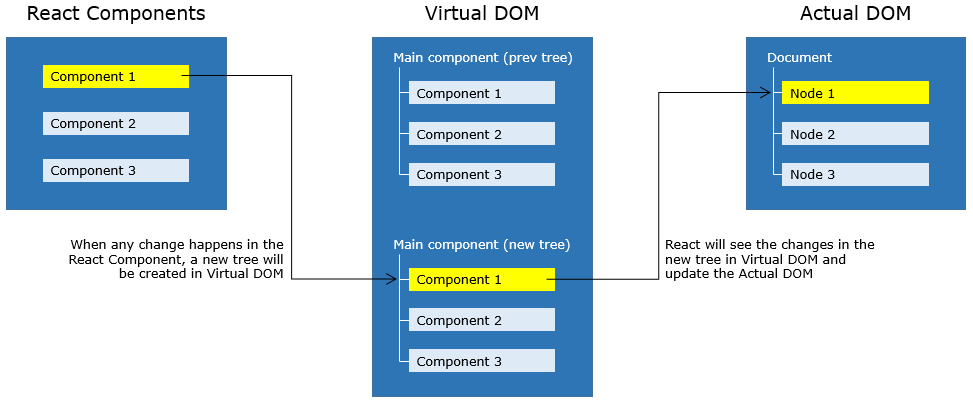
2. You will have to re-write code for each item even though they have similar behavior

3. You will have to take additional care to make sure that the functionality of one part of the application do not interfere with another part

If you implement the above wireframe by dividing them into components - encapsulating its own data and presentation logic then you can achieve modularity and re-usability.

**React Component : Virtual DOM**

Below diagram illustrates the working of virtual DOM:



* Whenever any updates happens in the application, the virtual DOM gets modified. React computes the difference between the previous virtual tree and the current virtual tree
* Based on these differences React will figure out how to make updates to the actual DOM efficiently
* React does all the computations in its abstracted DOM and updates the DOM tree accordingly
* Virtual DOM enhances performance and efficiency by minimizing expensive updates in the actual DOM
* Hence React is said to be a great performer because it manages a Virtual DOM

**How to create React components**

There are 2 types of components that can be created in React :  
*1. Class based components  
2. Functional components*

* **Functional Components**  
  •    Created as a simple JavaScript function  
  •    It just returns the HTML elements  
  The functional component can be created as follows:

**import React from 'react';**

**function App() {**

**return React.createElement("h1", {}, "Hello World!!!");**

**}**

**export default App;**

* App - The component name should be in Pascal Casing
* React.createElement - helps to create an element in plain JavaScript

In the code, React.createElement("h1", {}, "Hello World!!!");

* h1 - is the HTML element to be used
* { } - Attributes of an element can be mentioned
* "Hello World!!!" - Content to be appended to the h1 element
* export default App - App component must be exported so that it could be used in any other files

To display elements of a component, the component must be rendered.  
For rendering a component, ReactDOM.render method is used as follows :  
Syntax:

**ReactDOM.render(<parameter 1/>, parameter 2);**

ReactDOM.render method will take 2 parameters:

* 'parameter 1' is the name of the component to be rendered
* 'parameter 2' is the HTML node reference where the component to be rendered

index.js:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**ReactDOM.render(<App/>, document.getElementById('root'));**

**reportWebVitals();**

**Demo: React Components**

**Highlights:**

* Creation of Functional component
* Rendering a component

**Demo steps:**

1. Create App component in App.js and create an element to render "Hello World" in the heading format as shown below:

**import React from 'react'**

**function App () {**

**return React.createElement("h1", {}, "Hello World");**

**}**

**export default App;**

1. In index.js, render App component using the ReactDOM.render method as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

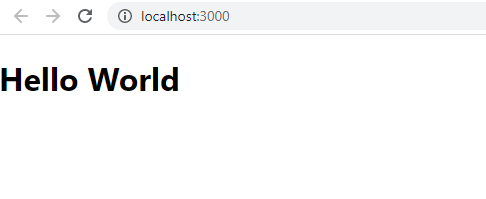
**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

1. Observe the below output:



**Demo: React Components**

**Highlights:**

* Creation of Class component
* Rendering a component

**Demo steps:**

1. Create App component in App.js and create an element to render "Hello World" in the heading format as shown below:

**class App extends React.Component {**

**render() {**

**return React.createElement("h1", {}, "Hello World");**

**}**

**}**

**export default App;**

2. In main.js, render App Component using the ReactDOM.render method as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

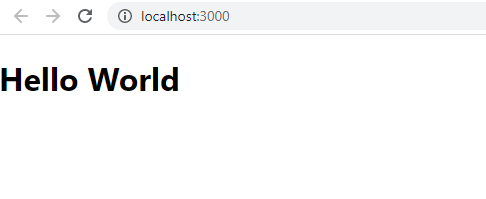
**import App from './App';**

**import \* as serviceWorker from './serviceWorker';**

**ReactDOM.render(<App />, document.getElementById('root'));**

**serviceWorker.unregister();**

3 . Observe the below output:



**Why JSX?**

Let us understand the need of JSX by taking the below example:

Code using pure JavaScript:

**function App() {**

**return (**

**React.createElement('form', {},**

**React.createElement("h1", {}, "Login"),**

**React.createElement('input', {type: 'text',placeholder:'Name', value: '',}),**

**React.createElement('br', {}),React.createElement('br', {}),**

**React.createElement('input', {type: 'password', placeholder: 'password',**

**value: '',}),**

**React.createElement('br', {}), React.createElement('br', {}),**

**React.createElement('button', {type: 'submit'}, "Login"))**

**)**

**}**

**export default App;**

You can observe from the above code, that you need to write more lines of JavaScript code to implement the Login component. The Code looks difficult to understand and hence productivity goes down. JSX has been introduced in React to create elements that are very easy to read and write, which makes the component's code simple and understandable. The above Login component can be written using JSX in an easier way:

**function App() {**

**return (<form><h2>Login</h2>**

**<input type="text" placeholder="Name" /><br/><br/>**

**<input type="password" placeholder="password" /> <br/><br/>**

**<input type="submit" nvalue="log" />**

**</form>);**

**}**

**export default App;**

​​​​​​​This technique is a replacement for writing pure JavaScript code and hence enhances productivity.

**Nesting of JSX elements**

When working with React components, you may need to render multiple React elements. For example, consider the below code where there is a need to render multiple JSX elements.

**function App() {**

**return**

**<h3>ReactJS:</h3>**

**<img src="./image/react.PNG" width="120" height="120"/>**

**<p> React is a JavaScript library for creating User Interfaces.</p>**

**}**

**export default App;**

The above code logs an error saying "Adjacent JSX elements must be wrapped in an enclosing tag".  As per the JSX syntax, all the adjacent elements must be wrapped in an enclosing tag i.e. there should be only one outermost element. Hence the above code can be modified as follows:

**function App() {**

**return (**

**<div>**

**<h3>ReactJS:</h3>**

**<img src="./image/react.PNG" width="120" height="120" />**

**<p> React is a JavaScript library for creating User Interfaces.</p>**

**</div>**

**);**

**}**

**export default App;**

**React Fragments**

By adhering to JSX syntax the <div> tag can be used for grouping the elements and this introduces an extra and unnecessary node into the DOM. As a solution to this, React Fragments are introduced which is a common pattern in React for a component to return multiple elements. React Fragments allows to group a list of React elements without adding any extra node to the DOM.

The previous code can be modified as follows using React Fragment:

**function App() {**

**return (**

**<React.Fragment>**

**<h3>ReactJS:</h3>**

**<img src="./image/react.PNG" width="120" height="120" />**

**<p> React is a JavaScript library for creating User Interfaces.</p>**

**</React.Fragment>**

**);**

**}**

**export default App;**

**Empty Tags**

You can use empty tags instead of React.Fragment

**function App() {**

**return (**

**<>**

**<h3>ReactJS:</h3>**

**<img src="./image/react.PNG" width="120" height="120" />**

**<p> React is a JavaScript library for creating User Interfaces.</p>**

**</>**

**);**

**}**

**export default App;**

Fragments can be passed with key attribute if required. Empty tags don’t support any attributes.

**Demo: JSX in components**

**Highlights:**

* Creating elements in JSX
* How JSX code will be converted to JavaScript

**Demo steps:**

* 1. Modify the file App.js by writing the following code using JSX:

**function App () {**

**return <h1>Hello World</h1>**

**}**

**export default App;**

* 1. Render App component in index.js file as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

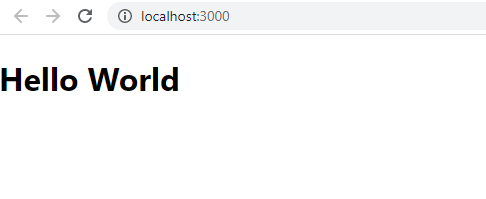
**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

* 1. Observe the below output:



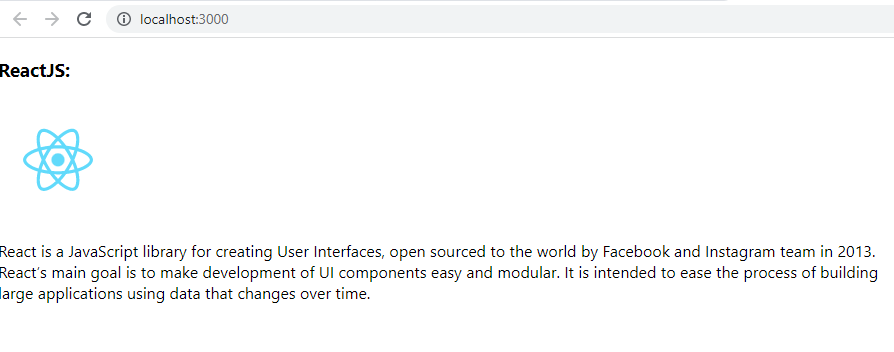
**Demo: Nested JSX elements**

**Highlights:**

* How to render multiple elements
* Nesting JSX elements

**Demo steps:**

Create a component to render an image with its description as follows:



1. Modify App.js as shown below:

**import logo from './logo.svg';**

**function App() {**

**return (**

**<>**

**<h3>ReactJS:</h3>**

**<img src={logo} width="120" height="120" />**

**<p>**

**{" "}**

**React is a JavaScript library for creating User Interfaces, open sourced**

**to the world by Facebook and Instagram team in 2013.**

**<br />**

**React’s main goal is to make development of UI components easy and**

**modular. It is intended to ease the process of building**

**<br /> large applications using data that changes over time.**

**</p>**

**</>**

**);**

**}**

**export default App;**

In order to use an image, the image has to be imported and the imported value should be provided to the src tag as an expression.

The logo.svg used in the component would be present in the src folder by default.

1. Render App Component in the index.js file as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

**JavaScript expressions in JSX**

You are now aware of how to use JSX to create React elements, you may have a requirement to include JavaScript expressions in React elements. So, let's see how to write JavaScript expressions in JSX.

JavaScript expressions to be evaluated should be wrapped within curly braces as follows:

**<h1> { Expression to be evaluated } </h1>**

Content which has to be displayed as is will be written in double quotes, wrapped within curly braces as follows:

**<h1> {" Content to be displayed "} </h1>**

**Accessing a variable:**

You can access the value of any variable within curly braces as shown below:

**function App() {**

**let count = 5;**

**return (**

**<>**

**<h1>{count}</h1>**

**<h2>{count \* count}</h2>**

**</>**

**);**

**}**

**export default App;**

Arithmetic operators can also be used as shown above

**Accessing an object:**

The properties of an object can be accessed as shown below:

**function App() {**

**let name = {**

**firstName: "John",**

**lastName: "Doe",**

**};**

**return (**

**<>**

**<h1>**

**{name.firstName} {name.lastName}**

**</h1>**

**</>**

**);**

**}**

**export default App;**

**Specifying attributes' expression:**

You can provide dynamic values to attributes as shown below:

**function App() {**

**let highlight = {**

**color: "blue",**

**backgroundColor: "grey",**

**};**

**return (**

**<>**

**<h1 style={highlight}>Welcome to React</h1>**

**</>**

**);**

**}**

**export default App;**

**Highlights:**

* Passing values to expression
* Evaluating the expression

**Demo steps:**

Create a component to display an algebraic expression as follows:



* + 1. Modify App component as shown below:

**function App() {**

**var x=25, y=30**

**return (**

**<>**

**<h2> Evaluating expression </h2>**

**<h3> {x} {">"}{y} {":"} {x>y ? 'True' : 'False'} </h3>**

**</>**

**);**

**}**

**export default App;**

* + 1. Render App component in the index.js file as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

**Conditional Rendering**

**Render the content based on conditions**

In any web application, you may require to show and hide any content based on some condition. For example, if admin is logged in to an application it should display a message saying "Welcome" else it should display saying "Please Login". Similarly, displaying the list of products based on its availability.

To achieve these requirements, the if-else or ternary operator can be used for conditional rendering.

* 1. **Using if-else:**

**function App() {**

**let element = null;**

**let isLoggedIn = false;**

**if (isLoggedIn) {**

**element = <h2>Welcome Admin</h2>;**

**} else {**

**element = <h2>Please Login</h2>;**

**}**

**return <>{element}</>;**

**}**

**export default App;**

* 1. **Using ternary operator:**

**function App() {**

**let isLoggedIn = false;**

**return isLoggedIn ? <h2>Welcome Admin</h2> : <h2>Please Login</h2>; }**

* 1. **Using variables:**

**export default App;**

**function App() {**

**let element = null;**

**let isLoggedIn = false;**

**isLoggedIn**

**? (element = <h2>Welcome Admin</h2>)**

**: (element = <h2>Please Login</h2>);​​​​​​​**

**return <>{element}</>;**

**}**

**export default App;**

Looping using map() method

You can render lists in React, using map() method.

