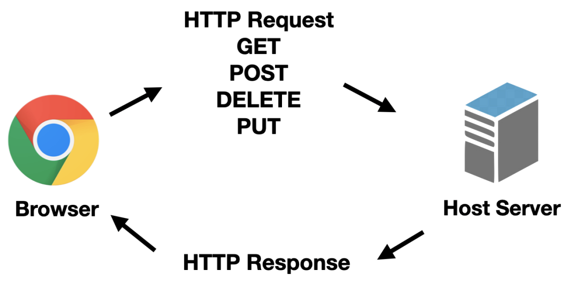
README.md

Lecture Notes

Table of contents

How the Web Works

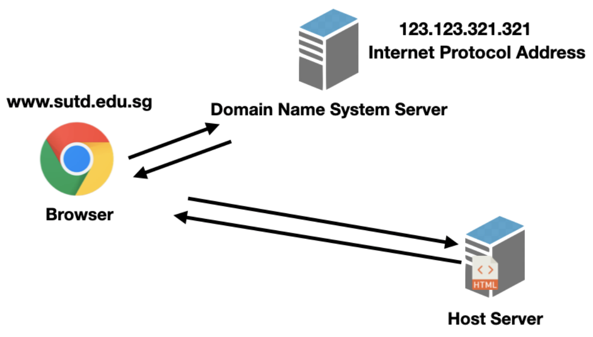
Overview of how data is transfered



The web works by sending out Hypertext Transfer Protocol (HTTP) Requests and getting Responses in return.

You can view this as your web browser sending out an envelope as a HTTP request to a server asking for data in return. The server then sends out another envelope with the requested data as a HTTP response.

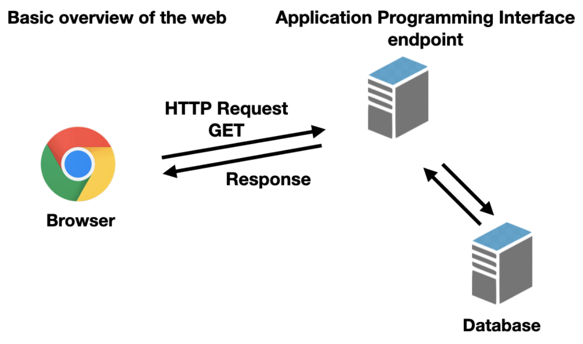
Resolving Domain Names



Whenever you type in a website link, it is also called a Domain Name. A domain name has no information on which server it should get the webpage from. Therefore, we have a Domain Name System server that caches (temporary stores) which domain name links to which Internet Protocol (IP) Address. An IP Address acts like your house address but for servers.

So in summary, when you type in a domain name, your browser sends out a HTTP Request to a DNS Server to get the IP Address of the server that contains the webpage. With the correct IP Address, your browser then sends another HTTP Request to that server for the webpage.

Application Programming Interface (API) Endpoint


Basic overview of the web
Application Programming Interface endpoint
HTTP Request GET
Response
Browser
Database

After recieving the files needed to load the webpage, the web application frequently also communicates with a database to load dynamic data. In order to do this, the webpage send out HTTP Requests to an Application Programming Interface (API) Endpoint. An API Endpoint acts like a vending machine where you exchange a HTTP Request for data as a HTTP Response.

The server whose API endpoint is on, will recieve a HTTP Request and depending on the requests, the server can access a database to find data then return a HTTP response with the requested data back to the server.

For example, we load the SUTD webpage and it wants to show an updated number of students currently enrolled. After loading the webpage, in order to keep the number up to date, the web application can send out a HTTP Request to SUTD's server. Once SUTD's server get a HTTP Request, it then communicates with a database to get the lastest number of enrolled students. The server then wraps the data in a HTTP Response and sends it back to the web browser.

Javascript vs Python refresher

Defining Variables

Defining a variable is similiar to how variable works in mathematics.
For illustration purposes, we will be assigning the string value "SUTD" to a variable named "school"

```
# PYTHON
school = "SUTD" #creates a variable named school and assign a string "SUTD" to it
```

```
// JAVASCRIPT
const school = "SUTD" //creates a constant variable named school
let school = "SUTD" //creates a variable named school and assign a string "SUTD" to it
```

Notice that javascript has 2 ways to define a variable.
`const` creates a constant variable which means the value you assign to it will not change.
`let` creates a variable in which its value will be changing down the line.

Lists / Arrays

Defining a list or array in python vs javascript
For illustration purposes, we will be creating an array named thisIsAnArray and assign an empty array to it.

```
# PYTHON
thisIsAnArray = [] #creates a variable named thisIsAnArray and assigns an empty array to it


// JAVASCRIPT
const thisIsAnArray = [] //creates a constant variable named thisIsAnArray and assigns an empty array to it
let thisIsAnArray = [] //creates a variable named thisIsAnArray and assigns an empty array to it
```

Object / Dictionary

Defining a list or array in python vs javascript
For illustration purposes, we will be creating an object named thisIsAnObject and assign a key value pair.

```
# PYTHON
thisIsAnArray = {
  key: "value"
} #creates a variable named thisIsAnArray and assigns an object to it.


// JAVASCRIPT
const thisIsAnArray = {
  key: "value"
} //creates a constant variable named thisIsAnArray and assigns an object to it.
let thisIsAnArray = {
  key: "value"
} //creates a variable named thisIsAnArray and assigns an object to it.
```

Functions

Defining a function in python vs javascript
For illustration purposes, we will be creating an function named thisIsAFunction and make the function take in 1 paramater named paramOne return hello world.

```
# PYTHON
def thisIsAFunction ( paramOne ):
  return( "hello world" )


// JAVASCRIPT : There are 2 types of functions. Function expressions and function declarations.

/*
----Function Expression----
Function is loaded when the line is reached
*/

// The below function is called a Arrow Function (I guess because of the => )
const thisIsAFunction = ( paramOne ) => {
  return( "hello world" )
}
const thisIsAFunction = ( paramOne ) => "hello world"
// Notice how this version of the Arrow Function, doesn't have the curly braces {}
// Without the curly braces, the function automatically return whatever after that. In this case, the function automat
const thisIsAFunction = function( paramOne ){
  return("hello world)
}
```

```
/*
----Function Declaration----
Function Declaration are hoisted to the top of the code.
Meaning the funtion is loaded before anything else
*/

function thisIsAFunction( paramOne ){
  return( "hello world" )
}



// To demostrate the differences
// Take note that console.log acts just like python print function.
console.log( helloWorldFunction() )  // This will have an error
const helloWorldFunction = () => "hello world"

console.log( helloWorldFunction() )  // This will work
function helloWorldFunction( ){
  return( "hello world" )
}
```

Loops

Creating a loop to reiterate over an array in python vs javascript
For illustration purposes, we will reiterate over an array named oneToTen which contains digits 1 to 10.

```
oneToTen = [ 1,2,3,4,5,6,7,8,9,10 ]

for eachElement in oneToTen:
  print(eachElement)
# prints 1 2 3 4 ...
```

```
const oneToTen = [ 1,2,3,4,5,6,7,8,9,10 ]

oneToTen.forEach( eachElement => console.log(eachElement) )
// prints 1 2 3 4 ...

// Another method to reiterate over an array is
oneToTen.map( (eachElement, eachIndex)=> console.log(eachElement, eachIndex) )

/*
  Take note that .map iterates over the array and returns a new array with the same length.
  .forEach only iterates over the array and does not return anything.
*/
//For example
const newArray = oneToTen.map(x => x * x)
console.log(newArray) // this will print a new array named newArray in which each digit is the square of the original
```

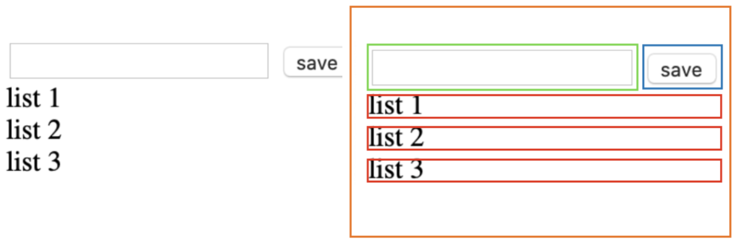
Hyper Text Markup Language & Cascading Style Sheets

Hyper Text Markup Language (HTML), thelanguage of the document that the browser reads for content to display.
Cascading Style Sheets (CSS), the language used by the browser to decide the visual look of the content.

