# React fundamentals

## What is react?

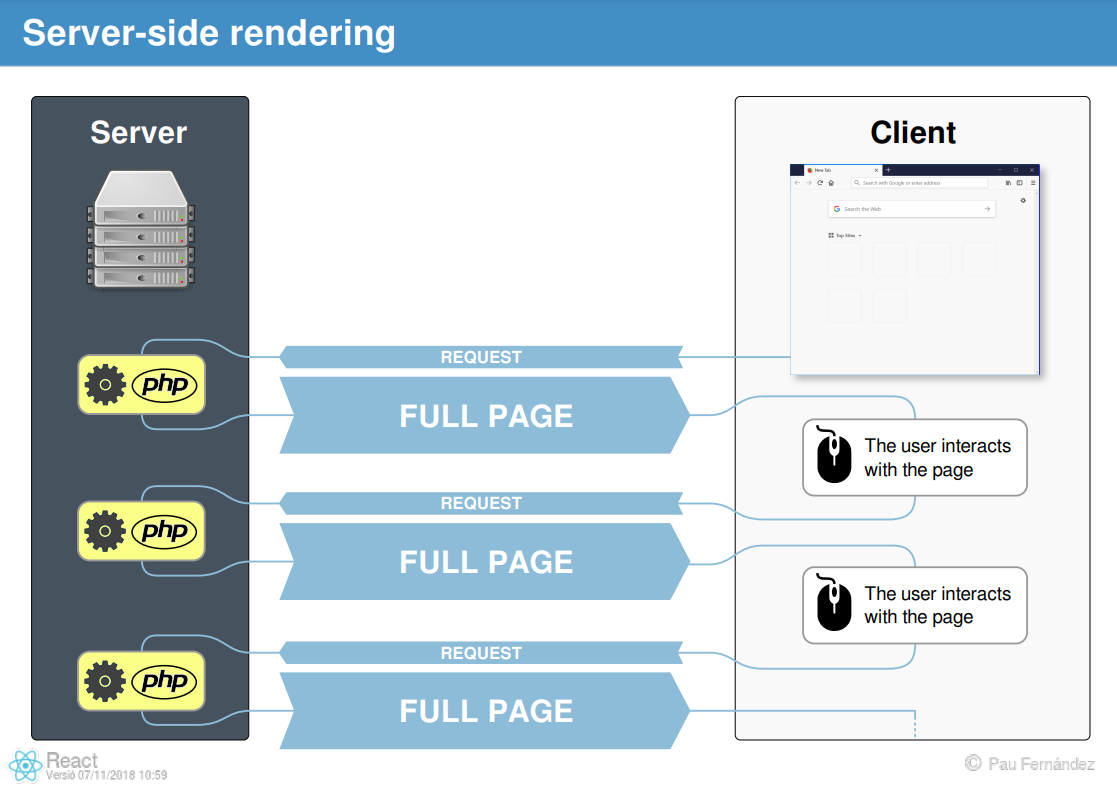
* Javascript library/framework for building user interfaces.
* It’s strictly front-end meaning it only runs in the client as a Single Page App.
* Often used combined with other frameworks to create full stack applications.
* Often referred as a framework because it has a solid ecosystem of packages that make it function as a full fledge framework, but still by itself is less complete than for example Angular (Provides a whole routing system).

## Main characteristics

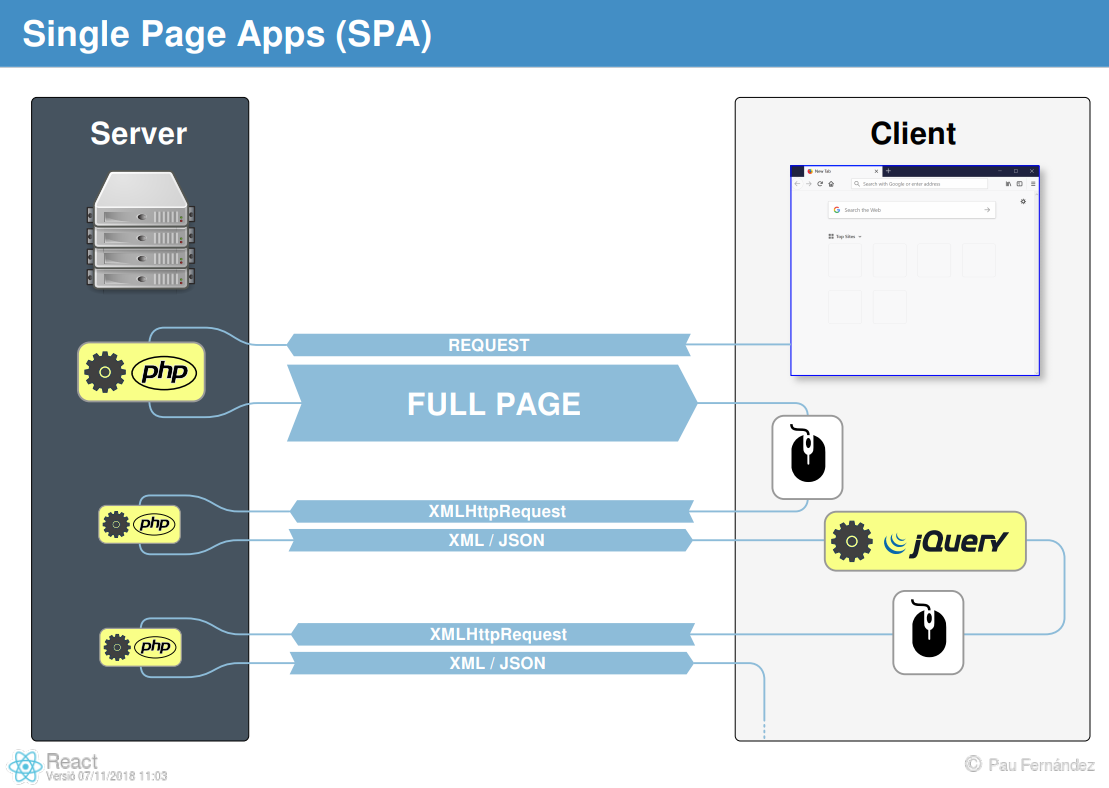
* Allows us to structure the view layer of applications, the part of the app that the user sees. (Model View Controller).
* Use of reusable dynamic components that can hold state and data.
* No need to separate the Markup from the logic thanks to the use of JSX. (JavaScript syntax extension) Allows us to write dynamic html but we’re actually writing JavaScript.
* Very efficient with pages with a lot of interactivity because it uses the Virtual DOM (Document Object Model). It allows us to update some parts of the UI without reloading the whole page.
* Benefits in performance and testing.

## Alternatives

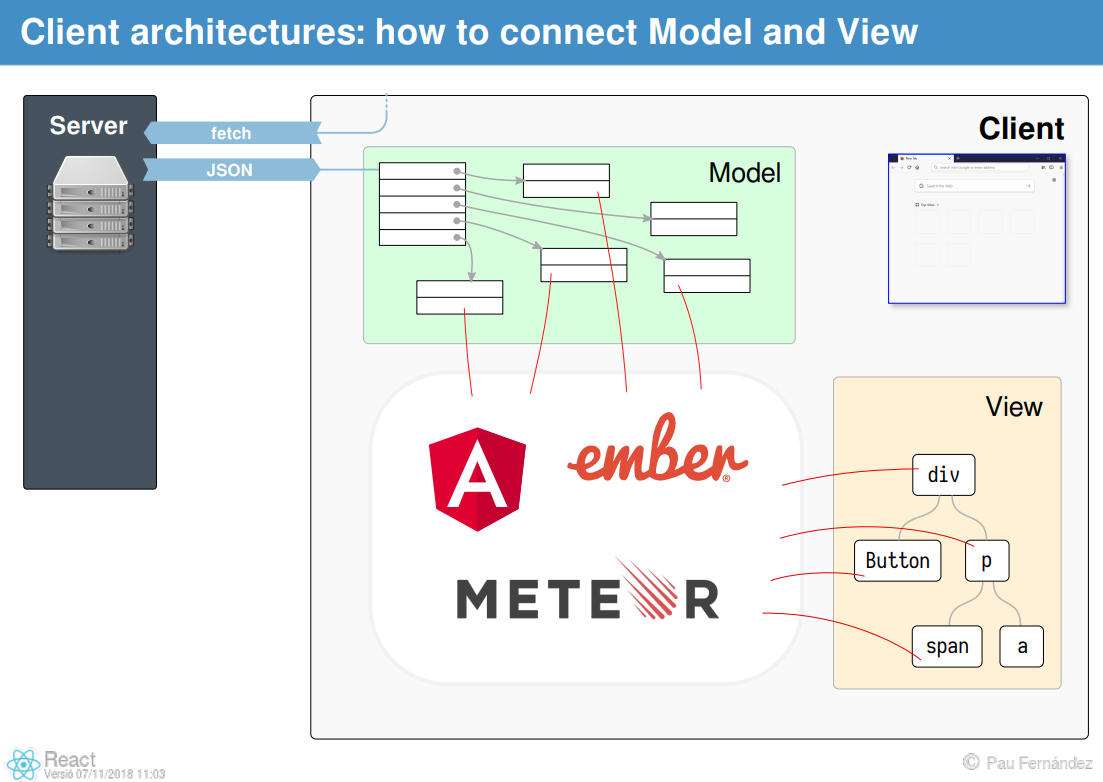
### Server-side rendering



### Single Page Apps (SPA)

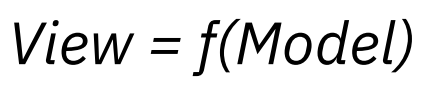


In jQuery, we patch the DOM whenever we receive the results of AJAX calls.

