REACT.JS

React is a javascript framework for building single page or component based application. It just like Angular but react.js doesn’t give you the opportunity to separate your program logic from the template, which angular does.

INSTALL REACT AND CREAT APP

We use this tool "create-react-app" for creating react applications.

If you don't have it, install it globally with this command on your command-prompt (make sure you have nodejs installed first):

npm install -g create-react-app

Then, open your command prompt as administrator and create a react app with this command:

npx create-react-app myapp

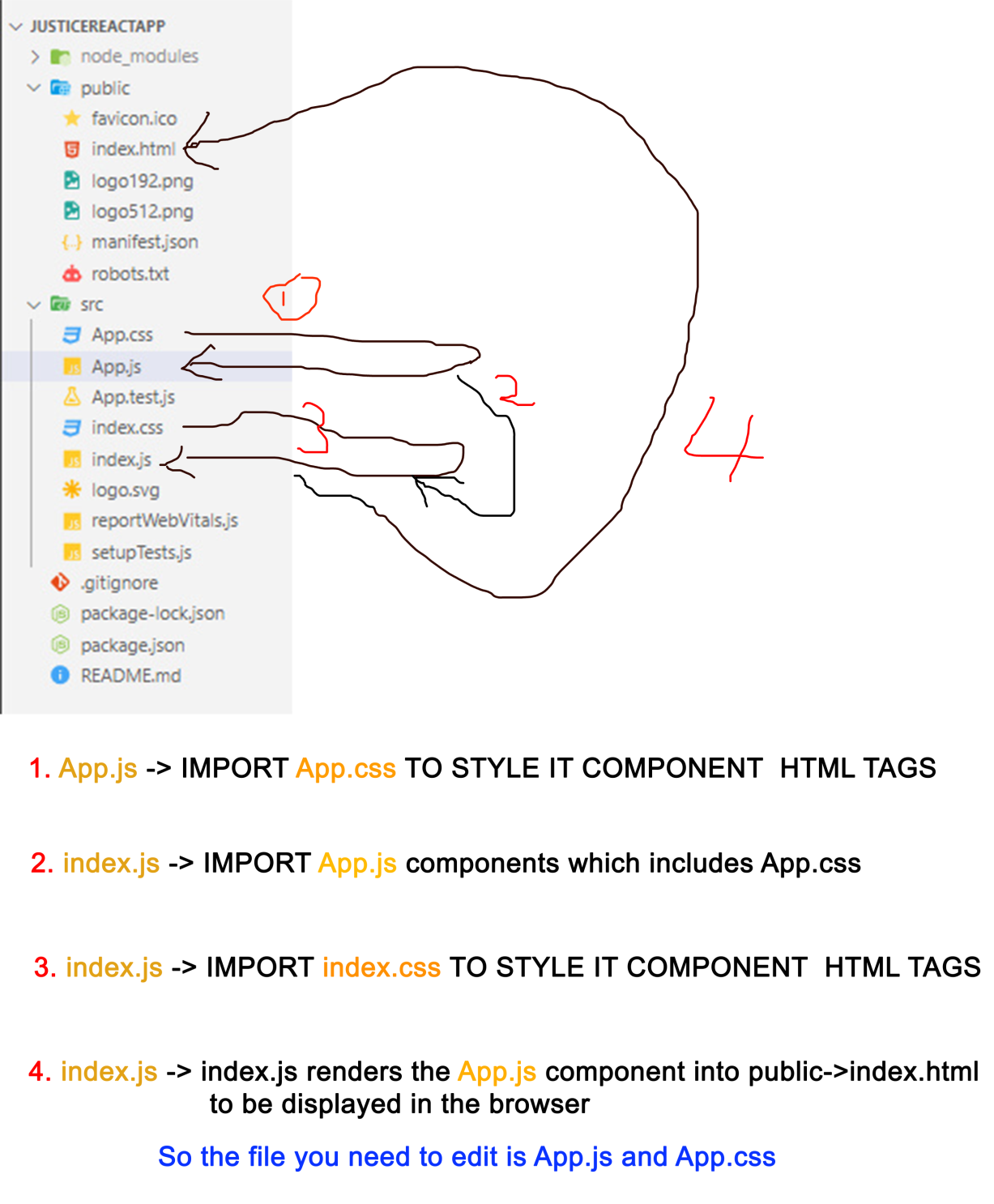
Then navigate to the app folder and start the app with the command:

npm start

====================================================

React Project Structure

You can expand below image and it really easy to understand react project structure.



