What is ReactJS?

ReactJS is a JavaScript library used to build User Interfaces(UI). It significantly decreases the code with it's components, states i.e. hooks, etc.

Creating react app

Open your terminal in the directory you would like to create your application. Run this command to create a React application named my-react-app:

```
npx create-react-app my-react-app
```

OR, you can directly make your application without specifying a name, like this:

```
npx create-react-app .
```

In this case, all files will be kept in the current directory.
Note: When choosing folder name, make sure there are no spaces or capital letters because of npm naming restrictions.
Once the base application is created, if the folder specified you just have to enter the folder. You can use this command to enter:

```
cd directory-name
```

Then just start up the application with this command:

```
npm start
```

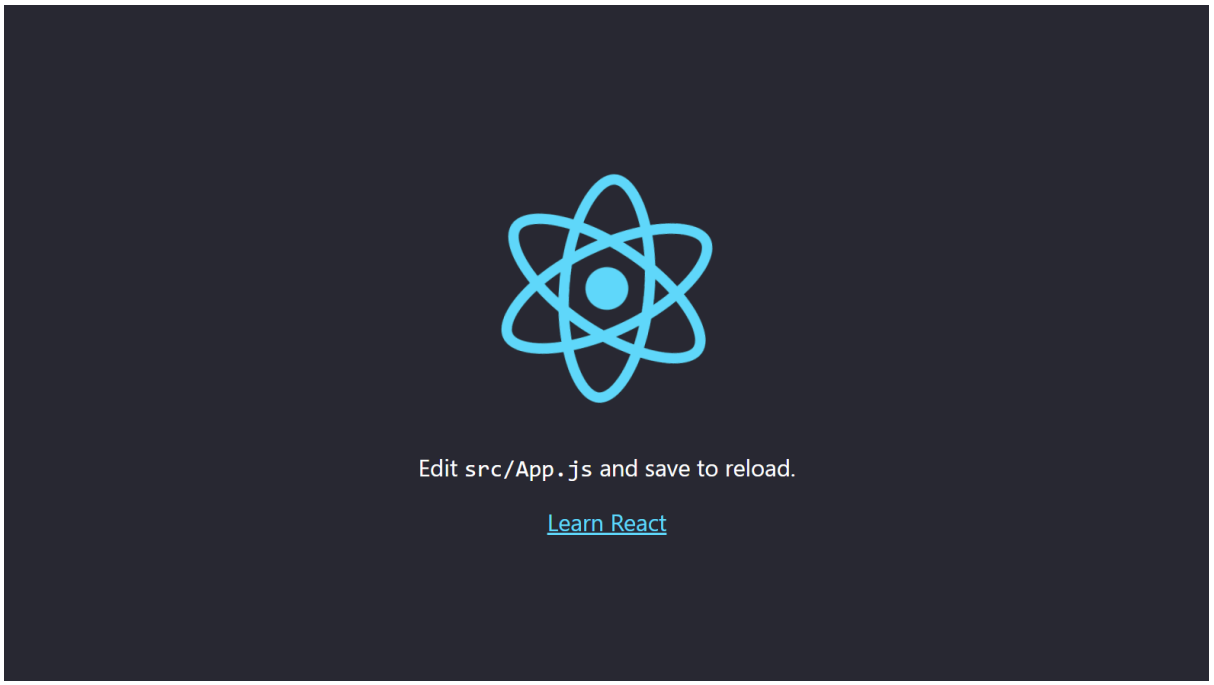
and you are good to go!

Run and Check

Run the React Application with this command:

```
npm start
```

A new browser window will pop up, if it does't then go on http://localhost:3000/.
Check if it is showing the same page:



If it's the same page then you are good to go!