HTML and CSS work hand in hand to display and beautify a webpage respectively.

HTML

HTML can be thought of as containers. HTML start and end off with tags enclosed in arrow brackets.



For example, we want to replicate the image above, we would then split each individual component into a container.
In HTML code, it will look like

```
    <div>
      <input/>
      <button> Save <button/>
      <div> list 1 </div>
      <div> list 2 </div>
      <div> list 3 </div>
    </div>
```

Take note that each html tag has an opening and a closing tag. Eg

```
  <div></div>
```

A self closing tag can also be used. Eg

```
  <input/>
```

Each tag has its own attributes too. Eg

```
  <button onClick="whenButtonClickRunThisFunction()" />.
```

This button has an event handler function name "whenButtonClickRunThisFunction" that runs everytime the button is clicked.

CSS

CSS can be thought of as a bunch of properties you want each HTML container to have.

```
    <div class="redBoxClass" >
      This is a Red Box
    </div>
```

First we assign a class to the HTML tag.

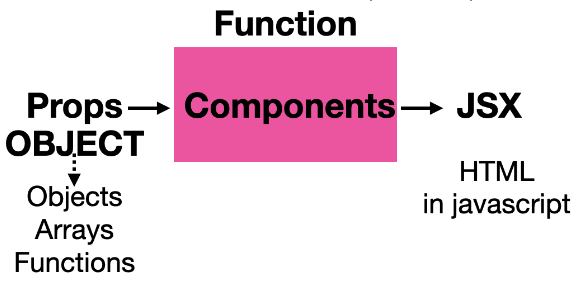
```
  .redBoxClass{
    background-color: red
  }
```

CSS then assigns a property to the following class.

React

Components

The react library are made out of components. Just like how HTML is made up of multiple containters, react uses the same container concept but calls it components insetad.



A React Component is a Javascript Function that takes in an object called `props` and return `JSX`.
You can view props as properties being pass into a component, just like how you pass parameters into a function.
You can view JSX as HTML but used as an extension of Javascript.
An example of a component

```
// We first create a function that takes in props as a parameter and return JSX.
const thisIsAComponent = ( props ) => {
  return(
    <div>
      This is like HTML but it is actually JSX.
    </div>
  )
}
// If we want to write javascript when we are in JSX, we need to write it in curly braces
const thisIsAComponent = ( props ) => {
  return(
    <div>
      This is like HTML but it is actually JSX.
      {
        console.log(" If you want to add Javascript code in JSX, you need to wrap it in curly braces {} ")
      }
    </div>
  )
}
```

State

A state is like a temporary storage on your browser. It can be viewed like a temporary database.

React uses a function called `useState` to intialize a state. `useState` is one of the many functions called react hooks.
useState function takes in the initial state of the state. It returns an array containing the current state and a function to update the state. For example, we want inialize a state that initially have a value of `"Hello World"`.

```
const [ currentValueOfState , functionToUpdateState ] = useState( "Hello World" )
// As shown, passing "Hello World" to useState assigns "Hello World" to currentValueOfState.
// In which to update currentValueOfState, we need to use functionToUpdateState.
```

To illustrate what is happening

```
const [ currentValueOfState , functionToUpdateState ] = useState( "Hello World" )

console.log( currentValueOfState ) // this will print "Hello World"

functionToUpdateState("SUTD")

console.log( currentValueOfState ) // this will print "SUTD"
```

CSS Flexbox

CSS Flexbox are useful bunch of CSS properties that allow webpages to be responsive. Meaning, if the browser changes width or height, the content will adapt accordingly. Flexbox works with the concepts of rows and columns.
Link to css flexbox documentation. It contains a detailed writeup on flexbox.

Display Flex

To enable flexbox properties, we first need to enable flex container.
To do this, we need the to use `display: flex`.

```
<div class="flex" >This will become a flex container</div>
```

```
.flex{
  display: flex
}
```

Flex-direction

Flex-direction determines if your flex container should arrange the content inside as a row or column.
For example

```
<div class="flex" >
  <div> content 1 </div>
  <div> content 2 </div>
  <div> content 3 </div>
</div>
```

```
.flex{
  display: flex;
  flex-direction: row;
}
```

Justify Content

Justify content determines how content should be spaced out in the main axis.
When I use `flex-direction: row` , my main axis is the row axis or the X axis.
When I use `flex-direction: column` , my main axis is the column axis or the Y axis.
There are a few properties that can be used like `space-around` , `flex-start`. Do check out the documentation, it is very well written.
Link flexbox Documentation
For example

```
<div class="flex" >
  <div> content 1 </div>
  <div> content 2 </div>
  <div> content 3 </div>
</div>
```

```
.flex{
  display: flex;
  flex-direction: row;
  justify-content: space-around;
}
```

Align Content

Align content acts similiar to justify-content, just that the axis is the opposite.
When I use `flex-direction: row` , my main axis is the row axis or the X axis. Therefore, `align-content` adjusts the column axis or Y axis.
When I use `flex-direction: column` , my main axis is the column axis or the Y axis. Therefore, `align-content` adjusts the row axis or X axis.
There are a few properties that can be used like `space-around` , `flex-start`. Do check out the documentation, it is very well written.
Link flexbox Documentation
For example

```
<div class="flex" >
  <div> content 1 </div>
  <div> content 2 </div>
  <div> content 3 </div>
</div>
```

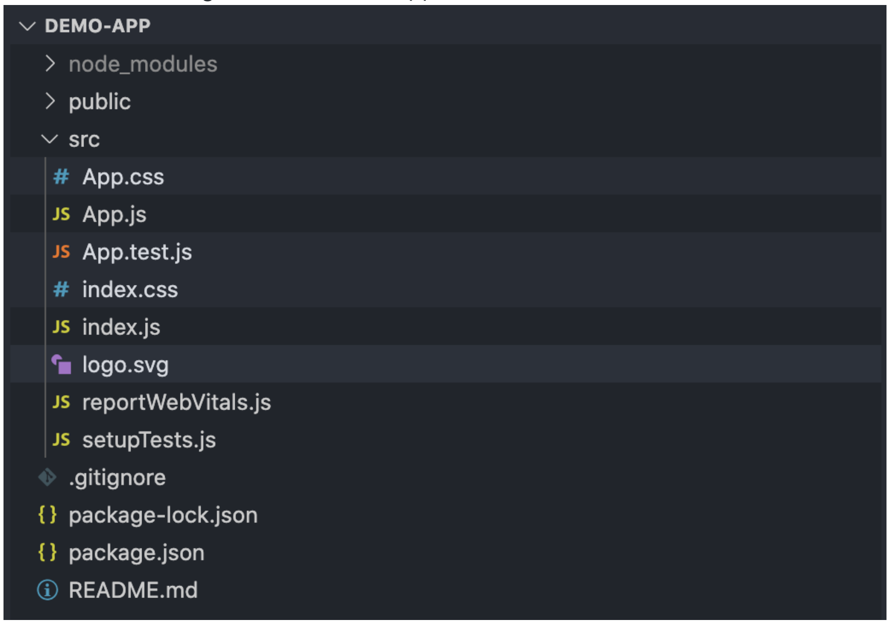
```
.flex{
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-content: center;
}
```

Demo

Cleaning up create react app

First we need to use the create-react-app package to initialize a templated app for us. To do this, open your terminal (mac) or command prompt (windows).
Change directory to the folder you want to place the application folder in. In this case, let's use the desktop folder.
To change directory to the desktop folder, run `cd ./desktop`.
Let's install the create react app template. Run the following commands `npx create-react-app todo-app-demo`
This will create a folder named todo-app-demo in your desktop folder.