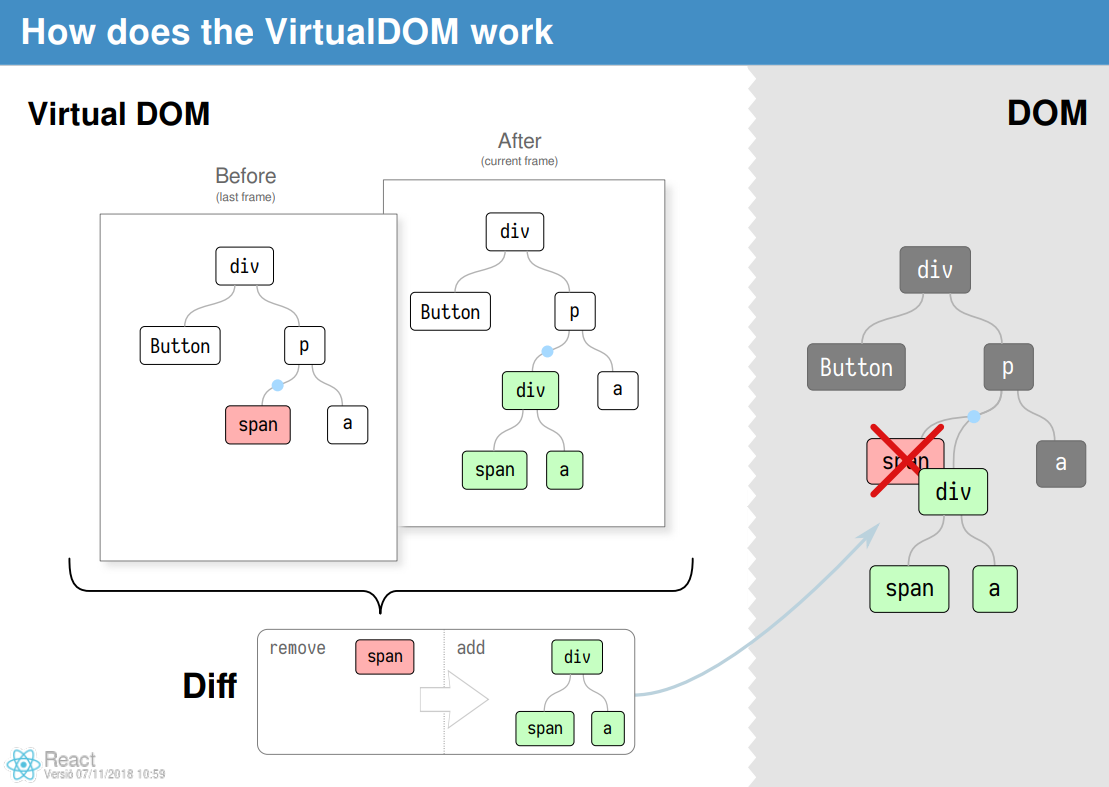


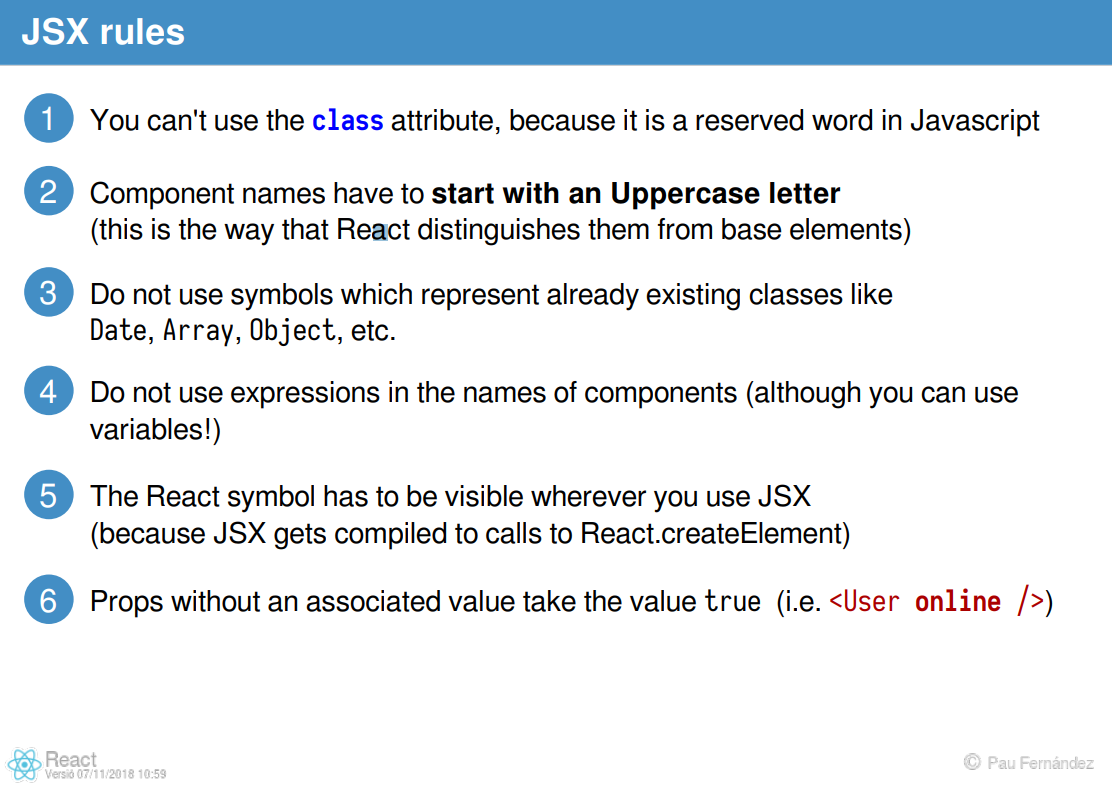
In other frameworks, some machinery lets us associate parts of the model with the views that represent it with two-way binding and similar techniques.

## React: The View is a function of the model

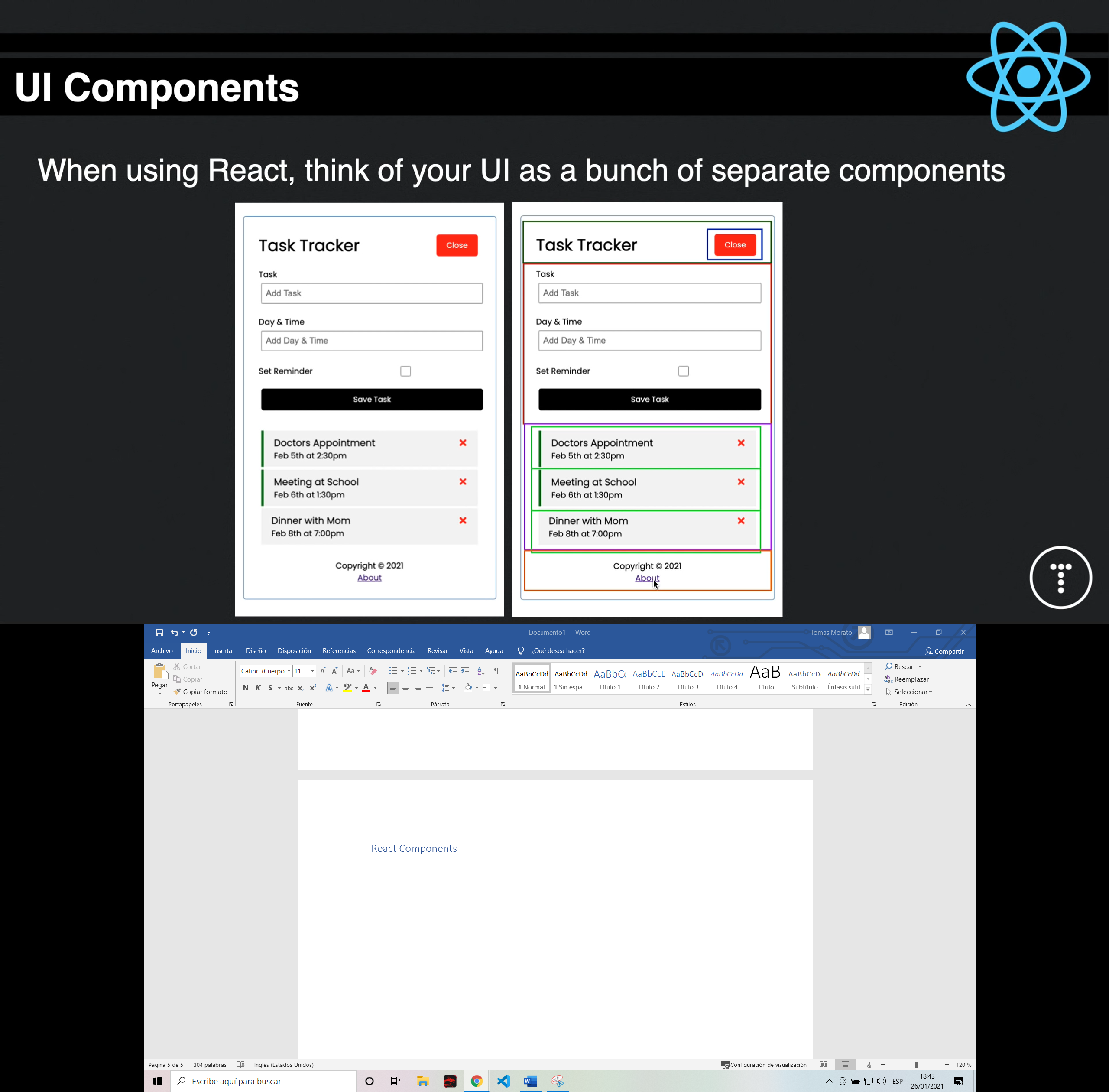


* To do this, React lets us write components (functions and classes) that generate the whole view (the whole page) from the model.
* These components produce virtual DOM elements which produce "frames" of the UI, at different points in time.
* React takes the role of comparing two "frames" (two snapshots of the whole view) and altering the DOM to produce the new UI from the old with minimal modifications (i.e. fast and efficient).





## React Components



Class based components vs. Function components:

functi

* Components can have “state” which is an object that determines how a component renders and behaves.
* Some times we use App or Gloval state, which we describe as a state that is available to the entire UI, not just a single component.
* Prior to react 16.8 we had to use class-based components to use state. Now we can use functional components with hooks.

## React Hooks

React Hooks are functions that let us hook (*enganchar*) into the React state and lifecycle features from function components.