PUBLIC -> MANIFEST.JSON

manifest.json provides metadata used when your web app is installed on a user's mobile device or desktop.

To see all your installed packages open the package.json file and check the dependencies object

{

// My installed dependencies or packages

"dependencies": {

    "@testing-library/jest-dom": "^5.14.1",

    "@testing-library/react": "^11.2.7",

    "@testing-library/user-event": "^12.8.3",

    "react": "^17.0.2",

    "react-dom": "^17.0.2",

    "react-scripts": "4.0.3",

    "web-vitals": "^1.1.2"

  },

  // the “scripts” object represent my development server commands

  "scripts": {

    "start": "react-scripts start",   // to start the app (it builds local development version of my react app)

    "build": "react-scripts build",  // build a production version of my react app so that you can deploy it to a server

    "test": "react-scripts test",

    "eject": "react-scripts eject"  // never use or run this eject script

  },

}

====================================================

REACT RENDER

The reactDom.render() method allows you to take one html element and put it into another.

import react from 'react';

import reactDOM from "react-dom";

  ReactDom.render(   <h1> helloo world</h1>, document.getElementByid("book")    );

// Above means, look for document with id "book" and put <h1> hello world</h1> into it. "REACT RENDER" is all about taking one html and putting it into another html element

// Remember the first ReactDom.render() parameter only takes one html container. to return multiple values, wrap them inside a single container eg..

Or wrap it inside a "jsx fragment (<> </>)" or a div Eg.

render(){

  return ( <> <h2>justice ankomah </h2> <p id=”book”> age 43 </p> </>);

}

====================================================

JSX – JAVASCRIPT XTENSION

Jsx helps you to write html codes directly in your react.js components and functions.

Although they look like actual html, but they are not, their jsx. It just that they use the same html tags and the same syntax.

JSX Rules For

/\*  ========== Rule 1:

You can only return a single html container tag.

If you need to return multiple html tags, wrap it inside a container like div or wrap it in JSX fragment (<> </>)

Example:

\*/

function myApp(){

  return (

    // jsx fragment

    <>

        <p>one</p>

        <p>ttwo</p>

    </>

  );

}

/\*  ========== Rule 2:

Use "className" to define class attribute for your jsx element. Because "class" is a reserved word in javascript.

But you can use "id" attribute because it not a reserved word

Example:

\*/

function myApp(){

  var myName = "justice";

  return (

    <>

        <p className="myparagraph">one</p>

     /\* you can use a variable name by surrounding it with {} instead of a String  \*/

        <p className={myName}>one</p>

    </>

  );

}

====================================================

Component

Component represents single html webpage with its own template, logic and styles that can be embedded into other components.

Note = When you create a component, start with a capital letter like App.js. And create a separate CSS file for it to style it templates.

You can create component by using functions or class.

**Example:**

// import React

import React from 'react';

// ========== CLASS BASED COMPONENTS ===============

// create a class that extends "React.Component" so you can have access to the render(){} method

class JusticeComponent extends React.Component{

  // the render method represent what will be return from this component to other components

  render(){

    return(

      <p>This is my first class component</p>

    );

  }

}

// export the class

export default JusticeComponent;

// ========== Function BASED COMPONENTS ===============

function myApp(){

  return(

<p>My first function based components</p>

  );

}

// export the function

export default myApp;

Props or Properties

Props, allows you to pass data to another components.

Example:

Inside Bacon.js

import React from "react";

class Bacon extends React.Component{

  // call the class constructor to initialize the props

  constructor(props){

   super(props);

  }

  render(){

    return (

     <div>

           /\* tell where you want to display the "name" and "country" properties that is being passed to this components.

           Use the format {this.props.property\_Name}  to access the properties\*/

      <h2> {this.props.name} </h2>

      <p> {this.props.country} </p>

    </div>

         );

  }

}

export default Bacon;

Inside App.js

// import the Bacon component

import Bacon from "./components/texting";

function App() {

  return (

    // pass in properties to the <Bacon> template and display it here

       <Bacon name="Iniesta" country="Ghana"></Bacon>

  );

}

export default App;

How to map through data and pass it to component

Inside App.js

// import the Bacon component

import Bacon from "./components/texting";

var students = [

  {firstName:"justice", lastName:"Ankomah", age: 20},

  {firstName:"Abert", lastName:"Aidoo", age: 2},

  {firstName:"Dennis", lastName:"Obodai", age: 15}

];

function App() {

  return (

    // map through the students array and pass the data

    // pass in properties to the <Bacon> template and display it here

    students.map(function(studentsDetails){

     return <Bacon firstname={studentsDetails["firstName"]} lastname={studentsDetails["lastName"]} age={studentsDetails["age"]}></Bacon>;