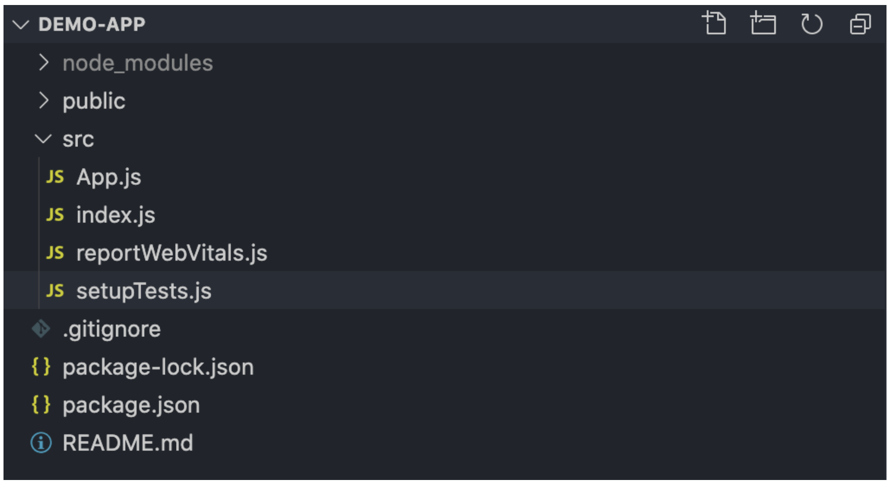
Let's start by cleaning up some files to get started from scratch.
Delete the following files in the todo-app-demo.



```
DEMO-APP
  node_modules
  public
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

Delete the following files:

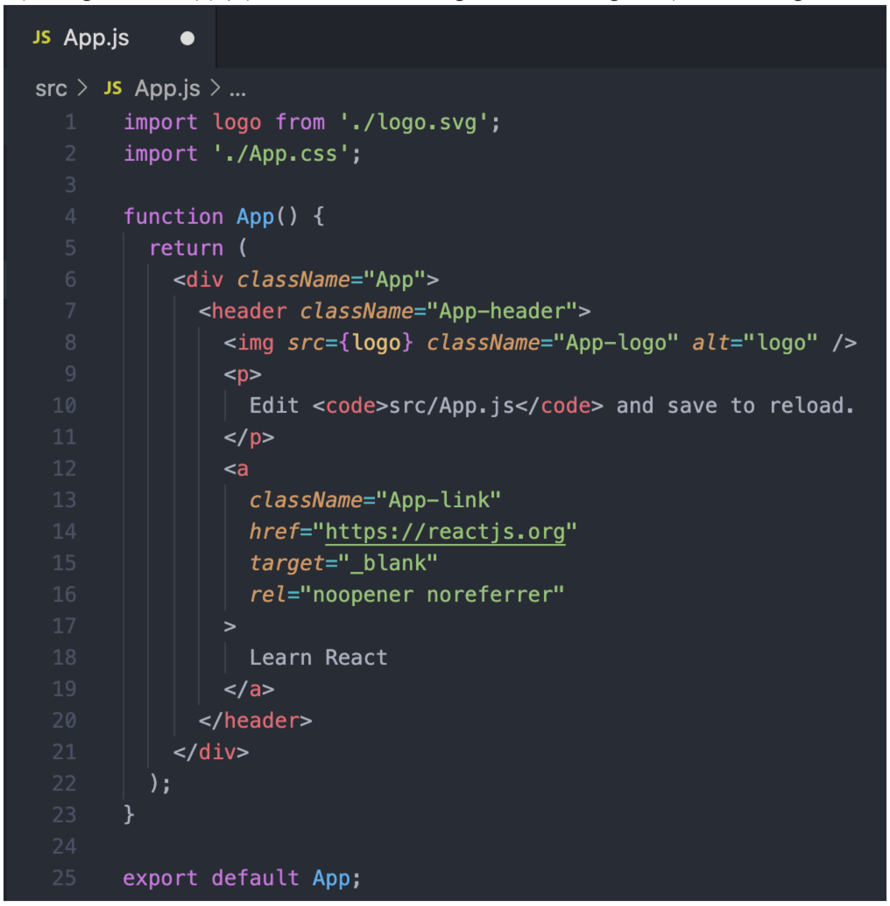
- App.css
- App.test.js
- index.css
- logo.svg



```
DEMO-APP
  > node_modules
  > public
  v src
      App.js
      index.js
      reportWebVitals.js
      setupTests.js
    .gitignore
    package-lock.json
    package.json
    README.md
```

You should be left with these files.

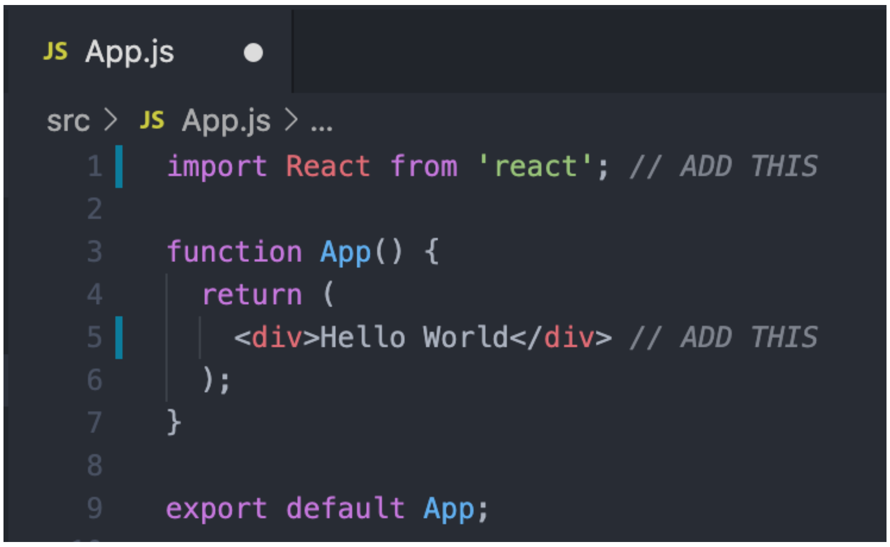
Next, we shall clean up the files.
Opening the file App.js, delete the following lines to change as per the image.



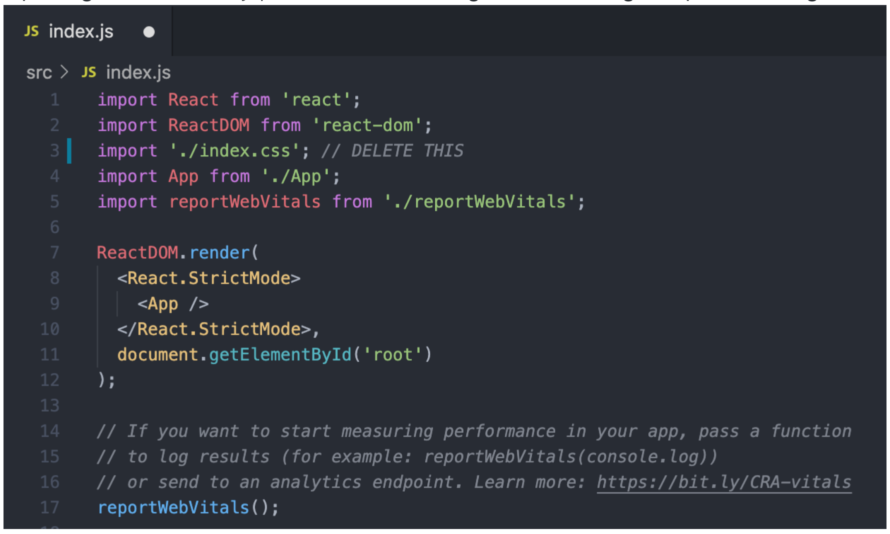
App.js ●

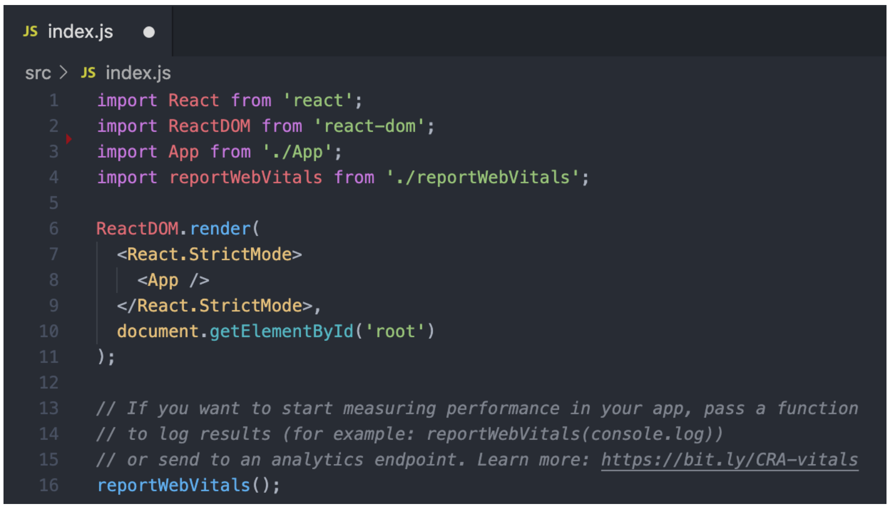
src > App.js > ...