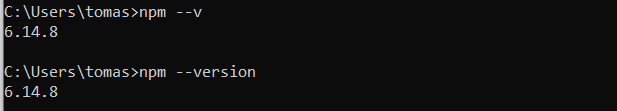
* useState – Returns a stateful value and a function to update it.
* useEffect – Perform side effects in function components. (ex: Fetch data)
* useContext, useReducer, useRef…

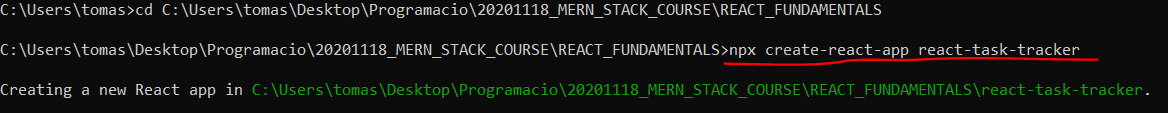
You can also create your own custom Hooks

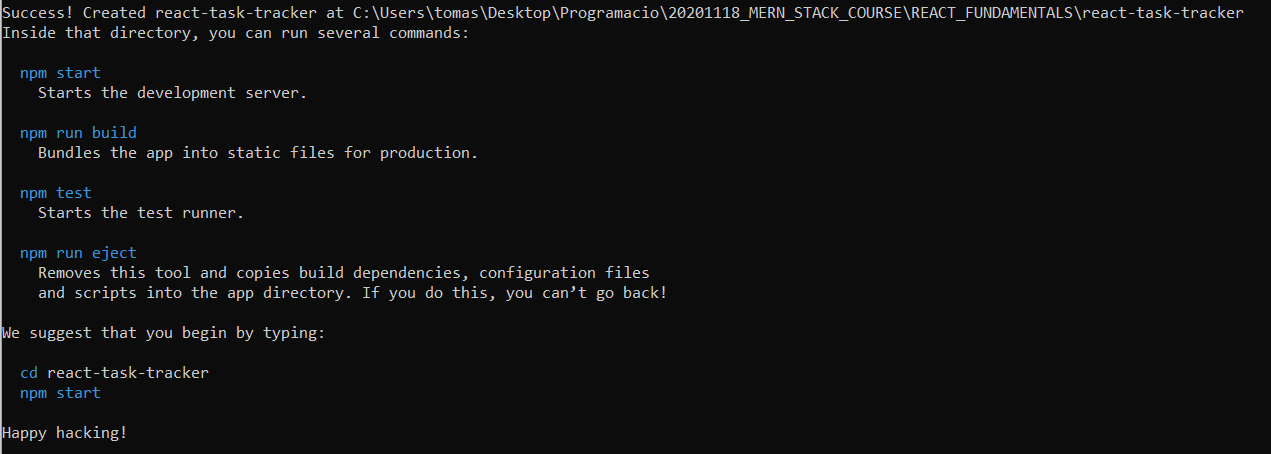
## Create a new react App

One of the easiest ways to do so is using a program called Create React App which is a CLI tool to easily set up a boilerplate with all the files and packages needed. It also has a development server and a bunch of other functionalities.

1. Check you have npm installed

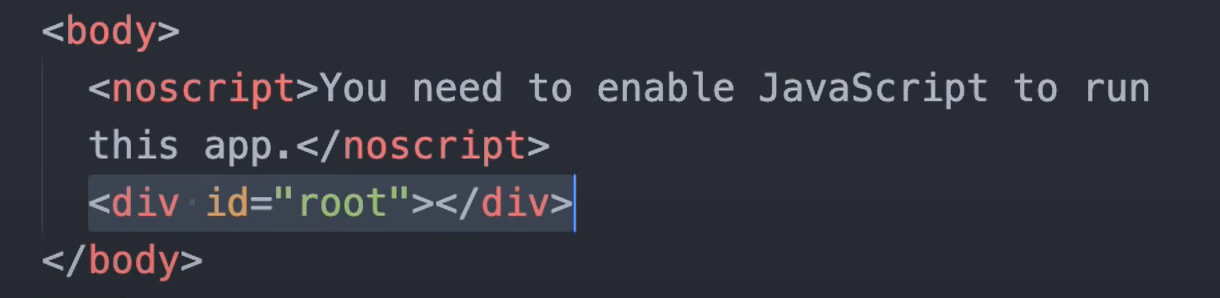


1. 



Note how the files are organized:

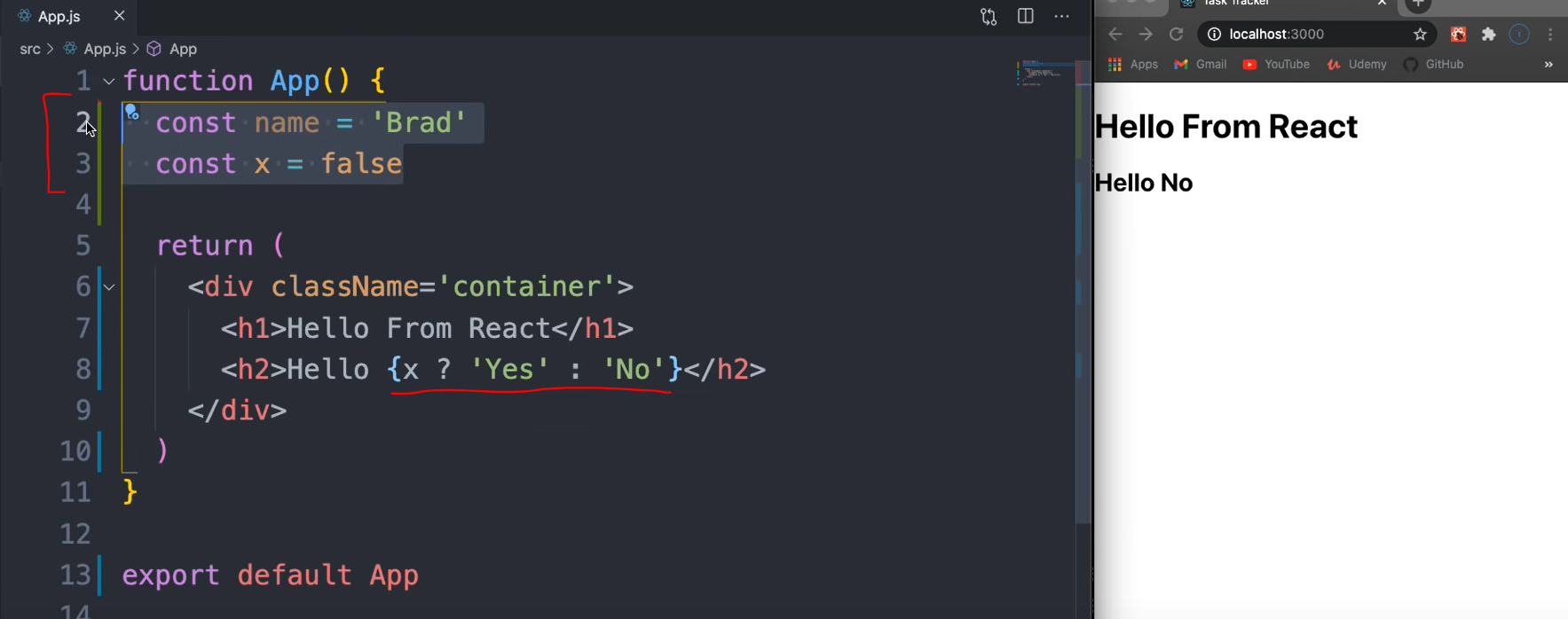
* We have a **public folder** with a basic **index.html** which contains a single <div> with an id of root.



* In the **source folder** where we put all our components, we have the **index.js** which is kind of the entry point for react, we import react and reactDOM and we use a method to insert our app inside the div created in the html document:



* This <App /> comes from the app.js file and contains the root app component.

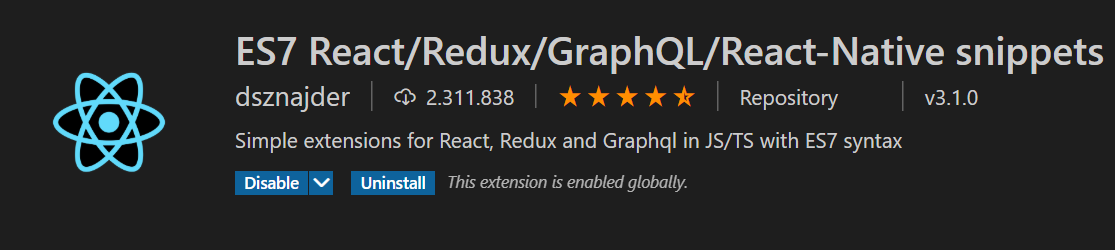


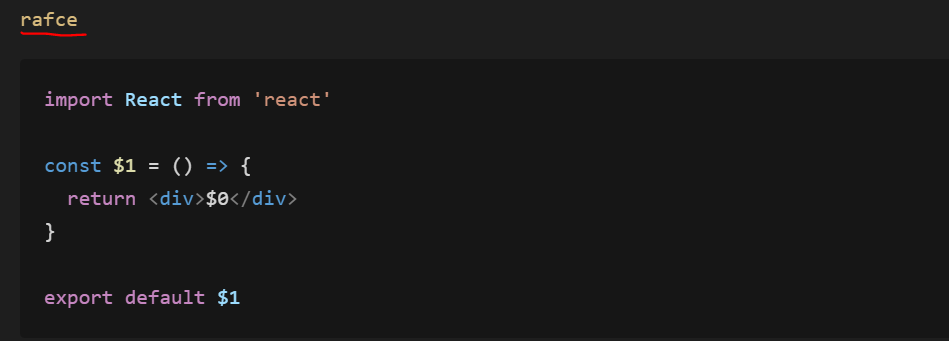
Note how we can use variables and conditionals and use JavaScript inside the html tags.

