**React**

React is a JavaScript library for building a fast and interactive user interface. Facebook released this in 2011. It allowed developers to create complex UIs by integrating small snippets of code.

Even though some people call this a framework because of its behaviors and capabilities but in reality it is a library.

For more information, please read this - https://reactjs.org/

**React Components**

When building an application using React, you will basically build a bunch of reusable components. So if you were to build an application, you build these individual components first. The parent component (known the root component) holds all the other components.

**The concept of virtual DOM**

This is a programming concept where the “virtual” representation of a UI is kept in memory and is synced with the real DOM by a library such as ReactDOM. You tell React what state you want the UI to be in, and it makes sure the DOM matches that state.

For more information, please read this - https://reactjs.org/docs/faq-internals.html

**Setting up the work environment**

Make sure you have nodejs and npm installed on your computer.

To check the versions of nodejs and npm –

In VS Code Terminal, type in –

nodejs –version

npm –version

Create a new folder – folderName

In the VS code Terminal type in –

*npx create-react-app my-react-app*

This will prompt you – type Y and Enter

*cd my-react-app*

*npm start*

This will compile your react app and display it in your browser – localhost:3000

**Example 1: Using React without the Server**

Include the 3 CDNs in your HTML document and it will work fine.

HTML

<!DOCTYPE html>

<html>

  <head>

    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

  </head>

  <body>

    <div id="mydiv"></div>

    <script type="text/babel">

      function Hello() {

        return <h1>Hello World!</h1>;

      }

      ReactDOM.render(<Hello />, document.getElementById('mydiv'))

    </script>

  </body>

</html>

<https://www.w3schools.com/REACT/react_getstarted.asp>

**Using the Create React App CLI**

Make sure you have nodejs and npm installed on your computer.

To check the versions of nodejs and npm –

In VS Code Terminal, type in –

Nodejs –version

Npm –version

Create a new folder – reactApp\_1

In the VS code Terminal type in –

*npx create-react-app my-react-app*

This will prompt you – type Y and Enter

*cd my-react-app*

*npm start*

This will compile your react app and display it in your browser – localhost:3000

Now open up **App.js** (inside the src sub-folder)

Replace – Learn React – with something like – Your Name

You will notice, as soon as you change the text, it will update it in the browser.

Now let’s make some more changes –

Let’s replace the existing code with the following –

import './App.css';

function App() {

  return (

      <header className="App-header">

          John

      </header>

  );

}

export default App;

**Example 2: Creating your first component**

Create a new project.

Create a new folder – example2

In the VS code Terminal type in –

*npx create-react-app component-app*

This will prompt you – type Y and Enter

*cd component-app*

*npm start*

Open the **index.js file** (inside the src folder).

Make the following changes to it –

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

function Car() {

  return <h2>Hi, I am a Car!</h2>;

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Car />);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

So you created your first React component and it is named car.

**Props**

Components can be passed as props – which is basically properties. Props are like function arguments and you send them into the component as attributes

Now let’s open up the index.js file and make the following changes –

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

function Car(props) {

  return <h2>I am a {props.color} Car!</h2>;

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Car color="red"/>);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

<https://www.w3schools.com/REACT/react_components.asp>

**Adding Styling**

Here is an example of how you can use inline styling.

**Open the index.js file and make the following changes –**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const Header = () => {

  return (

    <>

      <h1 style={{color: "red"}}>Hello Style!</h1>

      <p>Add a little style!</p>

    </>

  );

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Header />);

Here is an example of using external style sheet. Open the App.css (inside the src folder) and paste the following code at the bottom of the page –

body {

  background-color: #282c34;

  color: white;

  padding: 40px;

  font-family: Sans-Serif;

  text-align: center;

}

https://www.w3schools.com/REACT/react\_css\_styling.asp

**Example 3: React Render HTML**

In many ways, the main goal of React is to render HTML on a web page. There is special function called ReactDOM.render(). It takes two arguments – HTML code and an HTML element.

In fact, in the recent version of React, it is deprecated. Use createRoot instead.

The React application will be rendered in the <div> that is in the body of the following file.

Open **index.html** (that is inside the folder public)

When you scroll down, you will notice the following (line 31)–

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

Open **index.js** file (inside the src folder).

Copy and paste the following code –

import React from 'react';

import ReactDOM from 'react-dom/client';

const myFirstElement = <h1>Hello React!</h1>

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myFirstElement);

You can even try this –

import React from 'react';

import ReactDOM from 'react-dom/client';

const myelement = (

  <table>

    <tr>

      <th>Name</th>

    </tr>

    <tr>

      <td>John</td>

    </tr>

    <tr>

      <td>Elsa</td>

    </tr>

  </table>

);

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(myelement);

**Example 4: Using an Array and concatenating with some text**

*npx create-react-app my-app*

Give it a minute or two to install everything. Please note, my-app is the name of the folder that you created. It should have all the required files to build your app.

To open your project folder –

*cd my-app*

To run the app in development mode

*npm start*

You should see the browser open the page – <http://localhost:3000>

Open the index.html file which is inside Public Folder > Index.html

**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" />

    <!--

      manifest.json provides metadata used when your web app is installed on a

      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/

    -->

    <link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

    <!--

      Notice the use of %PUBLIC\_URL% in the tags above.