```
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;
```

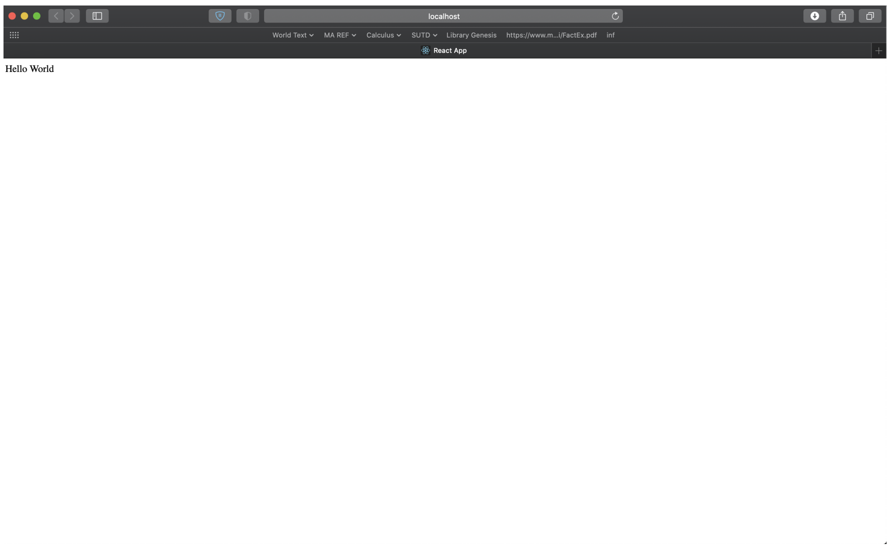

```
import React from 'react'; // ADD THIS

function App() {
  return (
    <div>Hello World</div> // ADD THIS
  );
}

export default App;
```

Opening the file index.js, delete the following lines to change as per the image.


```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css'; // DELETE THIS
import App from './App';
import reportWebVitals from './reportWebVitals';

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

// 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();
```


```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import reportWebVitals from './reportWebVitals';

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

// 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();
```

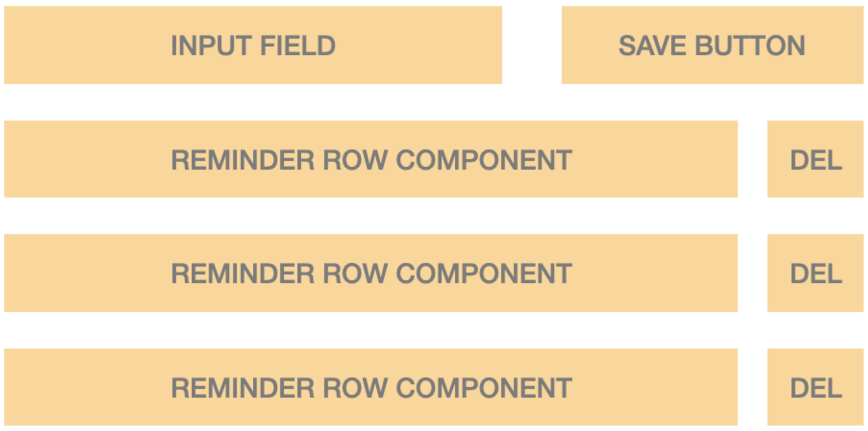
Lets run the application. Go back to your terminal and change directory into the todo-app-demo by running `cd ./todo-app-demo`
To run the application, run `npm run start`.
Your browser should open and look like this.



Hello World

Breaking down structure of html

Let's first see how we can picture how the app will look like then break it down into html/jsx code.



The image shows how the app will look like. We have 3 main big containers.
1 container to contain the entire app.
1 inner container that should behave like a row for the input and save componenets.
1 inner container that should behave like a column for the rows of reminder row components.

In HTML code

```
<div>
  <div> For the input and save button </div>
  <div> For the rows of reminder components </div>
</div>
```

In React

```
const myComponent = (props) => {
  return(
    <div>
      <div> For the input and save button </div>
      <div> For the rows of reminder components </div>
    </div>
  )
}
```

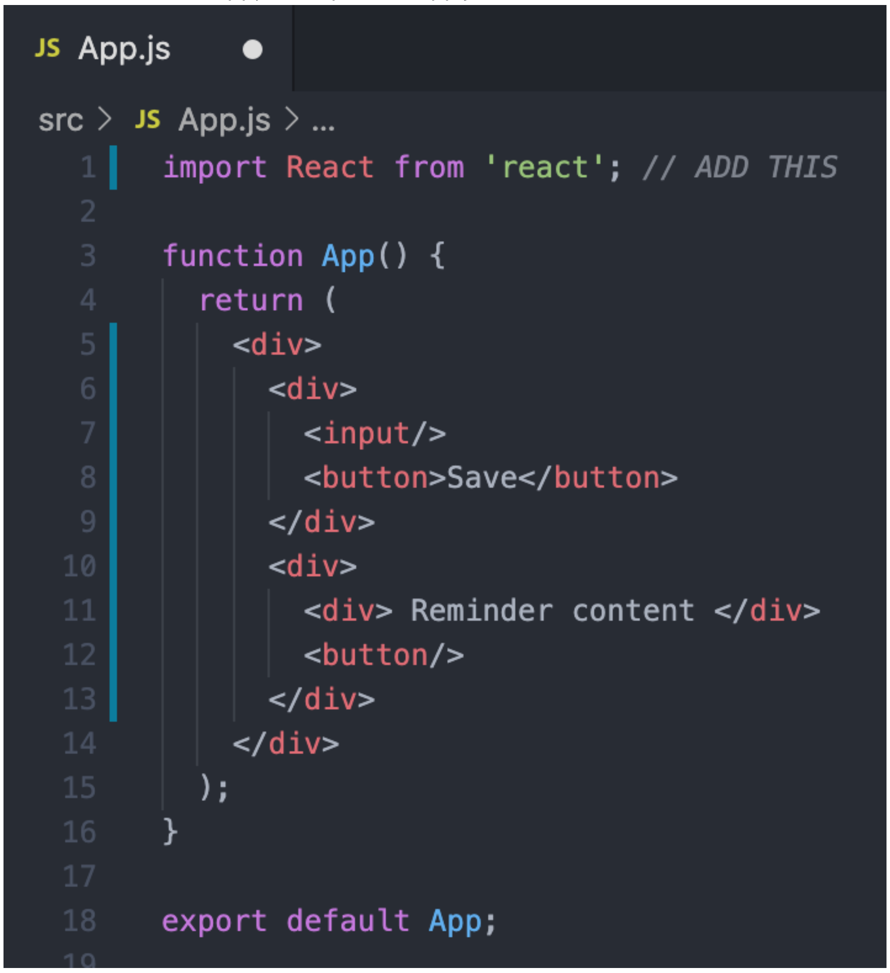
We then can further break down into the individual input and save button. Followed by the reminder and delete button.
In HTML code