## Create Components

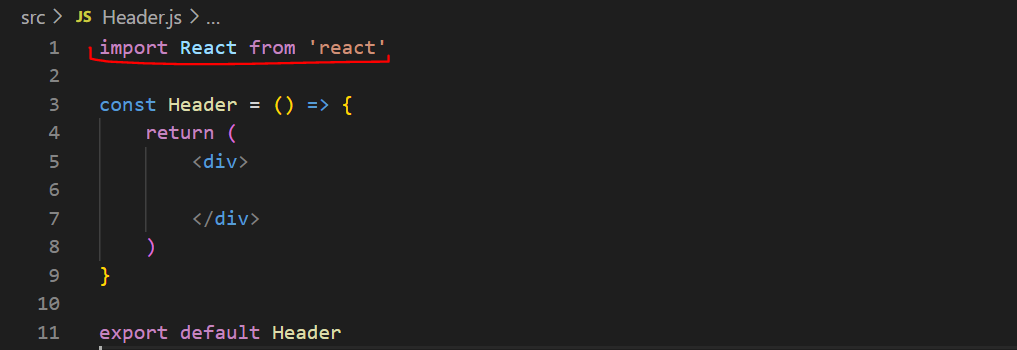
Create a folder inside the src called components. Inside this folder we’re will create all the components for the project.

Start by creating Header.js (By convention we use uppercase for the components). Create this file and look for a extension to create the boilerplate you need.

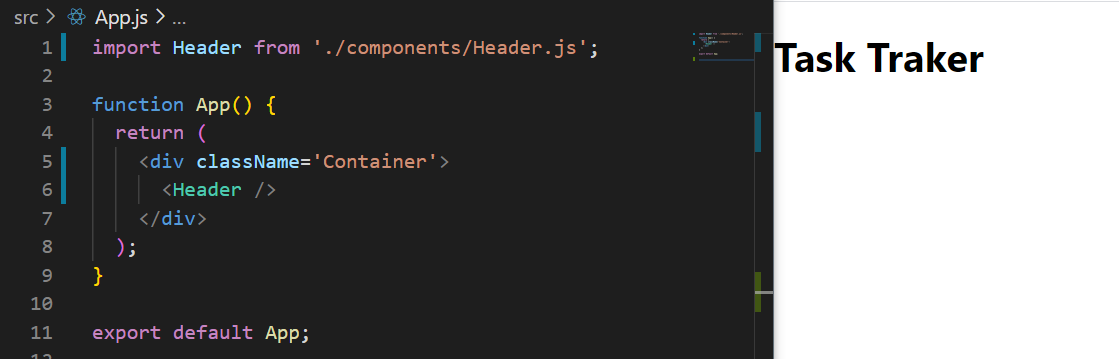




With this **snippets** creates a basic arrow function that exports down in the bottom.



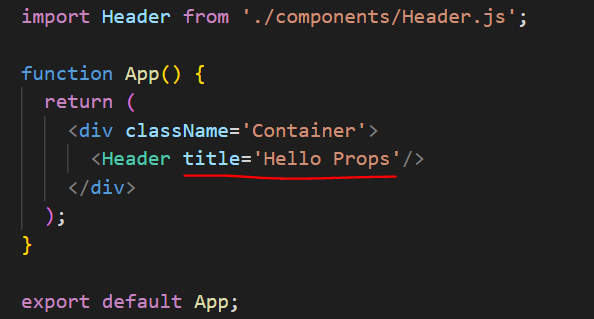
Actually, in the lasts versions of react we don’t need to import react in each component. Then we need to import it in our main App file and It will render out.



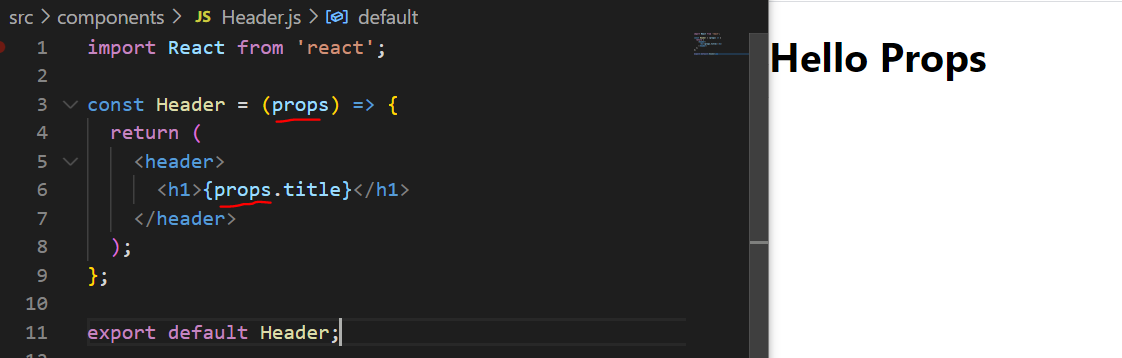
To do it as a class component we do need to import react:

* Extend React.Component to make use of all the lifecycle methods.
* Add a render method to paint the UI.

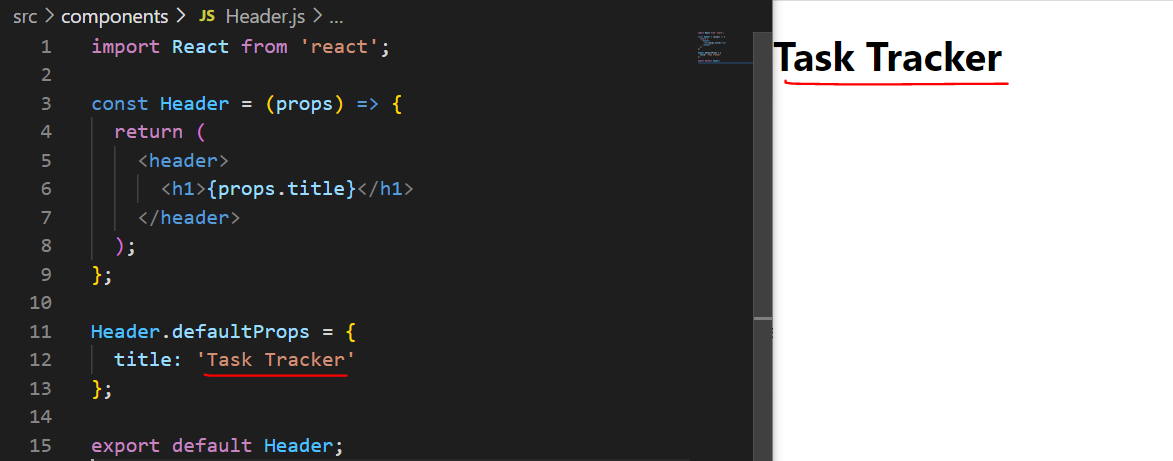
## Basic use of props



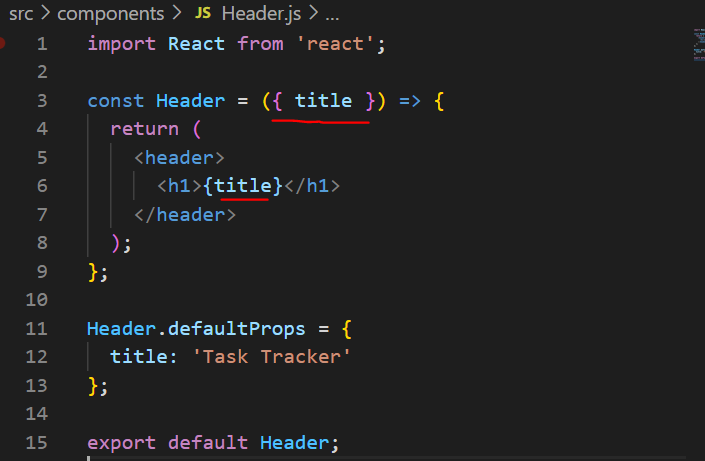
We can pass anything we want to this component but then in order to handle it, inside the functional component we must receive it by taking in ***props*** object as a parameter:



We can also have default props in case we don’t retrieve anything.

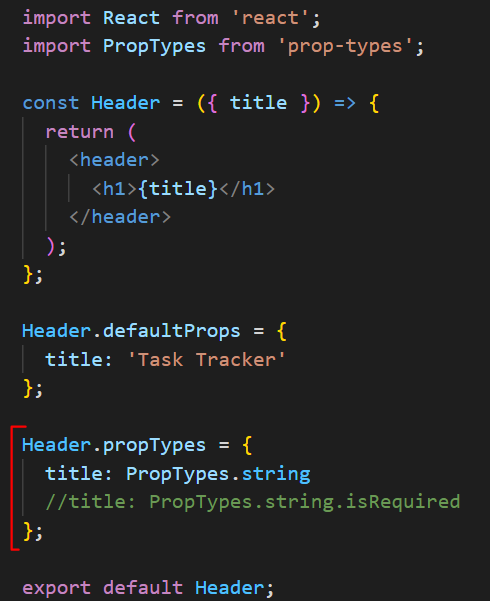


We can always destructure the parameter, in this case the props object and cleaning our code a little bit by doing:



To make our code a little more robust we can make use of PropTypes. We can import it by using the snippet ***impt***.

