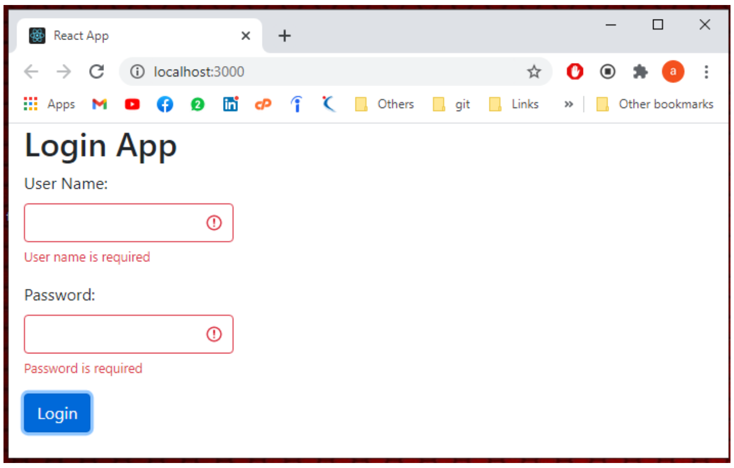
Integratation of Bootstrap:



Login App

User Name:

User name is required

Password:

Password is required

Login

Add the fallowing cdn links in index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <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" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
    <script
      src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
      integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
      crossorigin="anonymous"
    ></script>
    <script
      src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js"
      integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx"
      crossorigin="anonymous"
    ></script>
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
      integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
      crossorigin="anonymous"
    />
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import { Formik, Field, Form, ErrorMessage } from "formik";
import * as yup from "yup";

const LoginComponent = () => {
  return (
    <Formik
      initialValues={{
        uname: "",
        password: "",
      }}
      validationSchema={yup.object({
        uname: yup
          .string()
          .required("User Name is required")
          .matches("^[a-zA-Z]*$", "Invalid User Name"),

        password: yup
          .string()
          .max(10, "Password should not exceed 10 Characters")
          .required("Password is required"),
      })}
      onSubmit={(values) => {
        if (values.uname === "adi" && values.password === "adi123") {
          alert("Welcome..." + values.uname);
        } else {
          alert("Login Denied");
        }
      }}
    >
      {({ errors, touched }) => (
        <div className="col-4">
          <h2>Login App</h2>
          <hr />
          <Form>
            <div className="form-group">
              <label>User Name</label>
              <Field
                name="uname"
                type="text"
                className={
                  errors.uname && touched.uname
                    ? "form-control is-invalid"
                    : "form-control"
                }
              />
              <span className="invalid-feedback">
                <ErrorMessage name="uname"></ErrorMessage>
              </span>
            </div>

            <div className="form-group">
              <label>Password </label>
              <Field
                name="password"
                type="password"
                className={
                  errors.password && touched.password
                    ? "form-control is-invalid"
                    : "form-control"
                }
              />
              <span className="invalid-feedback">
                <ErrorMessage name="password"></ErrorMessage>
              </span>
            </div>

            <button type="submit" className="btn btn-primary">
              Login
            </button>
          </Form>
        </div>
      )}
    </Formik>
  );
};

const element = <LoginComponent></LoginComponent>;
ReactDOM.render(element, document.getElementById("root"));
```

Introduction to Hooks

- We know that a Component in React can be created as a **Class Component** or a **function Component**.
- When we want features like **Managing State** in React Components or responding to **Lifecycle methods** then we will opt for using Class Components.
- Problems of existing Class Components:
  1. Wrapper Hell
  2. Huge Components
  3. Confusing Classes
  4. classes don't minify very well
- React want to present an API out of the box that makes React easier to build great UIs with the best Performance Hooks are introduced.

What is Hook?

- Hooks are special functions that let you "hook into" React state and lifecycle features from function components. Hooks don't work inside classes — they let you use React without classes.
- **Hooks** are a new addition in React 16.8.
- They let you use **state** and other React features without writing a class.
- To make function components more powerful, React has introduced several built in hooks.
- Hooks in React are classified into Basic Hooks, Additional Hooks.

Basic Hooks	**Additional Hooks**
useState	useReducer
useEffect	useCallback
useContext	useMemo
	useRef
	useImperativeHandle
	useLayoutEffect
	useDebugValue

Note: We can also create Custom Hooks.

Working with useState() Hook:

Eg:

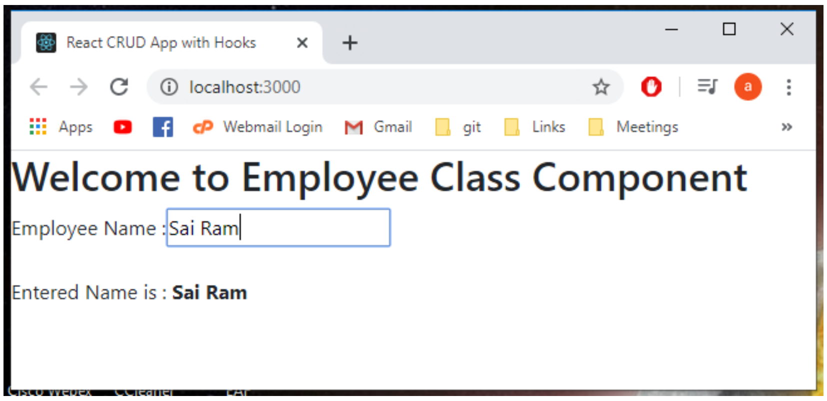
```
import React from 'react'
import ReactDOM from 'react-dom'