**function App() {**

**var employees = [**

**{ empId: 1234, name: "John", designation: "SE" },**

**{ empId: 4567, name: "Tom", designation: "SSE" },**

**{ empId: 8910, name: "Kevin", designation: "TA" },**

**];**

**return (**

**<>**

**<table>**

**<thead>**

**<tr>**

**<th>EmpID</th>**

**<th>Name</th>**

**<th>Designation</th>**

**</tr>**

**</thead>**

**<tbody>**

**{employees.map((employee) => {**

**return (**

**<tr>**

**<td>{employee.empId}</td>**

**<td>{employee.name}</td>**

**<td>{employee.designation}</td>**

**</tr>**

**);**

**})}**

**</tbody>**

**</table>**

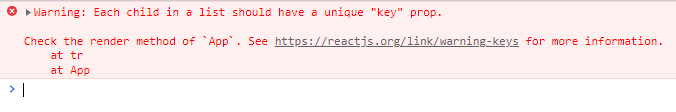
**</>**

**);**

**}**

**export default App;**

The above code provides a warning in the console as shown below



To overcome the above error, you need to provide a unique key to each element in an array as shown below

**<tbody>**

**{employees.map(employee => {**

**return (<tr key={employee.empId}>**

**<td>{employee.empId}</td>**

**<td>{employee.name}</td>**

**<td>{employee.designation}</td>**

**</tr>)**

**})**

**}**

**</tbody>**

**Keys**

Keys are used by React to identify the items that are added, removed, or modified. Keys should be provided to the elements inside the array to provide a unique identity.

Keys should be unique and most commonly the ID’s of the data are used as keys

**Demo: Conditional Rendering in JSX**

**Highlights:**

* Usage of ternary operator for conditional rendering
* Usage of map function for looping

**Demo Steps:**

1. Create App component within App.js file as shown below

**function App() {**

**var employees = [**

**{ empId: 1234, name: "John", designation: "SE" },**

**{ empId: 4567, name: "Tom", designation: "SSE" },**

**{ empId: 8910, name: "Kevin", designation: "TA" },**

**];**

**return (**

**<>**

**<table>**

**<thead>**

**<tr>**

**<th>EmpID</th>**

**<th>Name</th>**

**<th>Designation</th>**

**</tr>**

**</thead>**

**<tbody>**

**{employees.map((employee) => {**

**return (**

**<tr key={employee.empId}>**

**<td>{employee.empId}</td>**

**<td>{employee.name}</td>**

**<td>{employee.designation}</td>**

**</tr>**

**);**

**})}**

**</tbody>**

**</table>**

**</>**

**);**

**}**

**export default App;**

1. Render App component to the DOM in the index.js file

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

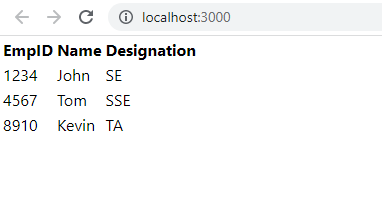
**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

The output of the above code is



**Highlights:**

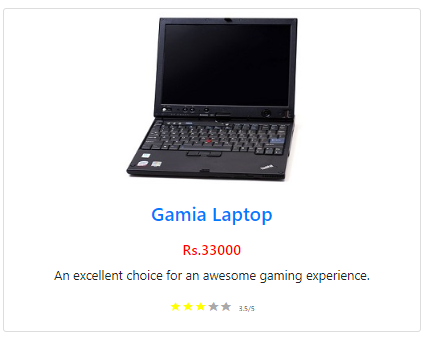
* Creating Product component
* Implementing star rating functionality

**Demosteps:**

Out of the listed components let us create 2 components Product and Rater component. Screenshot and the code is as follows:

**Product component:**

Create Product component as shown below



Download the Laptop.jpg image and place it within public/images folder

**import React from "react";**

**import Rater from "./Rater.js";**

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

**function Product() {**

**return (**

**<div className="thumbnail" style={{ textAlign: "center" }}>**

**<div className="row>">**

**<div className="col-sm-6">**

**<img src={"images/Laptop.jpg"} alt="laptop" className="img img-rounded img-responsive"/>**

**</div>**

**<div className="col-sm-6">**

**<div className={"caption"}>**

**<a href={"/productDetails"}>**

**<h3>Gamia Laptop</h3>**

**</a>**

**<h4>**

**<span style={{ color: "red" }}>Rs. 33000</span>**

**</h4>**

**<p>An excellent choice for an awesome gaming experience.</p>**

**<Rater />**

**<span style={{ fontSize: "9px" }}>3/5 </span>**

**</div>**

**</div>**

**</div>**

**<br />**

**</div>**

**);**

**}**

**export default Product;**

**Rater component:**

https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_10991335737918820000_shared/web-hosted/assets/REACT_rating_22Nov16_1419.PNG