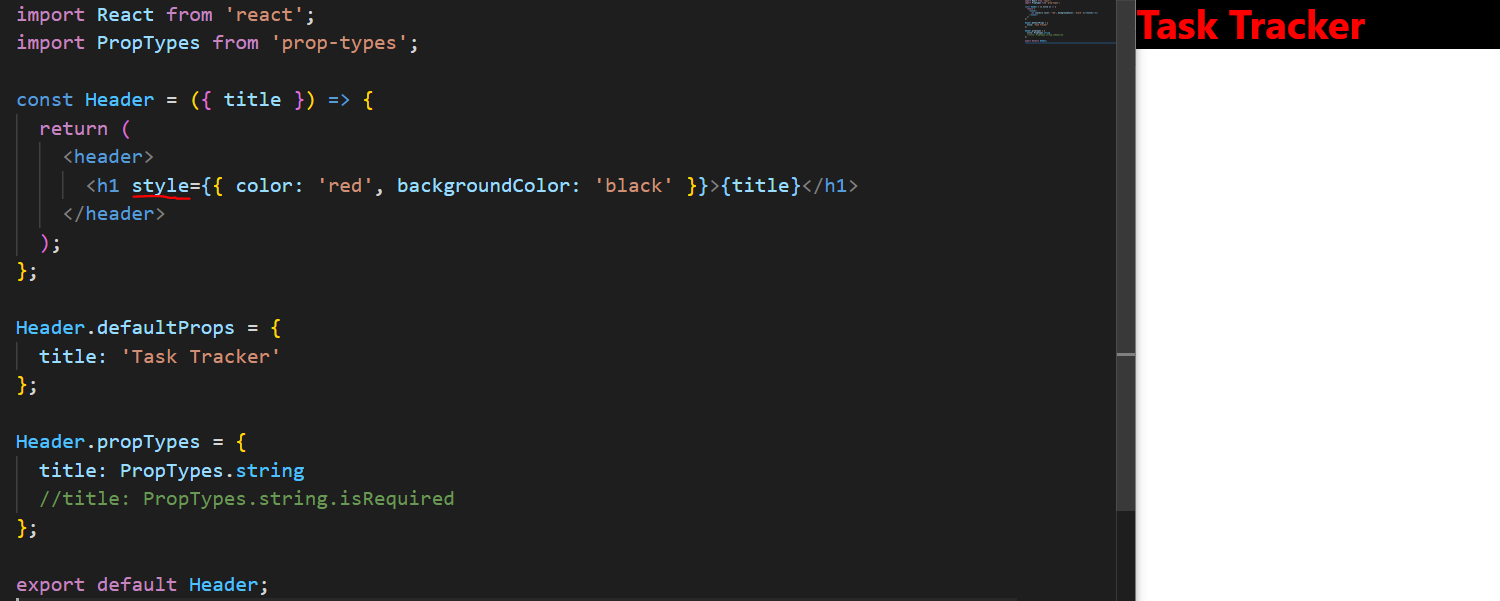
Then set the type of data we want for our props. If we pass some data that doesn’t correspond to the data type we define. It will render out but will fire a console error.

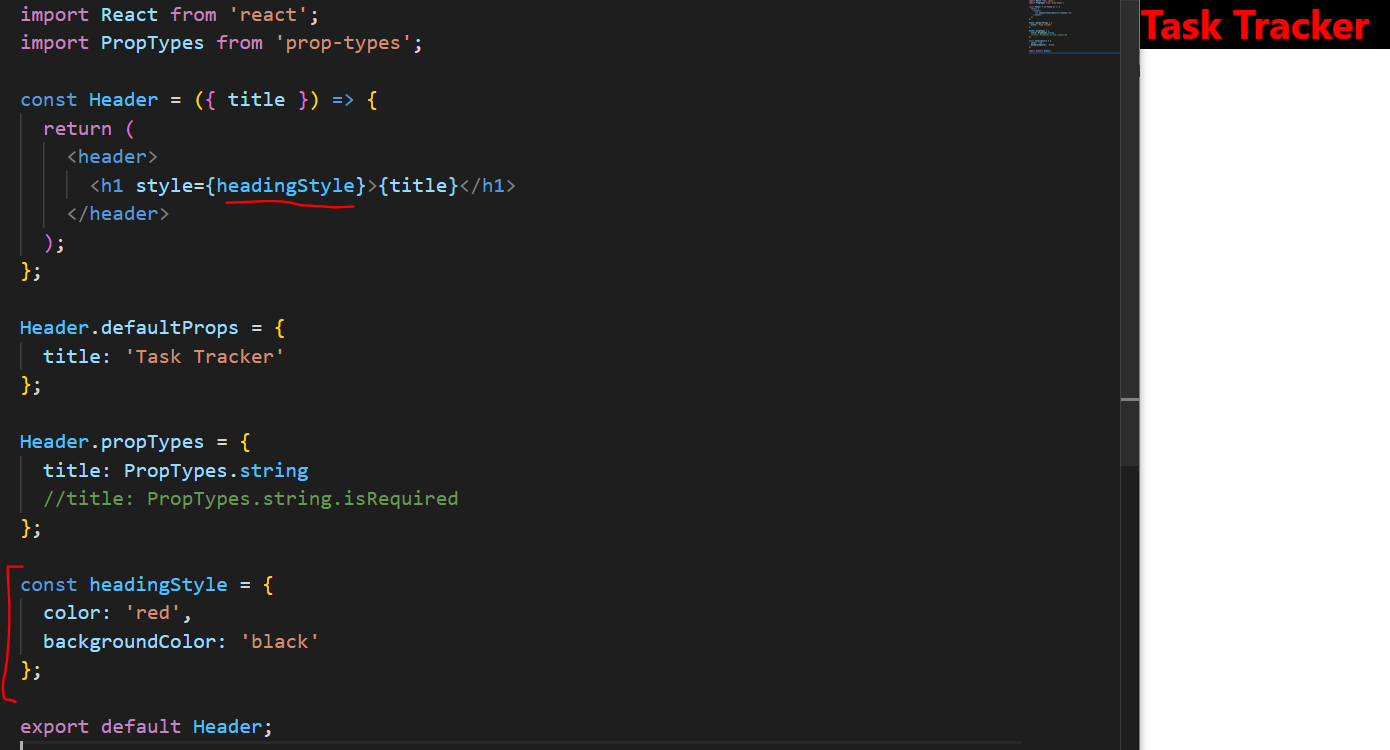


## Basic Style Components

We have a couple of ways to do it:

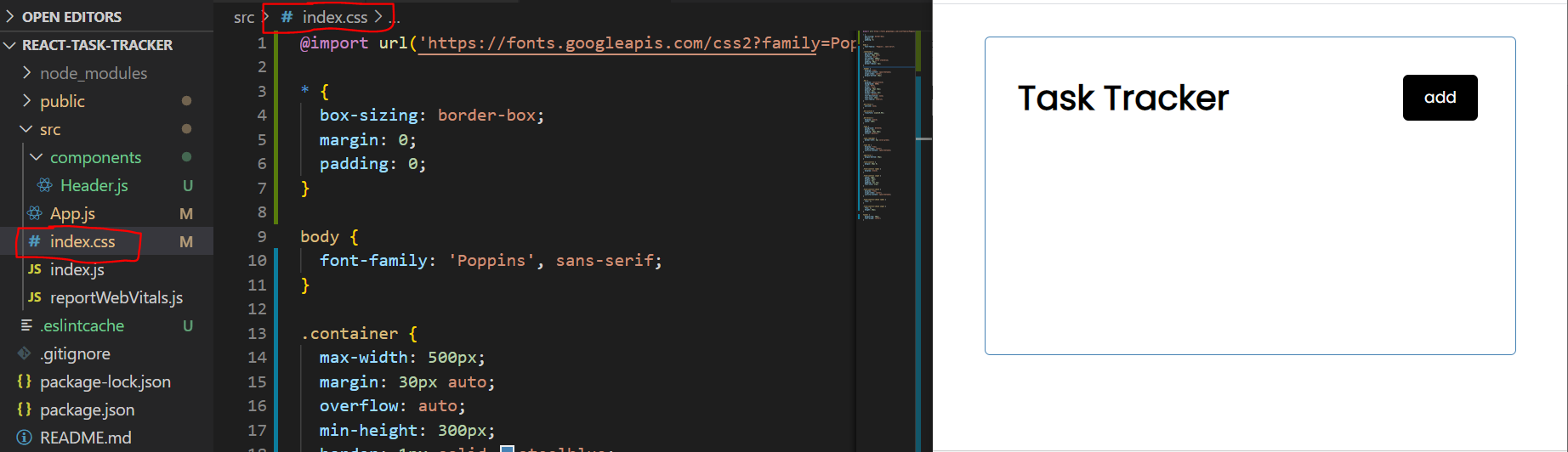
CSS in Js:



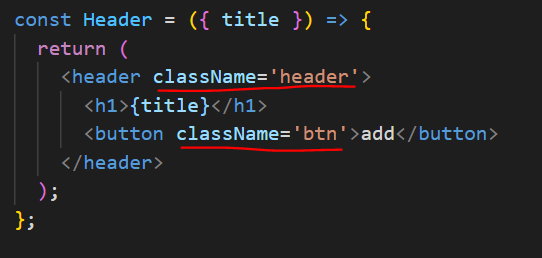


Using CSS and ClassName attributes:

Get the CSS from: <https://github.com/bradtraversy/react-crash-2021/blob/master/src/index.css>



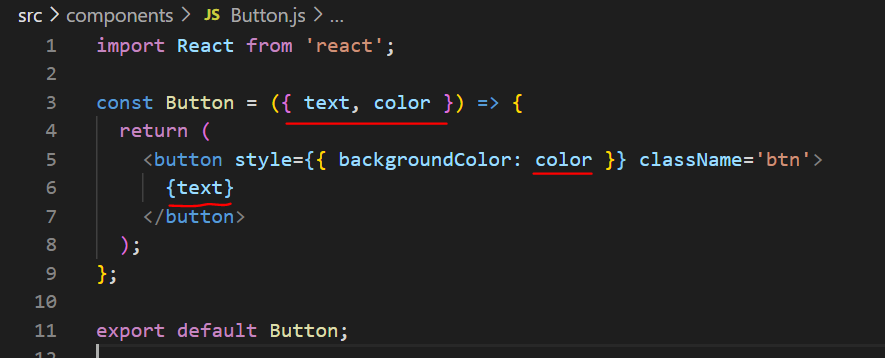
And add the classNames to the components:



## Extract Components

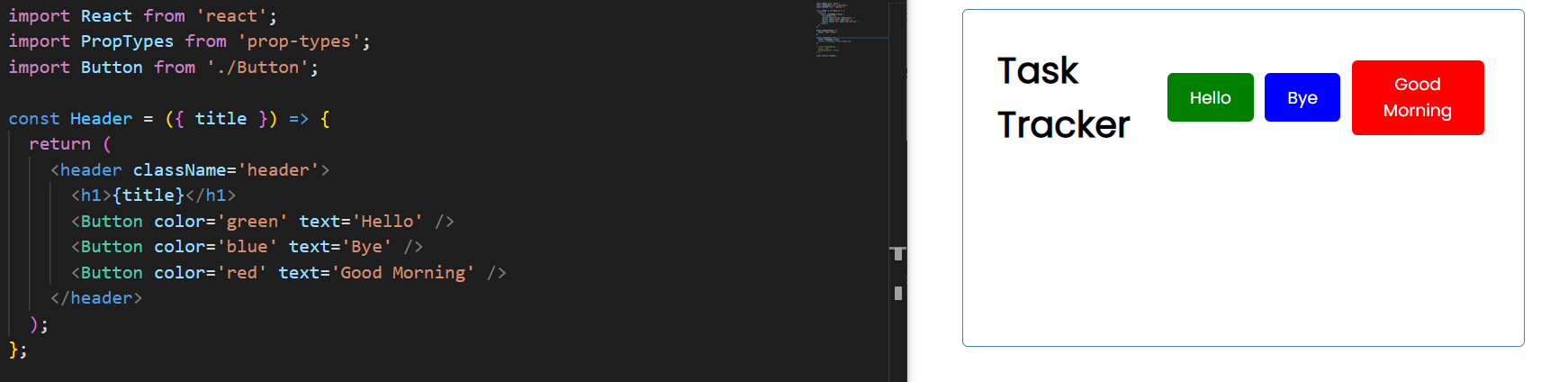
You can take out the button component to its own file and make it a reusable component:

* Add some props to the component so we can customize the color and the text



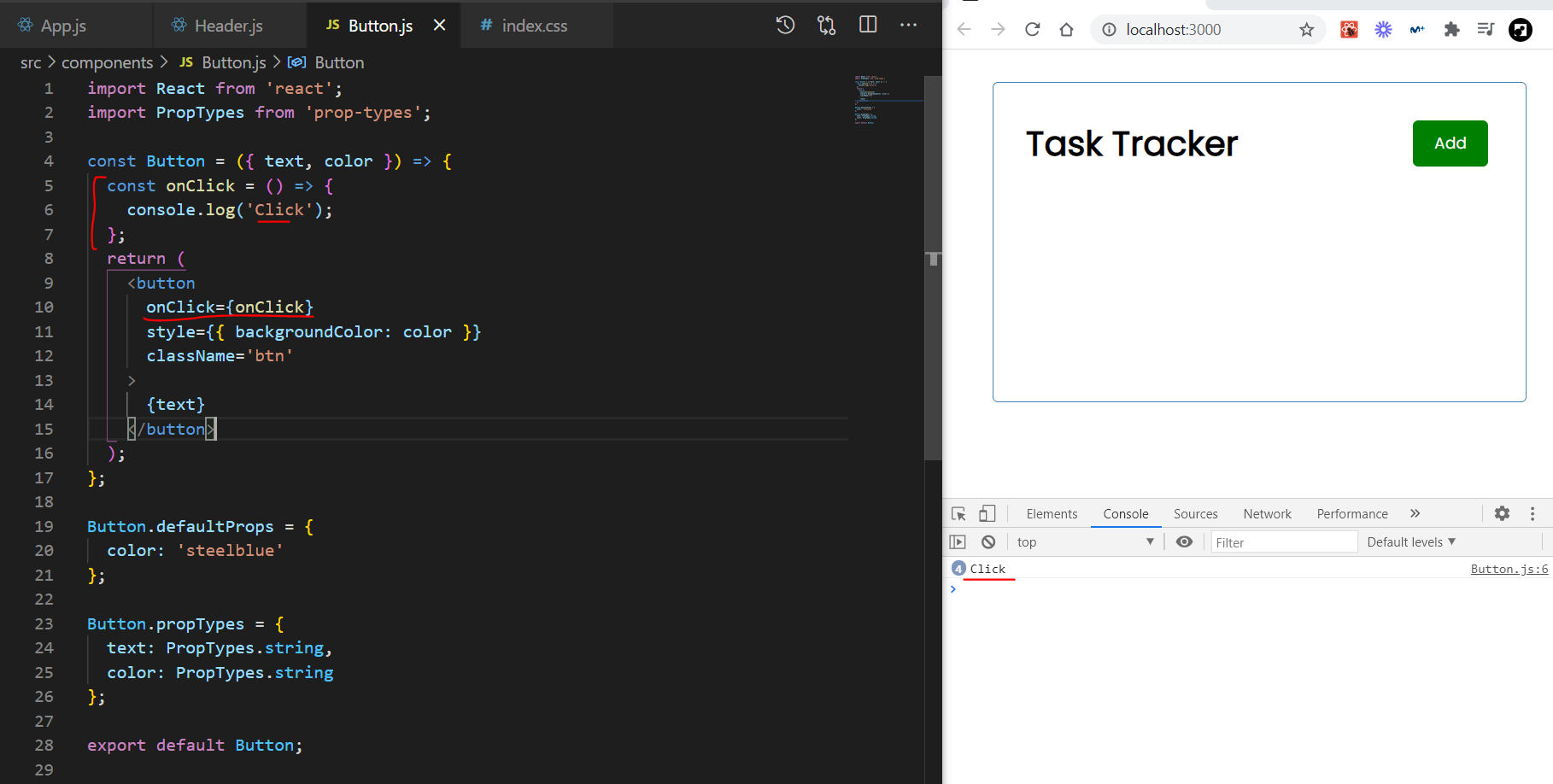
Then we can use this Component with different text and color:

* Import it in the file where you want to use it.
* Change the props you pass in.

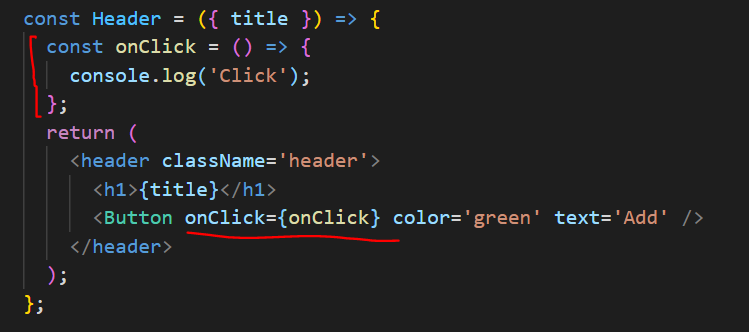


## Button Event

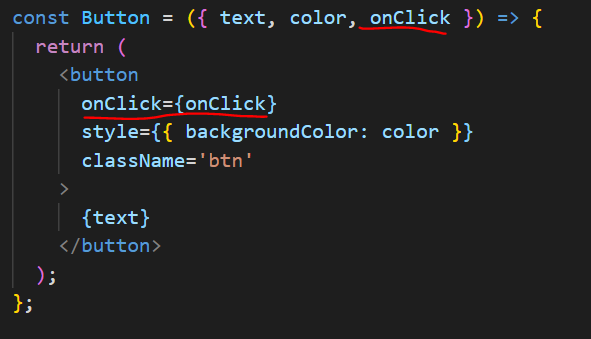
We can just call a function from the component.



But since this is a reusable component it’s not always going to have the same click. We can have that click as a prop. So we will move the event to the Header file and declare the function there. By clicking the button we will fire the function.



Remember to add the prop to the Button object:

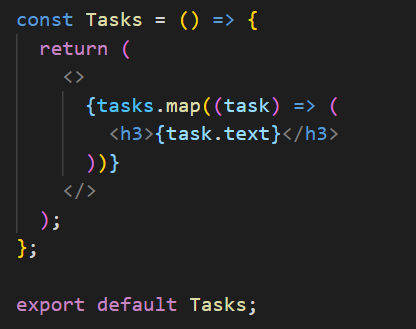


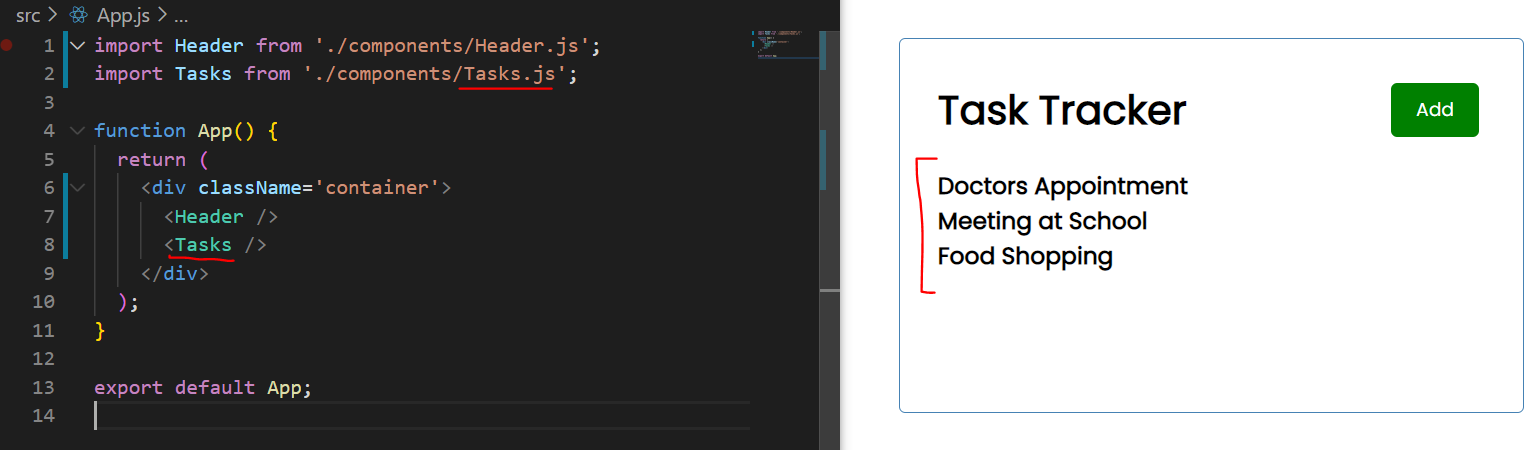
## Introduction to data

Create a new component named task. We’re going to add some dump data so we can see how to loop through it first:

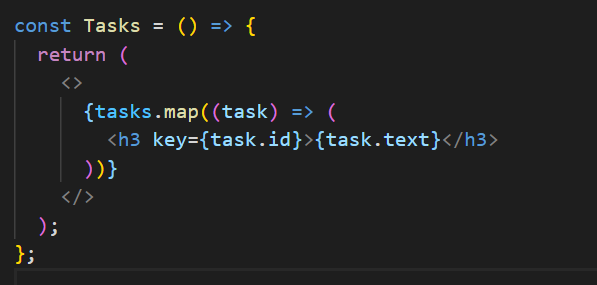


* Add this const to the Tasks.js file.
* We will use map to create a list.
* Iterate through the object and set a function to render the title of the data.
* Import the task component in the App.js file so we can render it.