Hello World

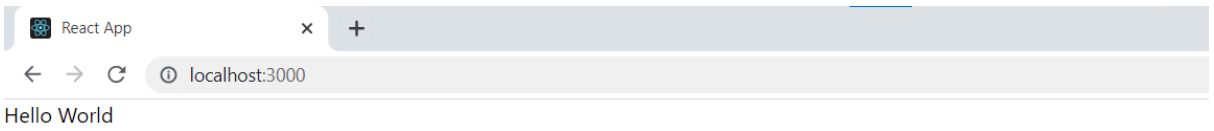
For this first you need to navigate to `src/App.js`, it will look like:

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

  export default App;
```

Whatever you put in return will be rendered as HTML on the page, you can change it like:

```
function App() {
  return (
    <div className="App">
      Hello World
    </div>
  );
  }
```

Note: Remember to wrap whole return value in an HTML element as you can't return multiple elements but you can return multiple elements in one element.
Page would look like this:



What is ES6?

ES6 stands for ECMAScript 6. ECMAScript is a JavaScript standard intended to ensure a common language across different browsers. ES6 is the 6th version of ECMAScript.

Why ES6? / Features of ES6 / Upgrades in ES6

React uses ES6 and all of these new features will make your coding experience in react much much better. You will be able to do things with much more ease and in very less lines! Features like:

- Arrow Functions:

```
const hello = () => {
  return "Hello World!";
}
```

or

```
const hello = () => "Hello World!";
```

- .map(): `.map` can be used for alot of things, one of it's use case is, we can make any number of cards through loop and just put it in jsx, like this:

```
const data = ['title1', 'title2', 'title3'];
let cards = data.map((item) => <card>{item}</card>)
```

- **Destructuring:**

Old Way:

```
const languages = ['JS', 'Python', 'Java'];
const js = languages[0]
```

```
const python = languages[1]
```

```
const java = languages[2]
```

New Way:

```
const languages = ['JS', 'Python', 'Java'];
```

```
const [ js, python, java ] = languages
```

- Ternary Operator: With this, you can write if/else conditions in one line. It's syntax is fairly simple like this:

```
condition ? <expression if true> : <expression if false>
```

Example:

```
let loading = false;
```

```
const data = loading ? <div>Loading...</div> :
<div>Data</div>
```

- Spread Operator:

```
const languages = ['JS', 'Python', 'Java'];
const morelanguages = ['C', 'C++', 'C#']
```

```
const allLanguages = [...languages, ...morelanguages]
```

Output:

```
["JS","Python","Java","C","C++","C#"]
```

and many more like, classes, modules.

React JSX

What is JSX?

JSX stands for JavaScript XML. It is similar in appearance to HTML, hence provides a way to easily write HTML in react.

Coding in JSX

Earlier we had to make an HTML element or append it into existing ones with methods like `createElement()` / `appendChild()`

```
const elem = React.createElement('h1', {}, 'Hello World!');
```

Now we can just do it directly, like this:

```
const elem = <h1>Hello World!</h1>
```

Expressions in JSX

You can write the expression in `{}`
You can write simple mathematical operations to variable to states to complicated operations with ternary operators and it will return the result, like:
Mathematical Operations:

```
const elem = <h1>React was released in {2010+3}</h1>
```

Variables/States:

```
const name = "CWH"
```

```
const elem = <h1>My name is {name}</h1>
```

Ternary Operators:

```
const elem = <h1>Hello {name ? name : 'World'}</h1>
```

React Components

There are two types of components:

1. Class Based Components
2. Function Based Components

Class Based Components

Before making class based component we need to inherit functions from `React.Component` and this can be done with `extends`, like this:

```
class Cat extends React.Component {
  render() {
    return <h1>Meow</h1>;
  }
}
```

it also requires a `render` method which returns HTML.

Function Based Components

In function it's simpler, we just need to return the HTML, like this:

```
function Cat() {
  return <h1>Meow</h1>;
}
```

Note: Component's name must start with uppercase letter.

Rendering a Component

We made a component, now we want to render/use it. Syntax for using a component is:

```
<ComponentName />
```

Components in Files

To have less mess inside main file(with all the components in the same file) and to resuse components on different pages, we have to make them separately. So that we can just import them in any file and use them!
For that we will just make a new file called `Cat.js`, make class or function based component there and `export default` that class/function! Like this:

```
function Cat() {
  return <h1>Meow</h1>;
}
```

```
export default Cat;
```

Note: File name must start with uppercase letter.

Props

As mentioned earlier, we can import the same component in different files and use it, but maybe in different files some changes in the component is needed. For that, we can use `props`! Like this:

Component:

```
function Cat(props) {
  return <h1>Meow's color is {props.color}</h1>;
}
```

Main file:

```
<Cat color="purple" />
```

React Props

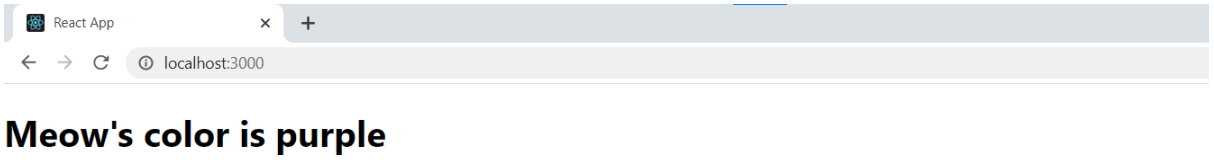
Props are arguments passed to React components via HTML attributes. Example:
Component:

```
function Cat(props) {
  return <h1>Meow's color is {props.color}</h1>;
 }
```

Main file:

```
<Cat color="purple" />
```

Output:



React Events

If you have coded even a little bit in javascript, you know the importance of events.

Events

Every HTML attribute in React is written in camelCase syntax. Event is also an attribute. Hence, written in camelCase.
As we learnt variables, states, javascript operations are written in curly braces `{}`, same is true with React event handlers too! Like this: `onClick={show}`

```
<button onClick={show}>Show</button>
```

Arguments in events

We can't pass arguments just like that, it will give syntax error. First, we need to put the whole function in arrow function, like this:

```
<button onClick={ ()=>show('true') }>Show</button>
```

React Event Object

Event handler can be provided to the function like this:

```
<button onClick={ (event)=>show('true', event)
}>Show</button>
```

React Router

React router is used for page routing as React App doesn't include it as default.

Add React Router

To install React router, run this in your terminal:

```
npm i -D react-router-dom
```

Creating Multiple Routes

To do this first we need to create multiple files and to keep code clean, we will make a folder and make all the pages there, hence we will create a folder named `pages` in `src`.

Folder Structure:

`src/pages/`:

- `Home.js`
- `Blogs.js`
- `Contact.js`
- `Nopage.js`

Usage

Now we will make routes in `src/index.js`, like this:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import reportWebVitals from './reportWebVitals';
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Home />} />
        <Route path="/blogs" element={<Blogs />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<NoPage />} />
      </Routes>
```

```
    </BrowserRouter>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

   reportWebVitals();
```

Home.js

Make all the Navigation links using `<Link>` tag, like this:

```
import { Link } from "react-router-dom";

const Home = () => {
    return (
        <>
            <nav>
                <ul>
                    <li>
                        <Link to="/">Home</Link>
                    </li>
                    <li>
                        <Link to="/blogs">Blogs</Link>
                    </li>
                    <li>
                        <Link to="/contact">Contact</Link>
                    </li>
                </ul>
            </nav>
        </>
    )
};

   export default Home;
```

Blogs.js

```
const Blogs = () => {
    return <h1>Blogs</h1>;
};

   export default Blogs;
```

Contact.js

```
const Contact = () => {
    return <h1>Contact</h1>;
};

   export default Contact;
```

NoPage.js

```
const NoPage = () => {
    return <h1>404</h1>;
};

   export default NoPage;
```

This is made for any route that doesn't exist. To show `404` error there!