```
<div>
  <div>
    <input/>
    <button>Save</button>
  </div>
  <div>
    <div> Reminder content </div>
    <button/>
  </div>
</div>
```

In React

```
const myComponent = (props) => {
  return(
    <div>
      <div>
        <input/>
        <button>Save</button>
      </div>
      <div>
        <div> Reminder content </div>
        <button/>
      </div>
    </div>
  )
}
```

To add this into our app, let's open the app.js file and add in the react code.



```
import React from 'react'; // ADD THIS

function App() {
  return (
    <div>
      <div>
        <input/>
        <button>Save</button>
      </div>
      <div>
        <div> Reminder content </div>
        <button/>
      </div>
    </div>
  );
}

export default App;
```

Setting up State

Let's breakdown how we should set up the state.
We need to store 2 things.
We need store the characters in which our user will type and then store those characters into a list or array.
Then how should we store the text and the list of todo reminders? For illustration purposes,

```
// For the text that the user types in we can just simply store it as a string.
// Something like,
let inputValue = "this variable will contain whatever the user type"

// But for the todoList, should we just store a bunch of strings in an array.
// Something like,
let todoList = [ "todo1" , "todo2", "todo3" ]
/*
  Then when we delete it, we can just compare if the text of the todo reminder is the same as the one the user selects

  This will work but will introduce an issue.
  What if 2 todo reminders have the same text?
  Which one to delete?
*/

// Therefore we need a unique identifier for each todo reminder.
// Something like,
let todolist = [
  {
    id: "SOME KIND OF UNIQUE INDENTIFIER",
    content: "THE TODO REMINDER TEXT"
  }
]
```

Therefore, we should intialise our state like

```
const [inputValue , setInputValue] = useState("") //initilize inputValue as an empty string
const [todoList , setTodoList] = useState([]) //initilize todoList as an empty array
```

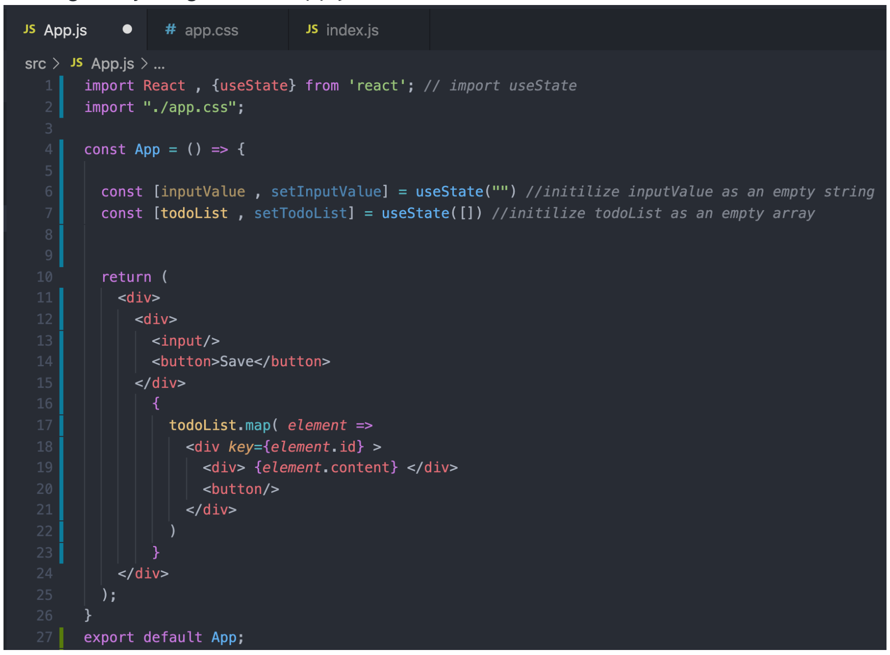
Take note that `useState` takes in the initial state you want your state to be.
It then returns an array containing 2 elements.
1. The current state of the
2. A function to update the state

So in the case of `const [inputValue , setInputValue] = useState("")` ,
We are telling React, to initialize `inputValue` as an `""` in which `useState` will return 2 values, the variable name of the state and a function `setInputValue` that updates the `inputValue`.
So whenever, we type something, it gets stored into inputValue and then when we save, it gets stored into todoList.

Now with state set up, we do not want to keep manually adding divs as the todoList array gets updated with more elements.
Therefore, we can use a loop to assit us.

```
const myComponent = (props) => {
  return(
    <div>
      <div>
        <input/>
        <button>Save</button>
      </div>
      {

        /*
          1.  Take note that we need to use .map and NOT .forEach
              This is because forEach DOES NOT return an array as it loops over the array.

          2.  Take note that when mapping over an array and returning JSX, we need a key property which
              needs to be a unique value. This key is used by React to keep track of component
        */

        todoList.map( element =>
          <div key={element.id}>
            <div> {element.content} </div>
            <button/>
          </div>
        )
      }
    </div>
  )
}
```

Putting everything into our app.js code as such



```
import React , {useState} from 'react'; // import useState
import "./app.css";

const App = () => {

  const [inputValue , setInputValue] = useState("") //initilize inputValue as an empty string
  const [todoList , setTodoList] = useState([]) //initilize todoList as an empty array


  return (
    <div>
      <div>
        <input/>
        <button>Save</button>
      </div>
        {
          todoList.map( element =>
            <div key={element.id} >
              <div> {element.content} </div>
              <button/>
            </div>
          )
        }
    </div>
  );
}
export default App;
```

Event Handlers

Now that we have the state and base structure of our React App. We need javascript functions to interact with our state. These functions are called event handlers.
A event handler is basically a function that is invoked/run when the user interacts with a webpage. Be it a click or keystrokes or etc...

Let's breakdown what kind of functions we need

Firstly, we need 1 function to take the characters our user type and update the `inputValue` value.

Secondly, Whenever our user click save, we need 1 functon to then save the `inputValue` characters into our `todoList` array.
Don't forget we also need to empty out `inputValue` characters so that the user can start typing on a clean input box again.
Another thing to take note is that when storing the texts, we need a unique identifier so that we know which task to delete even if the 2 tasks have the same text content. For the purposes of this demo, we will use javascript's build in random function `math.random.toString().replace(“0.”,””)` . However do not use this in production, please use a proper uuid library like uuid.

So our `todoList` state will store text like this

```
todoList = [
  {
    id: "RANDOM STRING TO INDENTIFY THE TASK",
    content: "WHATEVER THE USER TYPED AND SAVED"
  }
]
```

Next, we need a function to then delete the correct todo task and then update `todoList` with the updated todo list.
Making use of the unique id, the delete function will take the id as a parameter then compare it with all the ids currently in the todoList.
If they are the same, delete it and update todoList.

Lets code our the functions