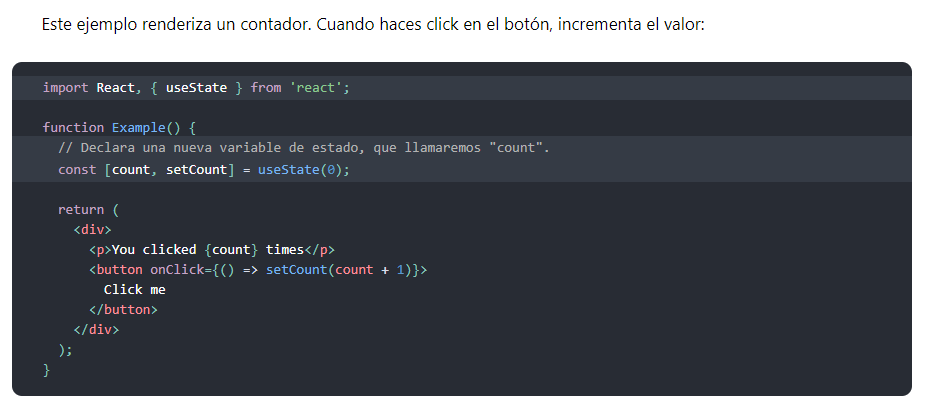


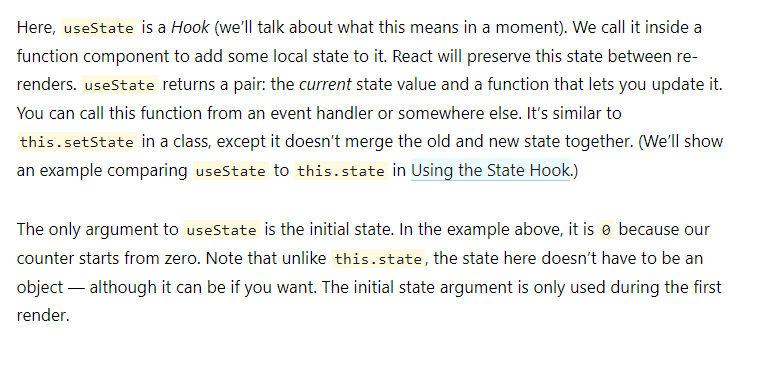
Remember to add a key to the element:



## Introduction to state

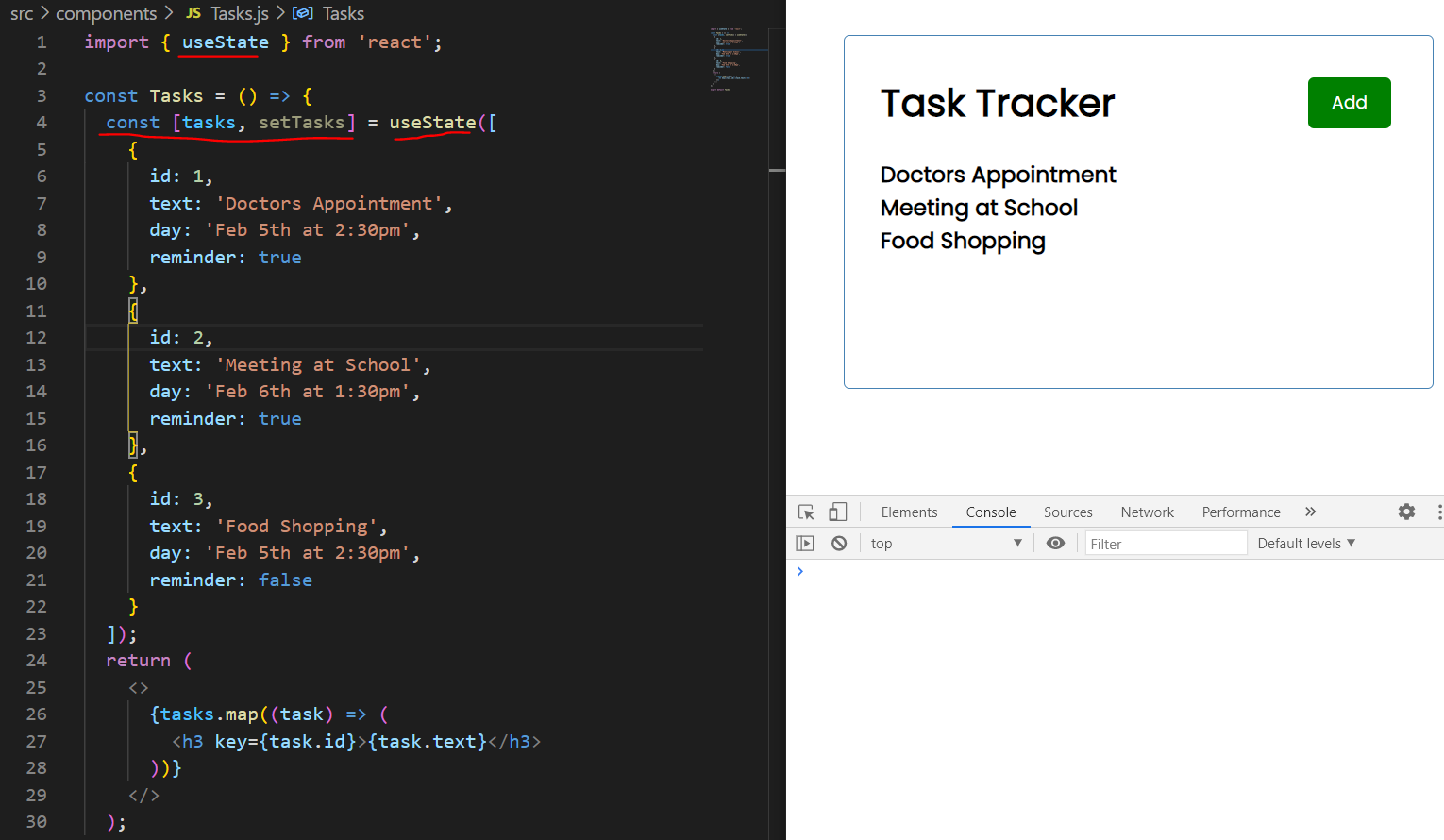
https://reactjs.org/docs/hooks-overview.html





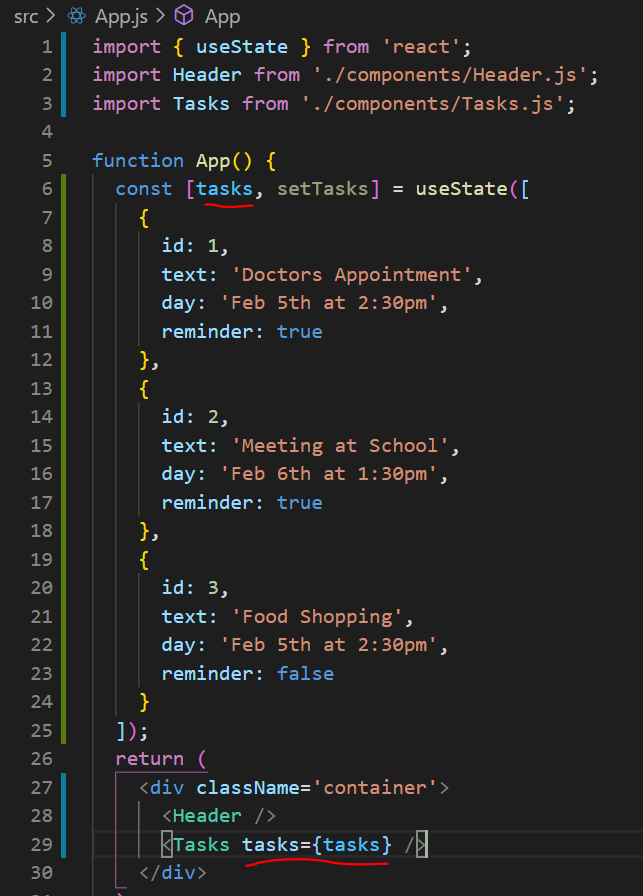
Let’s add state to our component:

* Import use State from react.
* Declare a new state variable as follows and add an update function.
* Set the default value of the state variable by calling useState(value)



Anyway, this corresponds to **App state** so we would not want to have it stored in the component itself. In a real project we will use the context api or **Redux** to manage the app state.