React CSS Styling

There are three ways to style in React:

1. Inline Styling
2. CSS Stylesheets
3. CSS Modules

Inline Styling

To inline style an element, we can make a Javascript Object, like this:

```
const App = () => {
  return (
    <>
      <h1 style={{ backgroundColor: "purple" }}>CodeWithHarry</h1>
    </>
  );
   }
```

In this first curly brace is to write javascript and second is to make a javascript object. We can also write it like:

```
const App = () => {

  const h1Style = {
    backgroundColor: 'purple',
    color: 'white'
  }

  return (
    <>
      <h1 style={h1Style}>CodeWithHarry</h1>
    </>
  );
   }
```

Note: CSS property name must be camelCase. `background-color` would be `backgroundColor`

CSS Stylesheets

You can save the whole CSS in a separate file with file extension `.css` and import it in your application.

`App.css`

```
body {
```

```
  background-color: 'purple';
  color: 'white';
  }
```

Here we are writing CSS, so we don't need to make JS Object or do it in camelCase.
`Index.js`
Just import it like this:

```
import './App.css'
```

CSS Modules

In this you don't have to worry about name conflicts as it is component specific. The CSS is available only for the file in which it is imported.
Make a file with extension `.module.css`, example: `index.module.css`
Make a new file `index.module.css` and insert some CSS into it, like this:

```
.button {
  background-color: 'purple';
  color: 'white';
  }
```

Import it in component like this:

```
import styles from './index.module.css';

const Home = () => {
    return (
        <button className={styles.button}>Click me!</button>
    )
};

    export default Home;
```

React useState Hook

`useState` is a Hook that lets you add React state to function components.

Importing `useState`

To use useState, first we need to import `useState` and initialize it, you can `import` it from `react` like this:

```
import { useState } from "react";
```

Initializing `useState`

You can initialize state like this:

```
import { useState } from "react";

const App = () => {
    const [name, setName] = useState('')
};
```

`useState` takes initial state as argument and gives a state and a function(`setName` in this case) to update that state as we can't directly change/update a state. Also, these state names are just like variables, hence you can name them anything you like.

Reading a state

As mentioned earlier, it returns a state and a function to change/update that state. Hence, everything is stored in `name`. We can read states just like variables:

```
import { useState } from "react";

const App = () => {
    const [name, setName] = useState('')

    return <h1>My name is {name}</h1>
};
```

Updating a state

To update state we use the function it returns to update state, in this case: `setName`. State can be updated like this:

```
import { useState } from "react";
```

```
const App = () => {
    const [name, setName] = useState('')
    setName('Lovish')

    return <h1>My name is {name}</h1>
  };
```

What can state hold?

Like normal variables, state can hold any datatype like strings, numbers, booleans, arrays, objects, objects in arrays, arrays in objects. For example:

```
import { useState } from "react";

const App = () => {
    const [data, setData] = useState({
        name: 'Lovish',
        age: 21
    })

    return <h1>My name is {data.name} and my age is
{data.age}</h1>
  };
```

Updating Objects and Arrays in State

```
import { useState } from "react";

const App = () => {
  const [data, setData] = useState({
    name: 'lovish',
    age: 21
  })

  return <>
    <h1>My name is {data.name} and my age is {data.age}</h1>
    <button onClick={() => setData({ ...data, name: 'CWH'
})}>Click Me</button></>
};

  export default App;
```

React useEffect Hook

The `useEffect` Hook allows you to perform side effects in your components.

`useEffect` accepts two arguments. The second one is optional.

Runs on every render:

```
import { useState, useEffect } from "react";

function Home() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I have rendered {count} times!</h1>;
}

   export default Home;
```

Runs on first render:

```
import { useState, useEffect } from "react";

function Home() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  }, []);

  return <h1>I have rendered {count} times!</h1>;
}

   export default Home;
```

Runs when `data` changes:

```
import { useState, useEffect } from "react";

function Home() {
```

```
  const [count, setCount] = useState(0);
  const [data, setData] = useState('');

  const handleChange = (e) => {
    setData(e.target.value)
  }

  useEffect(() => {
    setCount((count) => count + 1);
  }, [data]);

  return (
    <>
      <input onChange={handleChange} value={data} />
      <p>{count}</p>
    </>
  );
}

   export default Home;
```