Create Rater.css file within css folder the below styling

**.rating {**

**color: #a9a9a9;**

**margin: 0;**

**padding: 0;**

**}**

**ul.rating {**

**display: inline-block;**

**}**

**.rating li {**

**list-style-type: none;**

**display: inline-block;**

**padding: 1px;**

**text-align: center;**

**font-weight: bold;**

**cursor: pointer;**

**}**

**.rating .filled {**

**color: yellow;**

**}**

Create Rater component as shown below

**import './Rater.css';**

**function Rater() {**

**return (**

**<ul className="rating">**

**<li className="filled">{'\u2605'}</li>**

**<li className="filled">{'\u2605'}</li>**

**<li className="filled">{'\u2605'}</li>**

**<li>{'\u2605'}</li>**

**<li>{'\u2605'}</li>**

**</ul>**

**)**

**}**

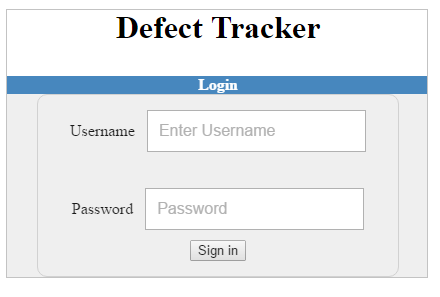
**export default Rater;**

Note: The star in the Rater component will be in yellow color if the class "filled" in added.

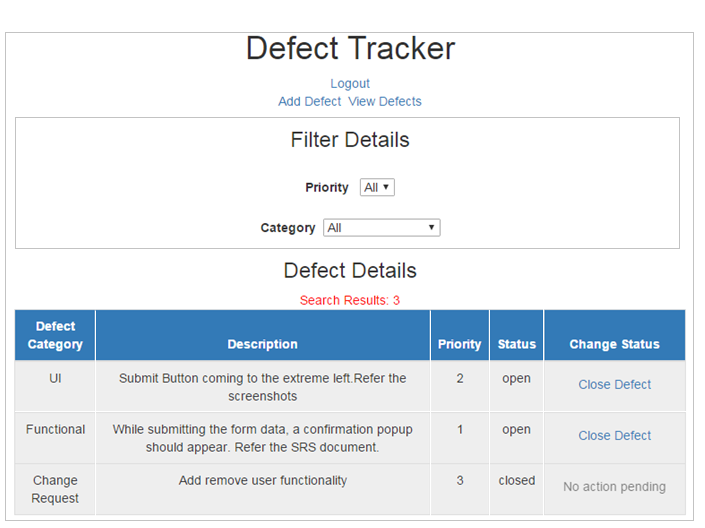
**Exercise 1:** Defect Tracker Application

You will be systematically developing an application to add, view, and filter the defects in an application which is identified by the tester. In the application there will be two roles - tester and developer.  
The screenshots of the entire application with its functionality is shown below:

**Login view:** Users should be able to enter the credentials and submit the form

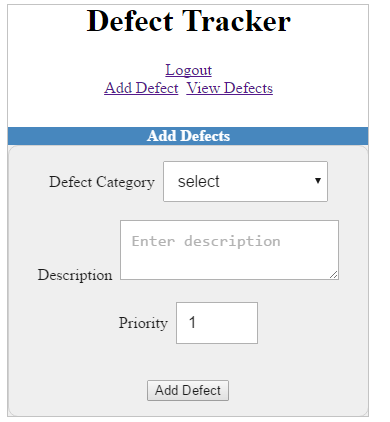


**View Defects:** In View Defects functionality, the user will be able to view the defects with the corresponding Description, Category, Status, Priority, and Status changing button.

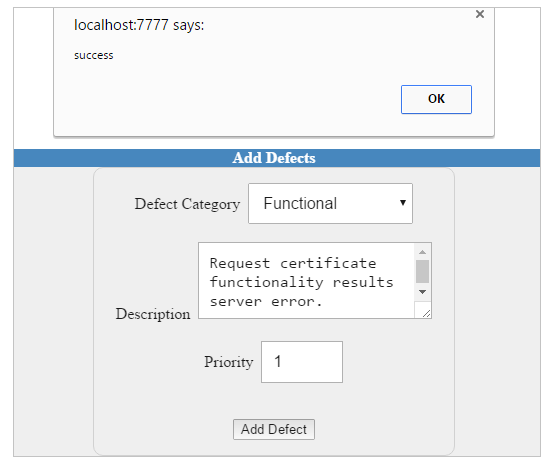


And develop other functionalities too by dividing it into different components.

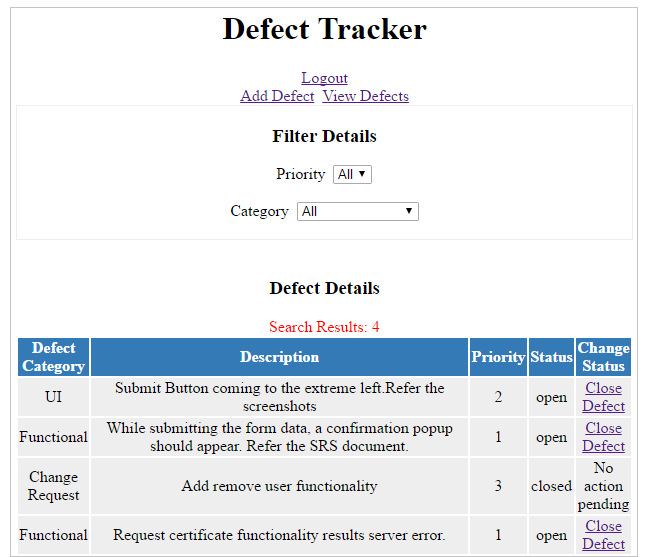
Add defects (available only for Tester role): Testers will be logging different defects after testing any application developed by the team. The tester can enter the below details and add the defects to the existing list.