    })

  );

}

export default App;

Inside Bacon.js

class Bacon extends React.Component{

  // call the class constructor to initialize the props

  constructor(props){

   super(props);

  }

  render(){

    return (

     <div>

           {/\* tell where you want to display the  properties that is being passed to this components. \*/}

      <h2> {this.props.firstname} </h2>

      <p> {this.props.lastname} </p>

      <p> {this.props.age} </p>

    </div>

         );

  }

}

export default Bacon;

How To Validate And Set default Value for your Props (PropTypes)

Example:

// inside App.js

// import Header.js

import Header from "./components/header/Header";

function App() {

 return (

   <>

  <p>Am App Js</p>

  <Header username="Kenneth" height="44"> children of header </Header>

   </>

 );

}

// export the components

export default App;

// inside Header.js

// improt PropTypes

import {PropTypes} from "prop-types";

export default function Header(props){

  // define list of your props

  const {username, age, height} =props;

  return(

    <>

     {/\* display the props here \*/}

  <p>my name, age and height is: {username + " " + age + " " + height}</p>

  {/\* get the parameter passed into this component open tag eg. <Header>this is the children’s to get</Header> \*/}

  <p>{props.children}</p>

    </>

  );

}

// default parameters, this means if no parameter is passed into these props, use these default values.

Header.defaultProps={

 username: "justice",

 age: 22

}

// validate your props (remember, the "propTypes" is of small "p")

Header.propTypes={

  // means, "username" must be of dataType "string" and is required (remember, the "propTypes" is of Capital "p")

  username: PropTypes.string.isRequired,

  age: PropTypes.number,

  // element represent: innerHTML (the property for accessing html element text values)

  children:PropTypes.string.isRequired,

  // meaning height can be of type string or number but it must be requird

  height: PropTypes.oneOfType([PropTypes.string, PropTypes.number]).isRequired

}

// there is a lot of proptypes. you can check react.js documentation for mores

/\*\* Result:

 Am App Js

my name, age and height is: Kenneth 22 44

children of header

 \*/

====================================================

EVENT HANDLING

Event helps you to do things to Dom element when some action like “click”, “hover” is performed on them.

Note = You can only add event to real jsx/html element but not imported component.

**Example:**

import Header from "./components/header/Header";

// event handler "evt" represent the event object

function myEvent(evt){

  console.log(evt.target);

  alert("am event");

}

function App(){

  return (

    <div>

      /\* this will not work, event should be added to actual html/jsx elements not components \*/

      <HeaderComponent  onClick={myEvent}></HeaderComponent>

     /\* this will work \*/

      <p onClick={myEvent}>click me</p>

    </div>

    );

}

====================================================

State management

State are used for setting component properties or variables and objects. React uses the useState() function for state management.

useState() => function that takes initial value. The value can be of any data-Type string, array or whatever.

const [colors, setColors] = useState(“default\_value”).

Colors = represent the default value in useState()

setColor => is a function that is used to update the value in useState() "it takes in the new value you want to set for the seState()"

**Example:**

import React from "react";

// =======Example 1:

function App() {

  // the data-Type of colors is String Because of "" in useState() method below

  const [colors, setColors] = useState("");

return (

  <div>

 {/\* when the button is clicked update the useState() value which is "colors"

whenever this button is clicked the value will change to "JUSTY" \*/}

    <button onClick={(e)=>{ setColors("JUSTY");}} > click me </button>

{/\* set the useState() value which is "colors" to this element  \*/}

    <p>{colors}</p>

</div>

);

}

// ===========Example 2:

function App() {

 // here useState() takes in an array, so "value" is an array. and 1,2,3,4 is the default value set to it

 const [value, setValue] = useState([1,2,3,4]);

return (

   <div>

    {/\* When the button is clicked set this as the  useState() value which is "value" \*/}

    <button onClick={function(e){setValue([value, 5]);}}>click me</button>

         {/\* loop through the  useState() value and set it to this element \*/}

    <p>{value.map(function(e){return <span>{e}</span>;})}</p>

</div>

  );

}

====================================================

useEffect() hook

It’s a react hook that will be called whenever a specific component get rendered or any of it properties changes.

It just like the build() method in flutter. When the component first get rendered it calls it, and whenever the components data changes it calls it.

You can use it to make https request and access local storage and do many things.

The useEffect() method takes in 2 parameters.

1. First parameter = represent a function that will be called whenever the component is rendered.
2. Second parameter = it represent when you want to run the useState().

no parameter = means run it whenever the page get rendered and when any of it data changes.

[] = means run the useState() only when the component get rendered the first time. It will not run when it data changes

Some value = means, run it whenever that value changes.

function App() {

// run this method whenever the page component rendered and whenever the component data changes because there is no second parameter

 useEffect(

// first parameter

()=>{

  console.log("finished re");

}

);

 return (

<p> helloo</p>

 );

}

====================================================

Routing

Routing helps you to navigate from one component to the another instead of using html anchor tags.

First install this package by running it on your CMD: npm install --save react-router-dom

**Example:**

Inside index.js

import App from './App';

import reportWebVitals from './reportWebVitals';

// IMPORt these from the package we just installed

import { BrowserRouter} from "react-router-dom";

ReactDOM.render(

  // wrap your top-most component "App.js" with BrowserRouter to provide navigation functionality in your entire app

  <BrowserRouter>

      <App />

  </BrowserRouter>,

  document.getElementById('root')

);

// inside App.js

import Header from "./components/header/Header";

// IMPORt these from the package we just installed

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

function App() {

 return (

   <>

   {/\* Use the <Link> imported above it just like <a href="to">Home</a>

   These represent the link users will click on to navigate to the path in the "to" attribute \*/}

       <Link to="/home">Home</Link>

       <br/>

        <Link to="/contact">contact</Link>

        <br/>

        <Link to="/footer">Footer</Link>

        {/\* this is the component that determins what component should be navigated to when any of the above link is clicked \*/}

        <Header />

   </>

 );

}

// export the components

export default App;

// inside Header.js

// IMPORt these from the package we just installed

import { Route, Switch } from "react-router-dom";

// import the other components

import Footer from "../footer/Footer";

import Contact from "../contact/Contact";

export default class Header extends React.Component{

    render(){

        return (

          <div>

            <p>am the header</p>

          {/\* Switch is like a switch case. Each Route is a single case of it. It says something like

          hei, check each single Route path and render the Component in that Route \*/}

<Switch>

{/\* "exact" if the link matches the same with this "path" render the component in the Route \*/}

   {/\* first case: \*/}

   <Route path="/footer" exact><Footer/></Route>

   {/\* second case: \*/}

    <Route path="/contact" exact><Contact/></Route>

{/\* default case: this is what will be rendered when none of the path above matches the link\*/}

   <Route path="" render={

     ()=>{

     // you can write some logic here before returning the component

     return <Header/>;

   }} />

  </Switch>

          </div>

         );

      }

}

How To pass parameter to a route

You can use the useParams() method to grab parameters passed into route. Example:

// inside App.js

import Header from "./components/header/Header";

// IMPORt these from the package we just installed

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

function App() {

 return (

   <>

    {/\* pass parameter to the router that is "/us" and "/me" \*/}

        <Link to="/contact/us">contact me</Link>

       <br/>

        <Link to="/contact/me">contact me</Link>

        <br/>

        <Link to="/footer">Footer</Link>

        <Header />

   </>

 );

}

// export the components

export default App;

// inside Header.js

// IMPORt these from the package we just installed

import { Route, Switch} from "react-router-dom";

// import the other components

import Footer from "../footer/Footer";

import Contact from "../contact/Contact";

export default class Header extends React.Component{

    render(){

        return (

          <div>

<Switch>

   <Route path="/footer" exact><Footer/></Route>

   {/\* secound case: the "id" represent the parameter being passed to this route \*/}

    <Route path="/contact/:id" > <Contact/></Route>

{/\* default case: this is what will be rendered when none of the path above matches the link\*/}

   <Route path="" render={

     ()=>{

     return <Header/>;

   }} />

  </Switch>

          </div>

         );

      }

}

// inside Conctact.js

// import use parames

import {useParams } from "react-router-dom";

export default function Contact(){

  // grap the parameter here

// the id should be the same as the one define in the Route

  const {id} = useParams();

    return (

        // display the parameter which is the "id" above

        <div >am the Contact Component {id}</div>

    );

}

====================================================

useRef

In react, you can’t access Dom element like document.getElementById(“p”) as you would do in vanilla JavaScript.

So we use refs to hold reference to Dom element like <p> and then use it to access the element properties and methods. The ref has a special property called “current” which represent the html/jsx element the ref will be assigned to.

Note = refs do not update a state. So when your refs value changes it will not reflect in your Dom. You can use useState() together with it.

**Example:**

// import useRef from react first

import {React, useRef} from "react";

export default function Header(){

  // create a ref

  let myInputRef = useRef();

  function check(){

    //  myInputRef.current represent the <input> element itself that "myInputRef" has been assigned to.

    // focus() means move the mouse pointer to this element

    myInputRef.current.focus();

    // print the value of <input> tag. You can use myInputRef.current to access it all properties and methods

   console.log(myInputRef.current.value);

  }

  return(

    <>

    {/\* assign your useRef to this <input> tag using the attribute "ref" \*/}

    <input type="text" ref={myInputRef}/>

    <button onClick={check}>Click me</button>

    </>

  );

}

forwardRef

ForwardRef allows you to pass a container of ref into others.

Example:

// inside App.js

// import useRef from react first

import {React, useRef} from "react";

// import Header.js

import Header from "./components/header/Header";

function App() {

    // create a ref

    let myInputRef = useRef();

    function myOnchangeFunction(){

     console.log(myInputRef.current.value);

    }

 return (

   <>

  <p>Type something below:</p>

   {/\* pass the above "myInputRef" and the "myOnchangeFunction" function as props \*/}

   <Header ref={myInputRef} onchangefun={myOnchangeFunction} />

   </>

 );

}

// export the components

export default App;

// inside Header.js

import {React, forwardRef} from "react";

// every component accept exactly two parameters, "props" and "ref"

// the params in { } are props and the other is ref

function Header({onchangefun}, ref){

  return(

    <>

    {/\* assign the props and ref passed into your tags \*/}

    <input type="text" onChange={onchangefun} ref={ref} />

    <button onClick={onchangefun}>Click me</button>

    </>

  );

}

// wrap your component in forwardRef

const myforwardInput = forwardRef(Header);

// export the component (other component needs to import the Header)

export default myforwardInput;

====================================================

**Createcontext() and useContext() Hook**

Context is a way of passing data down to multiple imported components.

The concept is that, if you have 5 imported components in your function, and you want to pass some single data to them all, you have to pass props to them all.

But you can use the context hook to wrap those component to let them have access to the context value.

**Example::**

// inside context.js

// create a new file called context.js

// import createContext

import {createContext} from "react";

// create a context

const nameContext = createContext();

// export the context

export {nameContext};

// inside App.js

// import Header.js

import Header from "./components/header/Header";

import Footer from "./components/footer/Footer"

// import the nameContext in the file above

import {nameContext} from "./context";

function App() {

    return (

        //wrap all your imported component with the context and specify a value for the context

       // all the component below wrapped in the context will have access to the context value

   <nameContext.Provider value="Justice Ankomah">

   <p>inside app.js</p>

   <Header/>

   <Footer/>

   </nameContext.Provider>

    );

}

// export the components

export default App;

// inside Header.js

// import useContext

import {React, useContext} from "react";

// import nameContext from context file

import {nameContext} from "../../context";

function Header(){

  // pass the context to the useContext() method

  const name = useContext(nameContext);

  return(

    <>

 <p>inside heading</p>

 {/\* show the context here \*/}

<p>helloo {name}</p>  {/\* result: justice ankomah \*/}

    </>

  );

}

export default Header;

// inside Footer.js

// import useContext

import {React, useContext} from "react";

// import nameContext from context file

import {nameContext} from "../../context";

function Footer(){

  // pass the context to the useContext() method

  const name = useContext(nameContext);

  return(

    <>

 <p>inside footer</p>

 {/\* show the context here \*/}

<p>helloo {name}</p>  {/\* result: justice ankomah \*/}

    </>

  );

}

export default Header;

So you can see from the above example that both, Header and Footer component have access to the single context value that is passed in the App.js file.

====================================================

How to Create a Production Build Version Of React Apps

To deploy your app to a real live hosting server, you need to build a production version of your app.

What happens is that, react will build and generate a production version of your app in a folder named “build” this is the folder to deploy to a hosting server.

1. Frist navigate to your project path and run this command on your cm to build production verstion: npm run-script build

This will create a “build” which you need to deploy to a hosting server

1. Test the build folder with VS-code live server extension with this command: npm build serve -s build

====================================================

React Developer Tools

There is a chrome extension called "React Developer Tools" that helps you to see what is going on in your component rendring.

Download that and use it for development.