```
const displayWhatIType = ( e ) => {
  // Notice how this takes in a e parameter.
  // It is actually the event object which your browser automatically passes into the function.
  setInputValue( e.target.value ) // e.target.value is how you access what the user has typed.
}

const saveWhatIType = () => {
  // Create an object that represents what the new todo task entry will look like
  let newContent = {
    id: math.random().toString().replace("0.",""),
    content: inputValue.trim()
  }

  /*
  Make a copy of the current value of todoList. This is because we do not append/push directly into the current valu
  */
  let copy = [...todoList]

  // Add the new content into the copy
  copy.push( newContent )

  // update todoList
  setTodoList( copy )
  // update inputValue
  setInputValue( "" )

  // You can combine all the code into a more compact version like this
  setTodoList( [...todoList, {
        id: math.random.toString().replace("0.",""),
        content: inputValue.trim()
      }
    ] )
  setInputValue( "" )
}

const deleteThisTodo = (id) => {
  setTodoList(
    todoList.filter( item => item.id != id )
    /*
        filter will iterate over the array and return a new array that passes the condition, item.id != id
        basically, filter will compare each item's id with the id of the todo reminder we want to delete.
        As long as the 2 ids are not the same (!=) , then filter will take the element and put it into a new array.
    */
  )
}
```

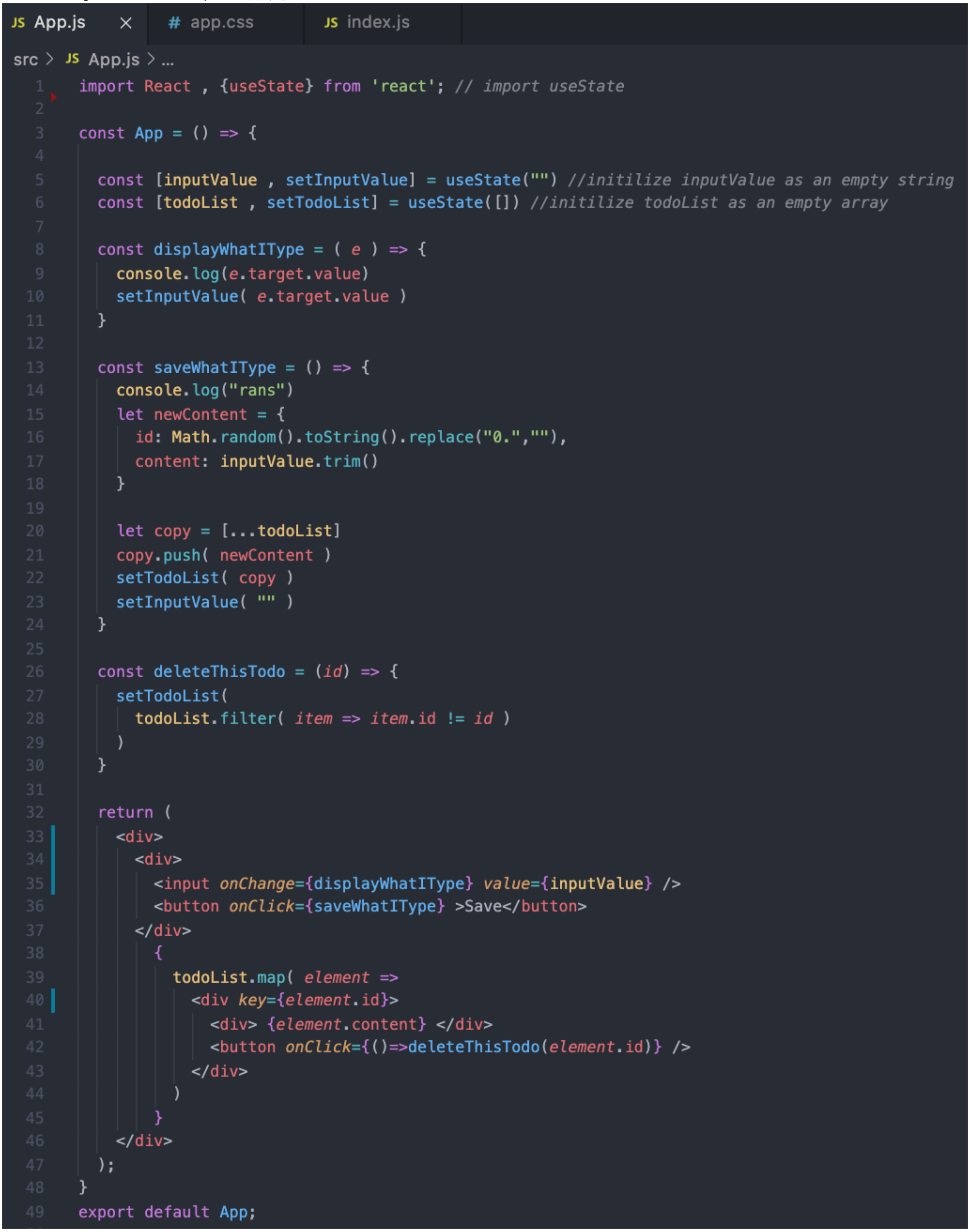
Going back to our react app, we need to add these event handlers into our app.js file. We also need to add the onClick and onChange attributes to to the buttons and input components.
We also need to `import React, { useState } from 'react';`

What's the difference between `import React from 'react'` and `import {useState} from 'react'`

When the import has `{}` , it means that whatever is inside the curly braces was not exported as default and therefore needs to be named. Therefore it is called a named import.
non default export looks like `export const helloWorld = "hello world" `

Whereas without `{}` , it means that the export is default and therefore doesn't need to be named.
Default export looks like `export default "hello world" `

After adding the function to your app.js, it should look like



```
import React , {useState} from 'react'; // import useState

const App = () => {

  const [inputValue , setInputValue] = useState("") //initilize inputValue as an empty string
  const [todoList , setTodoList] = useState([]) //initilize todoList as an empty array

  const displayWhatIType = ( e ) => {
    console.log(e.target.value)
    setInputValue( e.target.value )
  }

  const saveWhatIType = () => {
    console.log("rans")
    let newContent = {
      id: Math.random().toString().replace("0.",""),
      content: inputValue.trim()
    }

    let copy = [...todoList]
    copy.push( newContent )
    setTodoList( copy )
    setInputValue( "" )
  }

  const deleteThisTodo = (id) => {
    setTodoList(
      todoList.filter( item => item.id != id )
    )
  }

  return (
    <div>
      <div>
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
        {
          todoList.map( element =>
            <div key={element.id}>
              <div> {element.content} </div>
              <button onClick={()=>deleteThisTodo(element.id)} />
            </div>
          )
        }
    </div>
  );
}
export default App;
```

I would like to bring the focus now to the event handlers on the JSX code

```
<div>
  <div>
```

```
    <!--When input is changing (user typing), run displayWhatIType function -->
    <input onChange={displayWhatIType} value={inputValue} />
    <!-- Input then displays whatever is in the value attribute (in this case,inputValue) -->

    <button onClick={saveWhatIType} >Save</button>
    <!-- On click on save button, run saveWhatIType function -->
  </div>
  <div>
    {
      todoList.map( element =>
        <div key={element.id} >
          <div> {element.content} </div>
          <button onClick={()=>deleteThisTodo(element.id)} />
          <!--
            On click on delete button, run deleteThisTodo function.
            However, remember we need to pass the unique id of the todo task.
            To do this we need to pass the parameter element.id.

            But we can't type it out as onClick={deleteThisTodo(element.id)}

            This will cause the deleteThisTodo function to be invoked/run when the page loads.