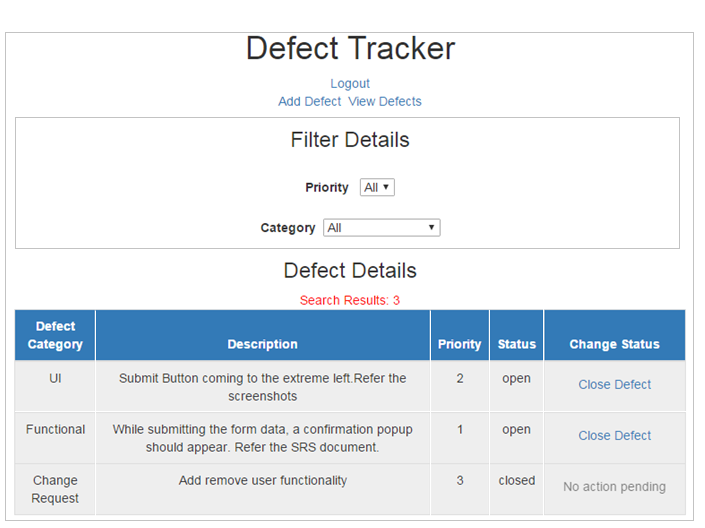
A success message should be displayed on submitting the form with the details as shown below:



And the updated details will be reflected in View defects as follows:



Now create a component as per the below screenshot for the View Defects page and hardcode the value wherever required:



**Typography in Material-UI**

To use various typography variants, import Typography component and use it within React component. You can display various typography variants like h1, h2, subtitle, body, caption using variant attribute as shown below.

**​​​<Typography variant="h4" gutterBottom>Welcome to React</Typography>**

gutterBottom adds a margin-bottom of 0.35em

**Highlights:**

* Styling the components using bootstrap library

**Demo Steps:**

1. Install bootstrap using the below command

​​​​​​​ **npm i bootstrap**

1. Modify the App component as shown below

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

**function App() {**

**return (**

**<>**

**<h1>Welcome to React</h1>**

**<button className="btn btn-success">Submit</button>**

**</>**

**);**

**}**

**export default App;**

1. Render the App component within the index.js file

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

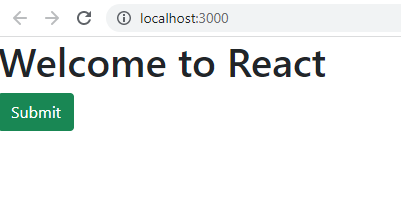
**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

1. The App component is styled as shown below



**Highlights:**

•    Use CSS inline styling   
•    Use CSS classes

**Demo Steps:**

1. Modify the App.css file as shown below

**.button {**

**background-color:blue;**

**color:white;**

**border-radius:10px;**

**width:100px;**

**height:30px;**

**}**

2. Modify the App component as shown below

**import './App.css'**

**function App() {**

**return (**

**<>**

**<h1 style={{ color: "green", fontFamily: "verdana" }}>**

**Welcome to React**

**</h1>**

**<button className="button">Submit</button>**

**</>**

**);**

**}**

**export default App;**

1. ​​Render the App component within the index.js file

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

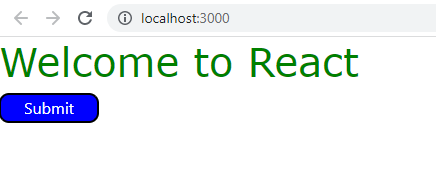
**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**reportWebVitals();**