class Employee extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            name: 'Sai'
        }
    }

    changeName = (e) => {
        this.setState({ name: e.target.value });
    }

    render() {
        return (
            <div>
                <h2>Welcome to Employee Class Component</h2>
                <p>
                    <label>Employee Name :
                      <input type="text" value={this.state.name} onChange={this.changeName
}></input>
                    </label>
                </p>
                <p>
                    Entered Name is : <b>{this.state.name}</b>
                </p>
            </div>
        )
    }
}

const element = <Employee></Employee>
ReactDOM.render(element, document.getElementById("root"));
```



Welcome to Employee Class Component

Employee Name : Sai Ram

Entered Name is : **Sai Ram**

Limitations of above Program:

1. For a simple use case, we have written huge lines of code that includes writing **constructor**, calling baseclass constructor.
2. All that involves additional overhead to the application performance.
3. As the application grows, even our code becomes more complex and it becomes unmanageable.

But as a developer, we don't want our code to become unmanageable. Now we will develop a new employee component but this time we will create it as a **function component**.

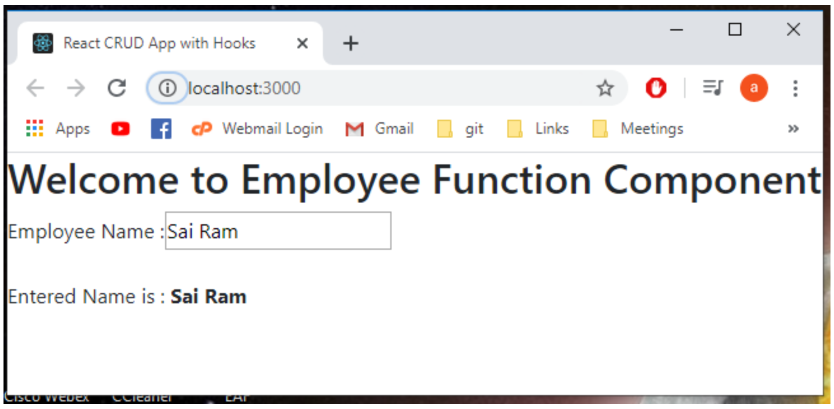
```
import React, {Component,useState } from 'react'
import ReactDOM from 'react-dom'


function Employee() {
    const [name, setName] = useState("Sai");

    function changeName(e) {
        setName(e.target.value);
    }

    return (
        <div>
            <h2>Welcome to Employee Function Component</h2>
            <p>
                <label>Employee Name :
                  <input type="text" value={name} onChange={changeName}></input>
                </label>
            </p>
            <p>
                Entered Name is : <b>{name}</b>
            </p>
        </div>
    )
}