            So to avoid that, we use an arrow function to create a new function that when it runs, it will then run th
          -->
        </div>
      )
    }
</div>
    </div>
```

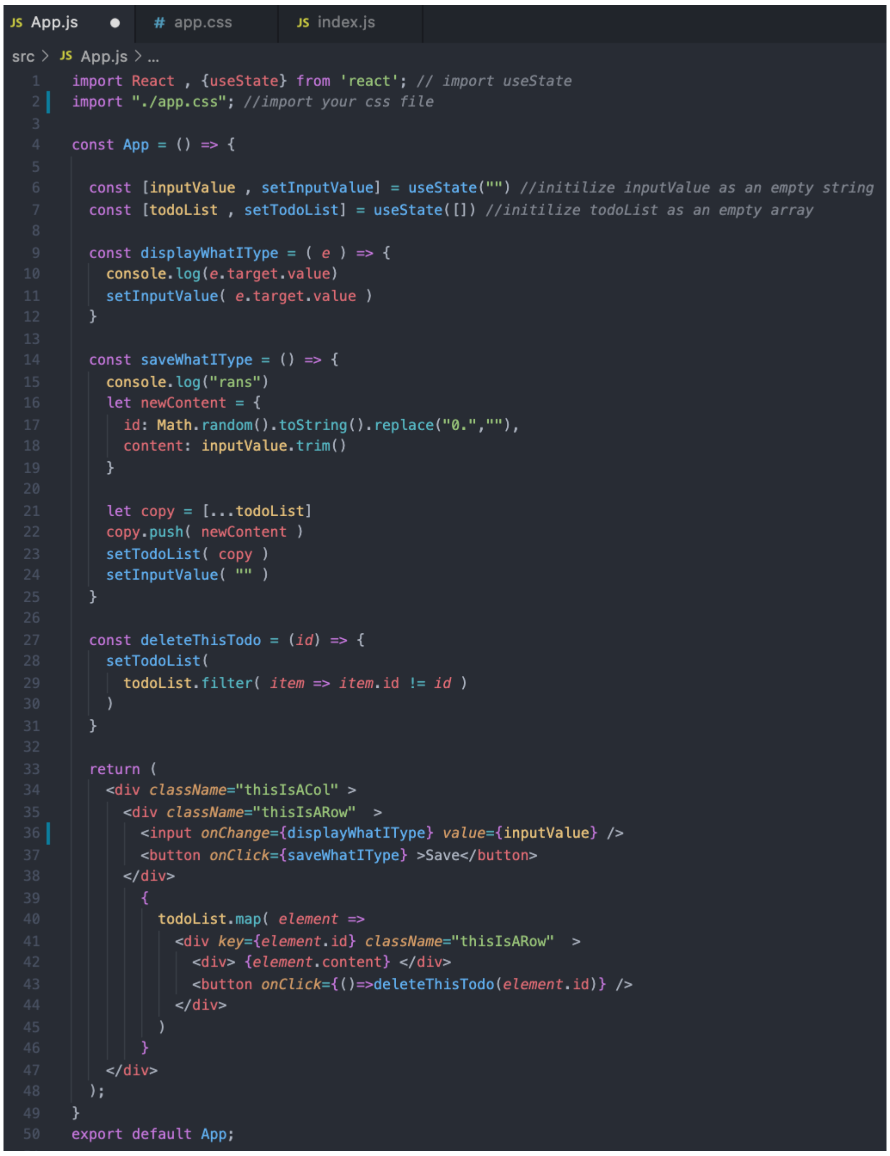
Setting up CSS

We first need give class names to our JSX elements.
Remember class names is like labelling the HTML tag.
CSS will then use the class names to identify the HTML block to assign properties to.

We will split up the containers that should act as a row or column. This means we can code 1 css class with that particular property and apply it anywhere else. This is a design prinipal you should try to apply as often as possible.

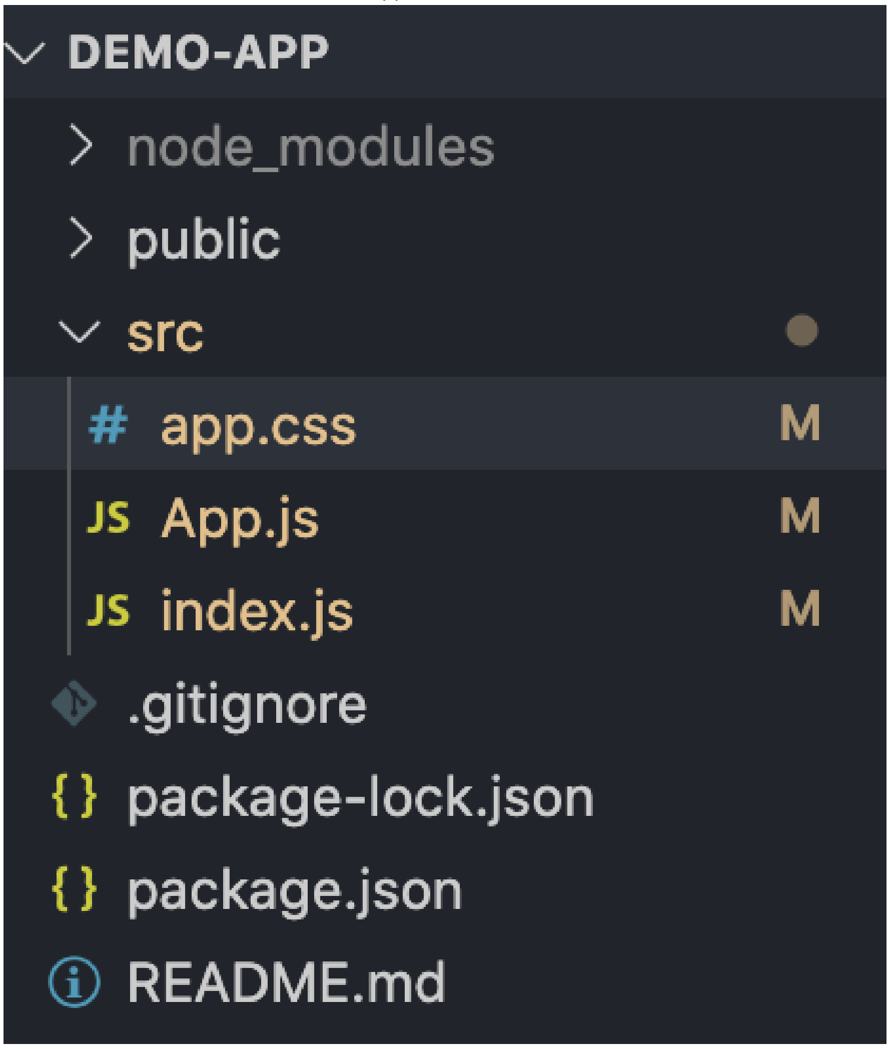
Remember to also import your css file as shown below.

```
  return (
    <div className="thisIsACol">
      <div className="thisIsARow" >
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
      {
        todoList.map( (element) =>
          <div key={element.id} className="thisIsARow">
            <div> {element.content} </div>
            <button onClick={()=>deleteThisTodo(element.id)} />
          </div>
        )
      }
    </div>
  );
```



```
import React , {useState} from 'react'; // import useState
import "./app.css"; //import your css file

const App = () => {

  const [inputValue , setInputValue] = useState("") //initilize inputValue as an empty string
  const [todoList , setTodoList] = useState([]) //initilize todoList as an empty array

  const displayWhatIType = ( e ) => {
    console.log(e.target.value)
    setInputValue( e.target.value )
  }

  const saveWhatIType = () => {
    console.log("rans")
    let newContent = {
      id: Math.random().toString().replace("0.",""),
      content: inputValue.trim()
    }

    let copy = [...todoList]
    copy.push( newContent )
    setTodoList( copy )
    setInputValue( "" )
  }

  const deleteThisTodo = (id) => {
    setTodoList(
      todoList.filter( item => item.id != id )
    )
  }

  return (
    <div className="thisIsACol" >
      <div className="thisIsARow"  >
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
        {
          todoList.map( element =>
            <div key={element.id} className="thisIsARow"  >
              <div> {element.content} </div>
              <button onClick={()=>deleteThisTodo(element.id)} />
            </div>
          )
        }
    </div>
  );
}
export default App;
```

Let's create a new css file named `app.css`



Let's add the css properties like this

```
.thisIsARow{
    display: flex;  /* enable flex */
    flex-direction: row; /* act as a row */
    justify-content: space-around; /* in the main axis (x axis), evenly space my content */
    align-content: center; /* in the other axis (y axis), center my content */
}
.thisIsACol{
    display: flex; /* enable flex */
    flex-direction: row; /* act as a row */
    justify-content: space-around; /* in the main axis (y axis), evenly space my content */
    align-content: center; /* in the other axis (x axis), center my content */
}
```

We are done! We got the basic functionality!

Seperating JSX into Components

To make our code more reusable, we can split our JSX into components and this allows us to reuse these components anywhere else. This reduces repetitive code and makes the codebase modular and easier to maintain.

Our task now is to make our todo reminder row into a component.