The App component is styled as shown below



**Highlights:**

* Style the components using the Material-UI library

**Demo Steps**

1. Install Material UI using the below command

**npm install @mui/material @emotion/react @emotion/styled**

2. Modify the App component as shown below

**import Typography from "@mui/material/Typography";**

**import Button from "@mui/material/Button";**

**function App() {**

**return (**

**<>**

**<Typography variant="h4" gutterBottom>**

**Welcome to React**

**</Typography>**

**<Button variant="contained" color="primary">**

**Submit**

**</Button>**

**</>**

**);**

**}**

**export default App;**

3. Render the App component within the index.js file

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**);**

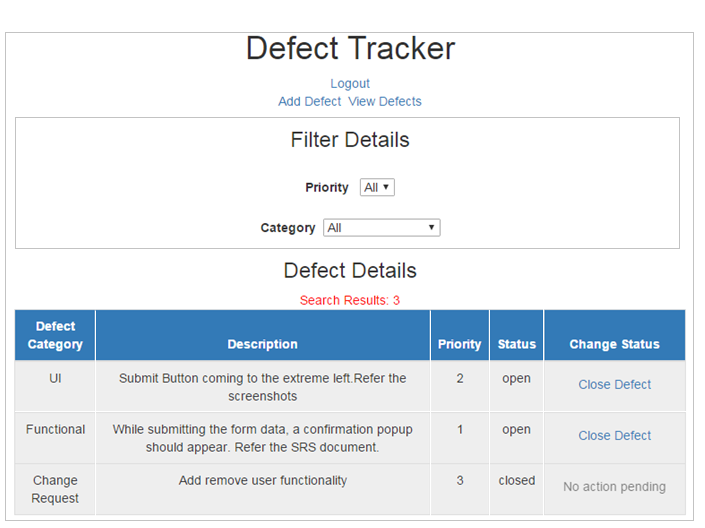
**reportWebVitals();**

The App component is styled as shown below



**Problem Statement:**

Style the Defect Tracker Application as per the below screenshot provided below using bootstrap



**Why Props and State?**

Look at the below Product component. You can notice that the product details are hardcoded.

**Product component:**

**const Product = () =>{**

**return(<div>**

**<div className={"thumbnail"}>**

**<img src = "Images/Laptop.jpg" class="img img-rounded img-responsive"/>**

**<div class = "caption" style="text-align:center">**

**{/\* Hardcoded value for a product  \*/}**

**<a href = "/productDetails"> <h3> Gamia Laptop </h3></a>**

**<h4><span style={{color:"red"}}>Rs. 33000</span></h4>**

**<p>An excellent choice for an awesome gaming experience.</p>**

**<Rater />**

**<span style ={{fontSize:"9px"}}>3/5</span>**

**</div>**

**</div>**

**</div>**

**);**

**}**

In a real-time application, components must deal with dynamic data.

This data could be something internal to the component or may be the data that is passed from another component. To bind the data to the component, you need two JS objects i.e. state and props.

**State**

**What is State?**

The state is an initial value set for a component, which is used for interactivity.

Let's see how to set the state of a component.

Using constructor, the state of a component is created and initialized using this.state as follows:

**Syntax:**

**constructor() {**

**super();**

**this.state = { counter: 1 };**

**}**

In the above code, the state 'counter' is set to 1 and 'counter' would be accessed as **this.state.counter.**

As the user-defined component extends React.Component, it becomes a child component of it. Hence, whenever a child component has a constructor, it has to call the parent class's constructor using the super() method. And super() method should be the first statement within a constructor.

Now let's create a Timer component where on clicking a button, the timer starts.

Below is the implementation to start the timer:

When a button is clicked, by invoking the handleClick() method - set the interval and pass it to start() method.

**handleClick(e) {**

**this.interval = setInterval(this.start, 1000);**

**}**

In start() method, for every second, state will be updated using **setState()** method. Whenever setState() is called, it calls render() method to render the updated value on UI.

**start() {**

**this.setState({ secondsElapsed: this.state.secondsElapsed + 1 });**

**}**

When an event occurs in the component, we will update the state of the component using the **setState()** method inside the event handler method as shown below:

**class Timer extends React.Component {**

**constructor() {**

**super ()**

**this.state = {**

**counter: 0**

**}**

**}**

**handleClick(e){**

**this.setState({counter:this.state.counter+1})**

**}**

**render() {**

**return(<React.Fragment>**

**<h2> Seconds Elapsed: {this.state.counter} </h2>**

**<button onClick = {this.handleClick}> Increment Counter </button>**

**</React.Fragment>)**

**}**

**}**

When we run the above code and click on the increment counter button, it throws an error as shown below:



The above error is because this keyword is not pointing to the component

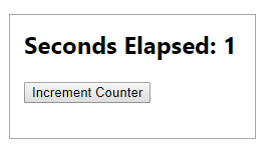
This error can be resolved using the arrow functions as shown below.

**handleClick = (e) => {**

**this.setState({counter:this.state.counter+1})**

**}**

Now we would get the output as shown below after clicking on the increment counter button



**setState() is asynchronous:**

Consider the below code snippet to understand how setState() is asynchronous:

**class App extends React.Component {**

**constructor() {**

**super()**

**this.state = {**

**quantity: 0,**

**price:0**

**}**

**}**

**update = () => {**

**this.setState({quantity:5})**

**if(this.state.quantity == 5) {**

**this.setState({price:2000-100})**

**}**

**}**

**render() {**

**return(<React.Fragment>**

**<h1>{this.state.quantity}</h1>**

**<h1>{this.state.price}</h1>**

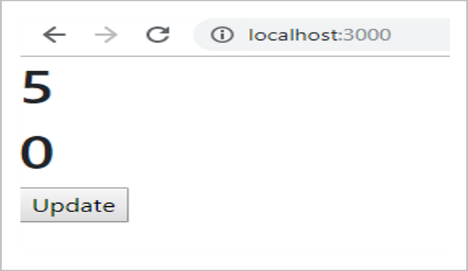
**<button onClick={this.update}>Update</button>**

**</React.Fragment>)}**

**}**

In the above code snippet, the price is updated based on quantity.