const element = <Employee></Employee>
ReactDOM.render(element, document.getElementById("root"));
```



Welcome to Employee Function Component

Employee Name : Sai Ram

Entered Name is : **Sai Ram**

What useState() method do when it is called?

- It declares a “state variable” that in the above example the variable is **name**.
- This is a way to “preserve” some values between the function calls.
- useState is a new way to use the exact same capabilities that **this.state** provides in a class.
- Normally, variables “disappear” when the function exits but state variables are preserved by React.

What do we pass to useState as an argument?

- The only argument to the useState() Hook is the initial state.
- Unlike with classes, the state doesn’t have to be an object.
- We can keep a number or a string if that’s all we need.
- In our example, we just want ‘Sai’ as initial state for our variable.
- (If we wanted to store two different values in state, we would call useState() twice.)

What does useState return?

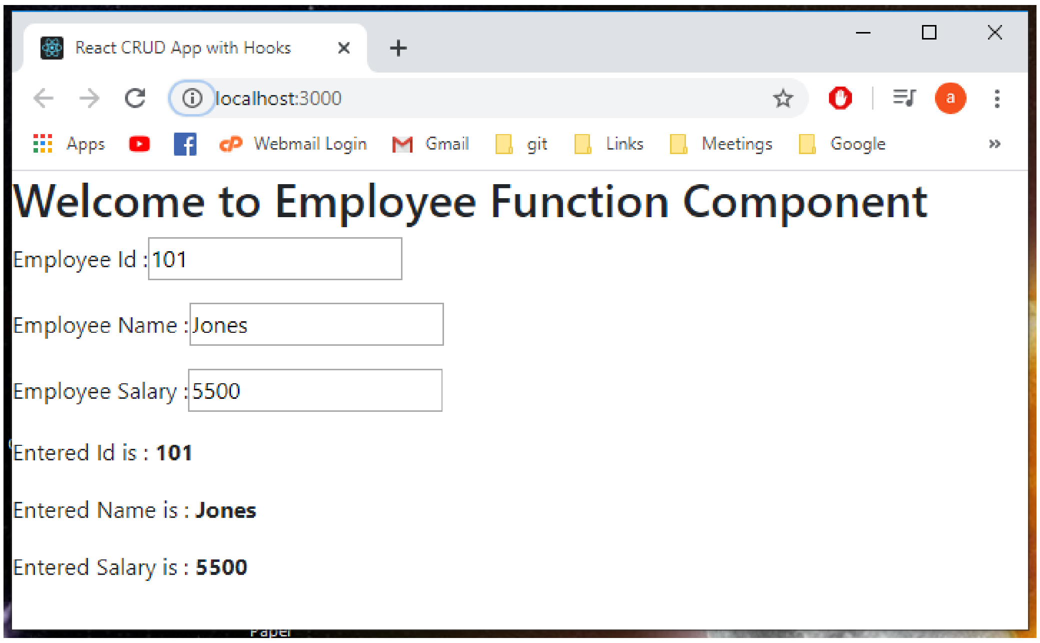
- It returns a pair of values:
  the current state
  a function that updates it.

- That’s why we write

```
const [name, setName] = useState("Sai");
```

- This is similar to **this.state.name** and **this.setState** in a class, except you get them in a pair.

Example: With more properties.



Welcome to Employee Function Component

Employee Id : 101

Employee Name : Jones

Employee Salary : 5500

Entered Id is : **101**

Entered Name is : **Jones**

Entered Salary is : **5500**

```
import React, { Component, useState } from 'react'
import ReactDOM from 'react-dom'

function Employee() {
    const [id, setId] = useState();
    const [name, setName] = useState();
    const [salary, setSalary] = useState();

    function changeId(e) {
        setId(e.target.value);
    }
    function changeName(e) {
        setName(e.target.value);
    }

    function changeSalary(e){
        setSalary(e.target.value);
    }

    return (
        <div>
            <h2>Welcome to Employee Function Component</h2>
            <p>
                Employee Id :
                <input type="text" value={id} onChange={changeId}></input>
            </p>
            <p>
                Employee Name :
                <input type="text" value={name} onChange={changeName}></input>
            </p>
            <p>
                Employee Salary :
                <input type="text" value={salary} onChange={changeSalary}></input>
            </p>
            <p>
                Entered Id is : <b>{id}</b>
            </p>
            <p>
                Entered Name is : <b>{name}</b>
            </p>
            <p>
                Entered Salary is : <b>{salary}</b>
            </p>
        </div>
    )
}
const element = <Employee></Employee>
ReactDOM.render(element, document.getElementById("root"));
```

Note:
Creating the multiple state variables for id, name and salary is not recommend approach. Because that lead to program lengthy and reduce readability of the code.
State variables can hold objects and arrays just fine, so you can still group related data together.
Lets create an object that hold **id, name and salary**, and pass it to the **useState()** function.

```
import React, { Component, useState } from 'react'
import ReactDOM from 'react-dom'

function Employee() {
    const initialFormState = { id: '', name: '', salary: '' };
    const [employee, setEmployeeData] = useState(initialFormState);

    function changeEmployeeInfo(e) {
        setEmployeeData({ ...employee, [e.target.name]: e.target.value });
    }

    return (
        <div>
            <h2>Welcome to Employee Function Component</h2>
            <p>
                Employee Id :
                <input type="text" name="id" value={employee.id}
                 onChange={changeEmployeeInfo}></input>
            </p>
            <p>
                Employee Name :
                <input type="text" name="name" value={employee.name}
                onChange={changeEmployeeInfo}></input>
            </p>
            <p>
                Employee Salary :
                <input type="text" name="salary" value={employee.salary}
                onChange={changeEmployeeInfo}></input>
            </p>
            <p>
                Entered Id is : <b>{employee.id}</b>
            </p>
            <p>
                Entered Name is : <b>{employee.name}</b>
            </p>
            <p>
                Entered Salary is : <b>{employee.salary}</b>
            </p>
        </div>
    )
}
const element = <Employee></Employee>
ReactDOM.render(element, document.getElementById("root"));
```