```
// make this code
    <div className="thisIsACol">
      <div className="thisIsARow" >
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
      {
        todoList.map( (element) =>
          <div key={element.id} className="thisIsARow">
            <div> {element.content} </div>
            <button onClick={()=>deleteThisTodo(element.id)} />
          </div>
        )
      }
    </div>

// become this
<div className="thisIsACol">
      <div className="thisIsARow" >
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
      {
        todoList.map( (element) =>
          <TodoRowComponent/>  // THIS IS A COMPONENT
        )
      }
    </div>
```

But this means that we have to somehow transfer the functions and content into that component.
This is where props come into the picture. We will be passing the required content and functions through props.

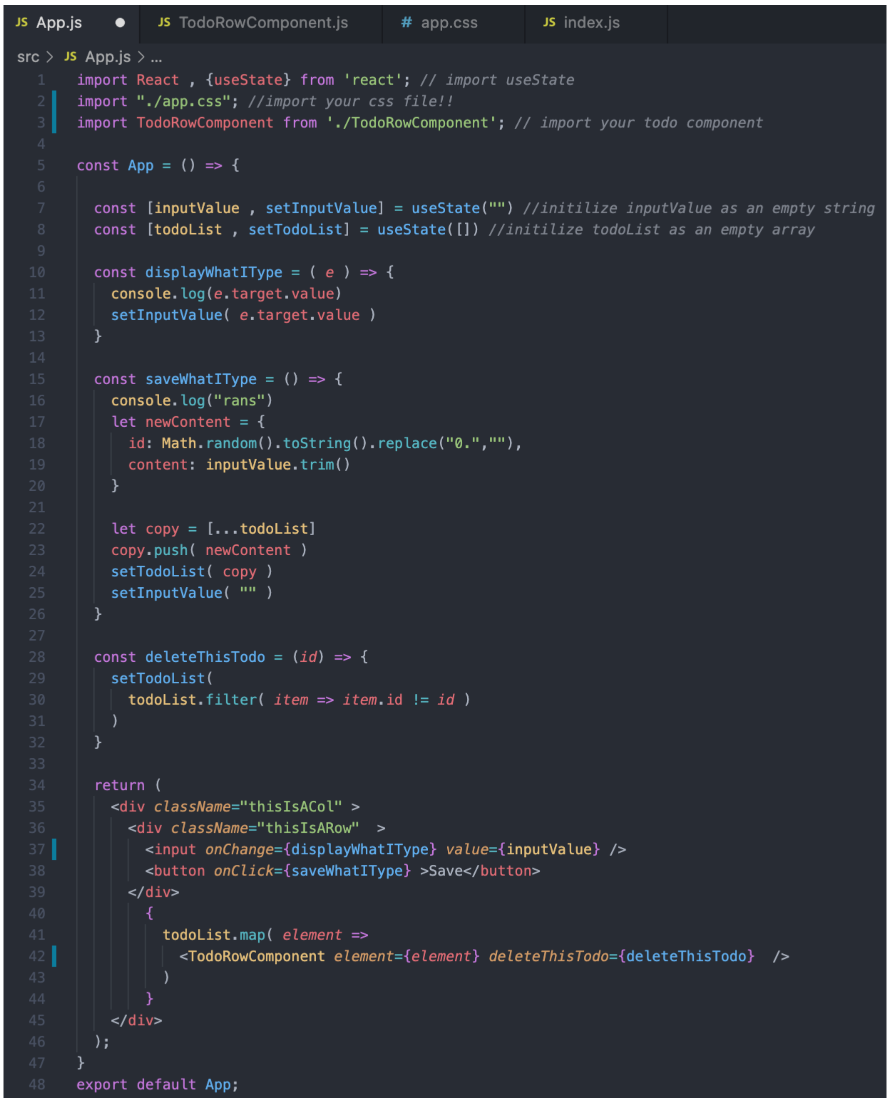
```
<div className="thisIsACol">
  <div className="thisIsARow" >
    <input onChange={displayWhatIType} value={inputValue} />
```

```
      <button onClick={saveWhatIType} >Save</button>
    </div>
    {
      todoList.map( (element) =>
        <TodoRowComponent element={element} deleteThisTodo={deleteThisTodo}  />
        // adding properties to the component will pass it as a prop into the array. Remember that a React componenet ta
      )
    }
  </div>
```

We then create a new file named TofoRowComponent.

```
import React from 'react' // remember to import react. All components need this as JSX as it is used to after JSX is t

const TodoRowComponent = (props) => {

    let { element , deleteThisTodo } = props  // this is called object destructuring, we use this to directly access c

    /*
      you can view props as an object

      props = {
        element: element data that was passed in ,
        deleteThisTodo: the function that was passed in
      }

    */

    return(
        <div key={element.id} className="thisIsARow">
            <div> {element.content} </div>
            <button onClick={()=>deleteThisTodo(element.id)} />
        </div>
    )
}

export default TodoRowComponent // remmeber to export the componenet to be used in the app.js file
```

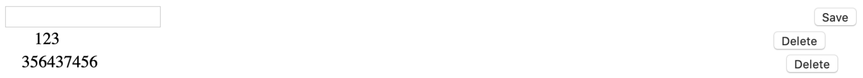
Your app.js file should look like this now



```jsx
import React , {useState} from 'react'; // import useState
import "./app.css"; //import your css file!!
import TodoRowComponent from './TodoRowComponent'; // import your todo component

const App = () => {

  const [inputValue , setInputValue] = useState("") //initilize inputValue as an empty string
  const [todoList , setTodoList] = useState([]) //initilize todoList as an empty array

  const displayWhatIType = ( e ) => {
    console.log(e.target.value)
    setInputValue( e.target.value )
  }

  const saveWhatIType = () => {
    console.log("rans")
    let newContent = {
      id: Math.random().toString().replace("0.",""),
      content: inputValue.trim()
    }

    let copy = [...todoList]
    copy.push( newContent )
    setTodoList( copy )
    setInputValue( "" )
  }

  const deleteThisTodo = (id) => {
    setTodoList(
      todoList.filter( item => item.id != id )
    )
  }

  return (
    <div className="thisIsACol" >
      <div className="thisIsARow"  >
        <input onChange={displayWhatIType} value={inputValue} />
        <button onClick={saveWhatIType} >Save</button>
      </div>
        {
          todoList.map( element =>
            <TodoRowComponent element={element} deleteThisTodo={deleteThisTodo}  />
          )
        }
    </div>
  );
}
export default App;
```

What else can be done

Now that we got basic functionality done. What else can we improve? Take a look at this



If our todo reminder is not the same length, the delete button does not line up. What do you think is causing this? (Hint: it is a css flexbox problem)

Another thing, can we somehow make the input fill up more width? (Hint: It is also a CSS property, it is called width)