When we run the above code and click on the update button, below is the output we get:



The price is not updated in the above screenshot. This is because setState() is asynchronous, the price gets updated before quantity gets updated.

To overcome the above problem, we can pass a callback function to the setState() method as shown below:

**update = () => {**

**this.setState({quantity:5},()=>{**

**if(this.state.quantity == 5) {**

**this.setState({price:2000-100})**

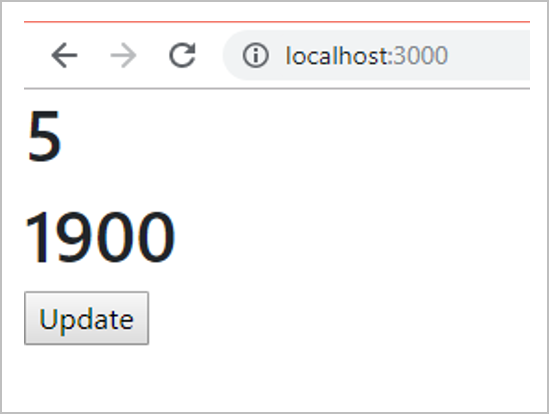
**}**

**})**

**}**

Now, the price gets updated after the quantity is updated.

When we run the above code and click on the update button, the output is:



**What is State? - Functional Components**

State is a JavaScript object used to manage the data of an application

To associate state with components you can use useState hook

**useState():**

useState hook is used to add or initialize state value within a functional component. This hook helps to preserve the state value of a component.

**Syntax**

**const [count, setCount] = useState(0);**

useState() is a function which takes ‘initialState’ as the initial value for any state variable and returns 2 values.

In the above code snippet, count is a state variable which is initialized with value 0 and setCount is the function used to update the state count

Whenever state is updated, the component is re-rendered, and updated value is displayed on the UI. React remembers the current value of state between re-renders and provides the most recent value to the function.

Observe the usage of  useState() from the below example:

**import {useState} from 'react';**

**function App() {**

**const [name,setName] = useState("Jack")**

**return (**

**<div>**

**Your Name is: {name}<br/>**

**<button onClick={()=>setName("Tom")}>Change</button>**

**</div>**

**);**

**}**

**export default App;**

* In the above example useState() hook helps to set the initial value as "Jack".
* ‘name’ is the state of the component ‘App’. useState() returns a state ‘name’ with its initial value ‘Jack’ and ‘setName’ method.
* When user clicks on the button, click event is fired and the value of ‘name’ state is updated to 'Tom' through setName method.

The state can be updated using an event handler method as shown.

**import {useState} from 'react';**

**function App() {**

**const [name,setName] = useState("Jack")**

**const handleChange = () => {**

**setName("Jill")**

**}**

**return (**

**<div>**

**Your Name is: {name}<br/>**

**<button onClick={handleChange}>Change</button>**

**</div>**

**);**

**}**

**export default App;**

You can use useState() hook any number of times in any component to set multiple state variables as shown below:

​​​​​​​ **​​​​​​​const [name,setName] = useState("Jack")**

**const [age,setAge] = useState(0)**

**Highlights:**

Handling data in a component

Making component interactive

**Usage of state**

**Demo steps:**

Create a component to start the timer:

1. Create a component Timer inside the Timer.js file as shown below:

**import React from 'react';**

**class Timer extends React.Component {**

**constructor() {**

**super();**

**this.state = {**

**secondsElapsed: 0**

**};**

**}**

**start = () => {**

**this.setState({**

**secondsElapsed: this.state.secondsElapsed + 1**

**});**

**}**

**handleClick = (e) => {**

**this.interval = setInterval(this.start, 1000);**

**}**

**render() {**

**return ( <React.Fragment><br/><br/>**

**<button onClick = {this.handleClick}>Start timer</button><br/><br/>**

**<h2> Seconds Elapsed:**

**{this.state.secondsElapsed}**

**</h2>**

**</React.Fragment>);**

**}**

**}**

**export default Timer;**

2. Render Timer component to the DOM within the index.js file as shown below

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import Timer from './Timer';**

**import \* as serviceWorker from './serviceWorker';**

**ReactDOM.render(<Timer />, document.getElementById('root'));**

**serviceWorker.unregister();**

3. Observe the below output:

