Going by the name of the project, it is a react app for a calculator. In the root directory we find the public and src folders along with some files that set the environment for react app development.

In the public folder we have the favicon.ico, index.htm and manifest.json files. The faveicon.ico is an icon. This icon is used to display the app logo when the app is bookmarked by a browser (added as a favourite).

The index.html file is where our react app typically starts. It contains the basic layout of to be displayed by the browser. In this HTML file we have a div tag with its id set to root. This is very important because this root div is used by the index.js file to render our application. Besides this root div tag we have the basic elements of an HTML file. The Meta charset element that sets the character encoding of our page to utf-8. Utf-8 can handle almost all of the languages. Then we have the Meta view port element that sets the users display size according to their device and disables zooming. After the Meta elements we have the page title and then a noscript element that tells the user to enable JavaScript to run this app if they have JavaScript enabled. After that we have our root div and then an attribute element that displays an alternate text if the app does not work properly.

After index.html we have the manifest.json file. This file sets the behaviour of our app. It contains the application’s name and icon. Furthermore it contains the starting point of our react app. This sums up the public folder in our root directory. **This sums up the public folder of our root directory.**

The src folder in the root directory contains a component and logic folder. Besides that we have index.css and index.js files. The component folder contains our app components JavaScript code and their CSS style sheets. The JavaScript code and the CSS in components folder is mainly regarding displaying or rendering the app components properly. The logic folder contains JavaScript codes for the mathematical operations to be done on each button press and the actual logic behind the working of the calculator.

The isNumber.js file in the logic folder exports a function called isNumber. This function takes an item as a parameter (the number input by the user will be the item in our case) and uses a regular expression to check if the item is a number or not. It returns true if the item is a number and false if the item is not a number.

The operate.js file in the logic folder imports big.js. Big.js is a library that handles numbers and maths. Operate.js file exports a function that takes two numbers and an operator as parameters then, it checks what operation needs to be performed on the two numbers. If it is a valid operation then it performs the operation and returns the result after changing it to string. It will throw an error if an invalid is operation is being requested to be performed.

The calculate.js in the logic folder has imported big.js, “isNumber” function from isNumber.js file and “operate” function from operate.js file. Calculate.js mainly contains a function that takes an obj (object name) and buttonName (button name) as its parameters. Then it checks what the button name is and performs the changes accordingly. For example if the button name is AC (all clear) then this function will return Null to all the properties of the object whose name has been passed in into the parameters. In this manner it will clear all the values the object had. Then the calculate function calls the isNumber function. If isNumber returns true then it checks if the number is 0 and in case the number is 0 then nothing needs to be changed so the function returns nothing. Then this function has another conditional if statement that uses the obj.next method to check whether there is another value available. If there is another value present then it concatenates that value with the button name and returns it. Calculate.js is a function that is basically being used to change the values of objects. Then these changes will be passed onto the App.js (our main app renderer) so that they may be reflected properly on the screen.

The calculate.test.js is our testing file. It imports the calculate function from calculate.js and chai library from chai. Chai is a framework for testing purposes. The calculate.test.js file will test our calculate function. It has function called pressButtons that takes a button as a parameter. Then it has a function called expectButtons that takes a button and its expecations as parameters. Lastly, we have the describe method. The pressButtons tests if the buttons are working properly. The describe method is a wrapper that acts as a header when the test results are shown. The header has been set to “calculate” which tells us that in this describe’s body we will be testing the calculate function. The describe method uses random values that have been set by the developer and it calls the calculate function and passes these values as parameters and then tests if their output is correct logically. It checks the output of all the mathematical operations and also makes sure that double pressing the operator buttons or performing operations with insufficient information does not give out app the wrong output. **This sums up the logic folder of our src directory.**

The display.js file in components folder imports propTypes and display.css. The display.css is imported for the styling of the display that will be rendered by this file. The propTypes makes this component reusable as it saves the properties of this component and thus any other developer using this compononet will not have trouble figuring out its properties. The Display class contains a static propTypes object with its value property set to propTypes.string. The render method of this component has a div element that renders the value property of our static object. The div element’s class name is set to component-display so it can be easily referenced and styled in the CSS file. The display.css file uses the dot notation to reference to the component-display div element and then some styling is added to it. This Display class is exported out by ‘export default class’ so that it may be imported anywhere else it may be needed. All our classes in our components will be exported so that they may be imported anywhere we need them.

After display.js and display.css we have buttonPanel.js and buttonPanel.css files. Their names tell us that this compononent will handle the rendering for our calculator’s button panel. The buttonPanel.js imports Button and just like display.js it imports PropTypes as well. Then buttonPanel.css is imported to link the styling. Then we have a buttonPanel class that has a static object and a handleClick arrow function. The static object has a clickHandler property that has PropTypes.func value. After the static object we have our handleClick function that takes in buttonName as an argument and then passes that buttonName to the clickHandler property of our static object. Then we have the render method for this component. In the render method we have our div tag with a className so it can be referenced easily in the .css file for styling. In the div element we have button elements for all our numbers from 0-9 and then some buttons for our calculator operations like addition, subtract, AC (all clear) etc. The colour of the operations has been set to orange to highlight them so they may be easily distinguished from other buttons. Each button element calls the handleClick function onClick as a reference.

Now we have the button.js and its corresponding button.css files. Just like the previous files button.js also imports PropTypes and the corresponding stylesheet. Then it has the Button class that has a static object. This static object has a name, orange, wide and clickHandler properties. The name property is set to PropTypes.string, orange and wide are set to PropTypes.bool and clickHandler is set to PropTypes.func. Then we have a clickHandle function that takes no arguments. It passes the name property of the static object to static object’s clickHandler. In the render method for this component we have a constant array named className that has a “component-button” string, a conditional statement that checks the orange property (it was initially set to a Boolean value so it will either be true or false) of static object. If the orange property returns true then index 1 has value of “orange” else it has an empty string “”, another conditional statement that checks if the wide property is true or false. If the wide property returns true then index 2 of className array has value of “wide” string else it holds an empty string “”. Lastly we have this render method’s return statement. In the return statement the div’s className is set to the constant className array and the Array.join method is used to stringify the array while the trim method is used to remove whitespaces. In the div tag we have a button element that has its onClick set to the handleClick function. The name of the button is set to the name property of our static object.

Next we have the App.test.js file that is used for testing our application when it’s executed.

After testing we have our App.js and App.css files. In the App.js file we have imported the Display class from display.js file along with the ButtonPanel class from buttonPanel.js file and the stylesheet for App.js i.e. App.css. Furthermore we have also imported the calculate function from the calculate.js file in src directory. Then we have the App class and a state object in it. The state object has total, next and operation properties all set to null value. Then we have a handleClick function that takes a button name as an argument. The handleClick function calls the setState method and calls the calculate function and passes the object’s state and the button name (that the handleClick function initially took as an argument) to the calculate function. So, the value returned by the calculate function is passed to the setState method. This setState method tells React that the component has had some changes and that React needs to reflect those changes on the DOM. Finally we have our render method for this file. We have a div tag in the render method and that div tag has a className for easy styling. Then we have a display tag. This display tag sets the value of the static object in the Display class that was imported from the Display.js file. Then this Display’s value is rendered on the screen in our result area. It is by default set to 0. **This sums up the components folder of our src directory.**

Finally we have two files in our src folder. Index.js and index.css. Index.js is the file that imports our App class from App.js and then uses the ReactDOM.render method to render our app. Index.css is its corresponding styling file.