      It will be replaced with the URL of the `public` folder during the build.

      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

      work correctly both with client-side routing and a non-root public URL.

      Learn how to configure a non-root public URL by running `npm run build`.

    -->

    <title>React App</title>

  </head>

  <body>

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

    <div id="root"></div>

    <!--

      This HTML file is a template.

      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.

      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.

      To create a production bundle, use `npm run build` or `yarn build`.

    -->

  </body>

</html>

Now open the **App.js** file (in the src folder). Delete all the code and replace it with the following -

**App.js**

import logo from './logo.svg';

import './App.css';

export default function App() {

  const continents = ["Africa", "America", "Asia", "Australia", "Europe"];

  let helloc = continents.map(function(c){

    return "Hello".concat(" ",c, "!");

  })

  let message = helloc.join(" ");

  return (

    <div className="App">

      <p>{message}</p>

    </div>

  );

}

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

**Example 5: Issue Tracker App**

The goal of this application is to display the UI of the main page of the Issue Tracker app containing a list of issues and other information.

In this example, we will see how to pass data from the parent component to a child component and also dynamically created components.

If you've previously installed create-react-app globally via npm install -g create-react-app, you can uninstall the package using npm uninstall -g create-react-app to ensure that npx always uses the latest version.

*npx create-react-app my-app*

Give it a minute or two to install everything. Please note, my-app is the name of the folder that you created. It should have all the required files to build your app.

To open your project folder –

*cd my-app*

To run the app in development mode

*npm start*

You should see the browser open the page – <http://localhost:3000>

**Issue Tracker**

Open the index.html file which is inside Public Folder > Index.html

*Please note, you can use any HTML editor or notepad to open the index.html file.*

Delete all the code in the HTML page and replace it with the following –

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <meta name="theme-color" content="#000000">

  <!--

      manifest.json provides metadata used when your web app is added to the

      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/

    -->

  <link rel="manifest" href="%PUBLIC\_URL%/manifest.json">

  <link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico">

  <!--

      Notice the use of %PUBLIC\_URL% in the tags above.

      It will be replaced with the URL of the `public` folder during the build.

      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

      work correctly both with client-side routing and a non-root public URL.

      Learn how to configure a non-root public URL by running `npm run build`.

    -->

  <title>React App</title>

</head>

<body>

  <noscript>

    You need to enable JavaScript to run this app.

  </noscript>

  <div id="root"></div>

  <!--

      This HTML file is a template.

      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.

      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.

      To create a production bundle, use `npm run build` or `yarn build`.

    -->

</body>

</html>

Now open the **App.js** file (in the src folder). Delete all the code and replace it with the following -

import './App.css';

//import ReactDOM from 'react-dom';

import React from "react";

//const contentNode = document.getElementById('root')

class IssueFilter extends React.Component {

    render() {

        return (

            <div>This is a placeholder for the Issue Filter.</div>

        )

    }

}

class IssueAdd extends React.Component {

    render() {

        return (

            <div>This is a placeholder for an Issue Add entry form.</div>

        )

    }

}

class IssueList extends React.Component {

    render() {

        return (

            <div>

                <h1>Issue Tracker</h1>

                <IssueFilter/>

                <hr />

                <IssueTable issues={issues}/>

                <hr />

                <IssueAdd/>

            </div>

        );

    }

}

class IssueRow extends React.Component {

    render() {

        const issue = this.props.issue;

        return (

            <tr>

                <td>{issue.id}</td>

                <td>{issue.status}</td>

                <td>{issue.owner}</td>

                <td>{issue.created.toDateString()}</td>

                <td>{issue.effort}</td>

                <td>{issue.completionDate ?

                issue.completionDate.toDateString() : ''}</td>

                <td>{issue.title}</td>

            </tr>

        )

    }

}

class IssueTable extends React.Component {

    render() {

        const issueRows = this.props.issues.map(issue => <IssueRow

        key={issue.id} issue={issue} />)

        return (

            <table className="bordered-table">

                <thead>

                    <tr>

                        <th>Id</th>

                        <th>Status</th>

                        <th>Owner</th>

                        <th>Created</th>

                        <th>Effort</th>

                        <th>Completion Date</th>

                        <th>Title</th>

                    </tr>

                </thead>

                <tbody>{issueRows}</tbody>

            </table>

        )

    }

}

const issues = [

{

    id: 1, status: 'Open', owner: 'Raven',

    created: new Date('2016-08-15'), effort: 5, completionDate: undefined,

    title: 'Error in console when clicking Add',

},

{

    id: 2, status: 'Assigned', owner: 'Eddie',

    created: new Date('2016-08-16'), effort: 14,

    completionDate: new Date('2016-08-30'),

    title: 'Missing bottom border on panel',

},

];

function App() {

    return <IssueList />;

}

export default App;

Save the files and test it in the browser.

You will notice, we have a few class components – IssueFilter, IssueAdd, IssueList, etc.

The components have to include the extends React.Component statement. This basically creates an inheritance to React Component and gives you access to React.Component’s functions.

The component also requires a render() method – to return HTML

<https://ibaslogic.com/react-tutorial-for-beginners/>

<https://www.guru99.com/reactjs-tutorial.html>

<https://reactresources.com/articles>

<https://www.w3schools.com/REACT/react_getstarted.asp>