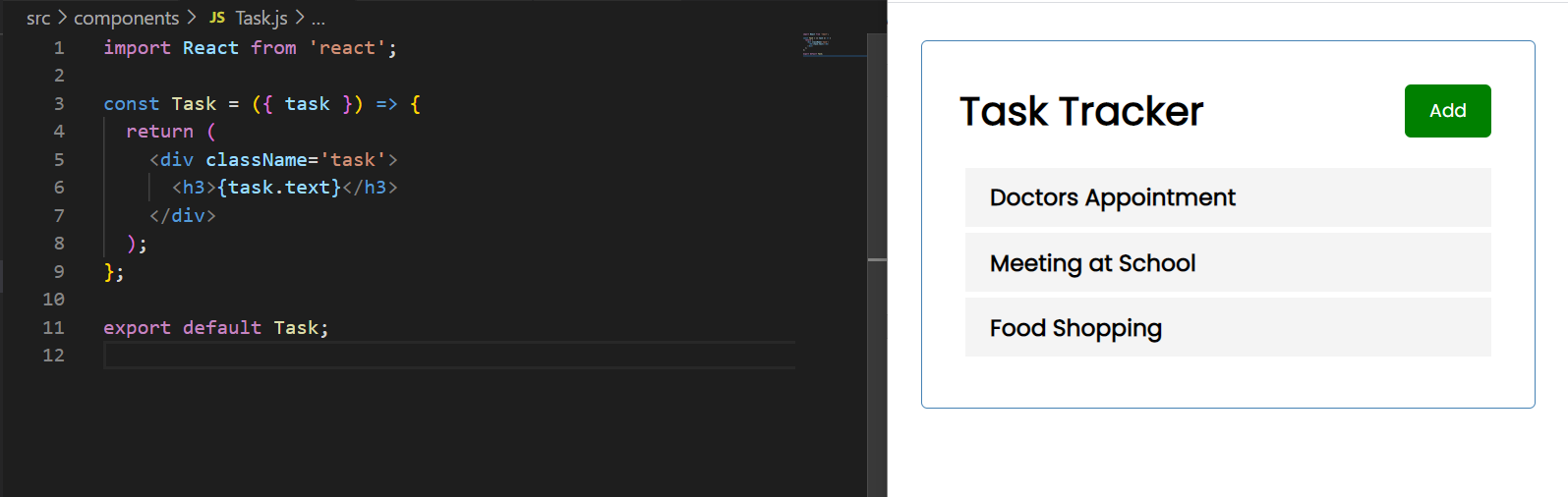
In this exercise we not use any external library but we will move it up to App.js that will make it our global state and we will pass it down to our components as props.

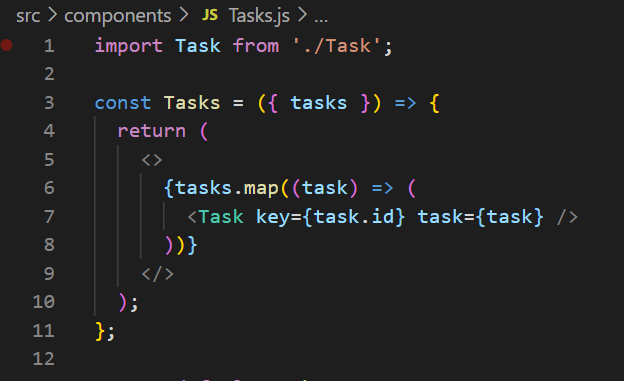


<https://www.youtube.com/watch?v=w7ejDZ8SWv8&t=1503s>

Now we can create a component for each task instead of a h3 tag.

* Create a new file Task.js singular
* Create a new component boilerplate
* Render a div with class task for the css style.
* Change the Tasks.js component so it renders the Task intead of the h3.
  + Pass each task as props





Lets add some style to the component:

* Install **react icons** (*npm i react-icons*) this way we will be able to use icons as components.
* Import the FaTimes icon
* Put it in the h3 tag and restart the dev server so we can see the changes make effect.



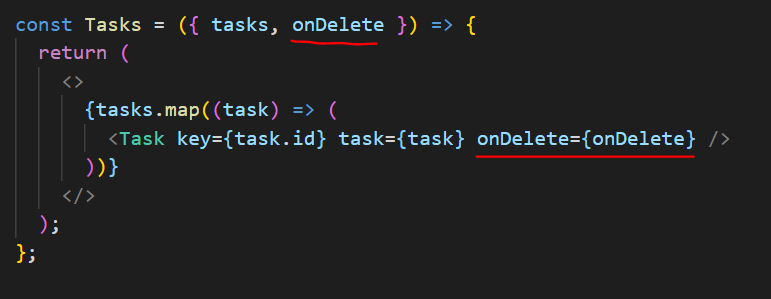
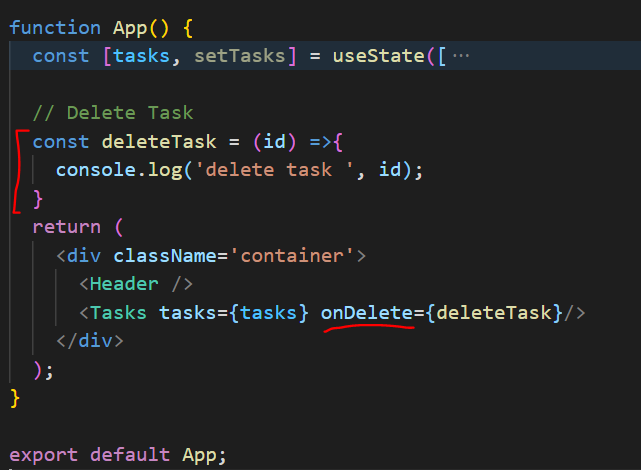
## Changing the state – delete task

We want to delete a task when we click on the red cross icon at the right side. In the actual Task component, we don’t have access to the state. Remember the state is stored in the App component so it can only be modified from that component.

With other libraries like redux, there are ways to access the state from a component pretty easy, but it’s not our case right now.

We will do it in the old fashions **using props**. We can send down a function as a prop and then fire the function when we click in the cross.

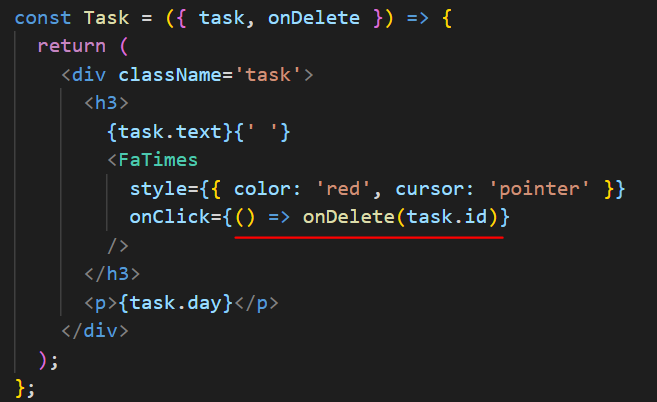
* Create a deleteTask function in the App.js
* This function will take in an ID and for now console.log it.
* Pass this function down as a prop to the Tasks component, we will call this prop onDelete.
* Destructure props on the Tasks component and set onDelete as a prop.
* Repeat the process and pass down the function as a prop to the Task component.



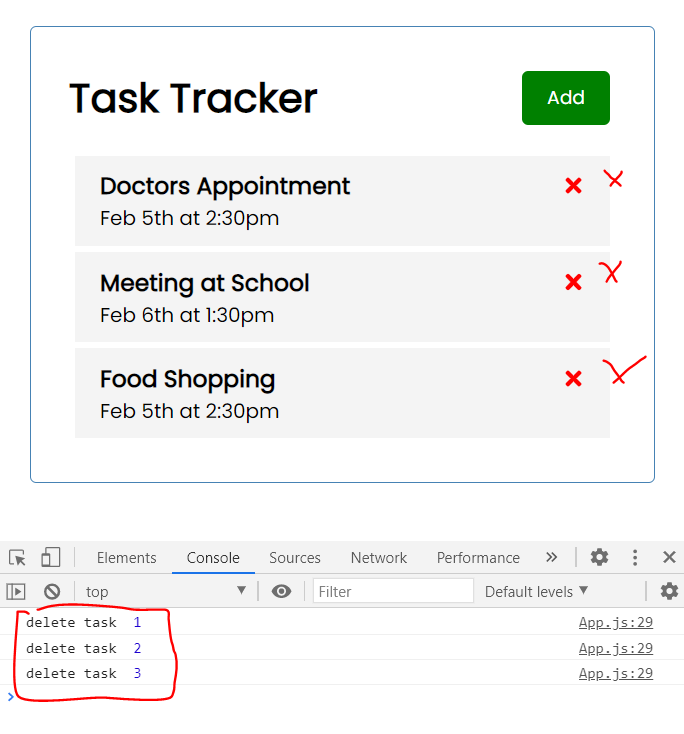


Notice that we’re not sending any params in the function call in the Task component, and by default the event information is being passed.

If we want to send a parameter in the function call, we have to call a function first:



Now we get the console.log with the exact id

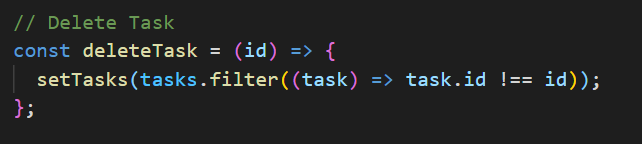


Now we get the console.log with the exact id

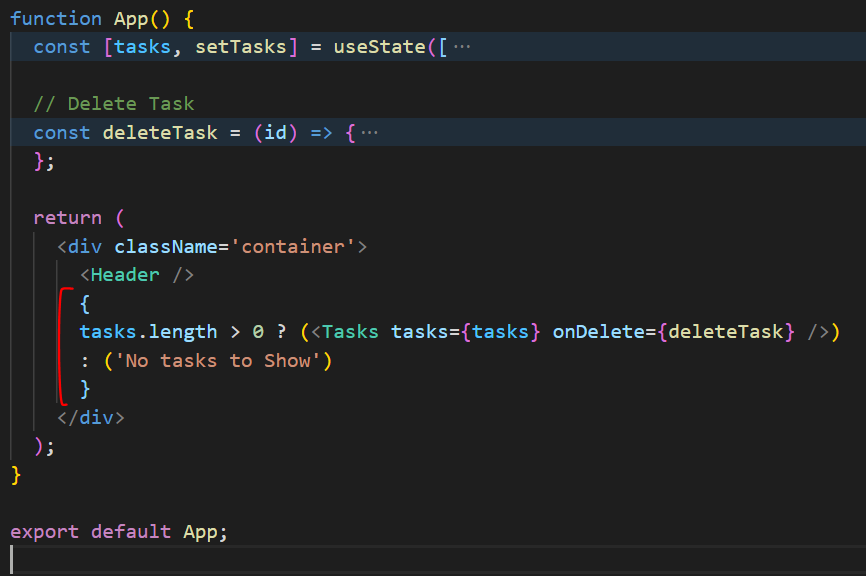
## Changing the state

Now we have seen how to call a higher function from a lower hierarchy component. In this case the *deleteTask* function stored in the App component where we deal with the global state.

In order to actually delete the task we will fake it by providing a new set of tasks. This tasks will be the same but, filtering out the selected ones. We will call setTasks to change the state and store the updated task list.



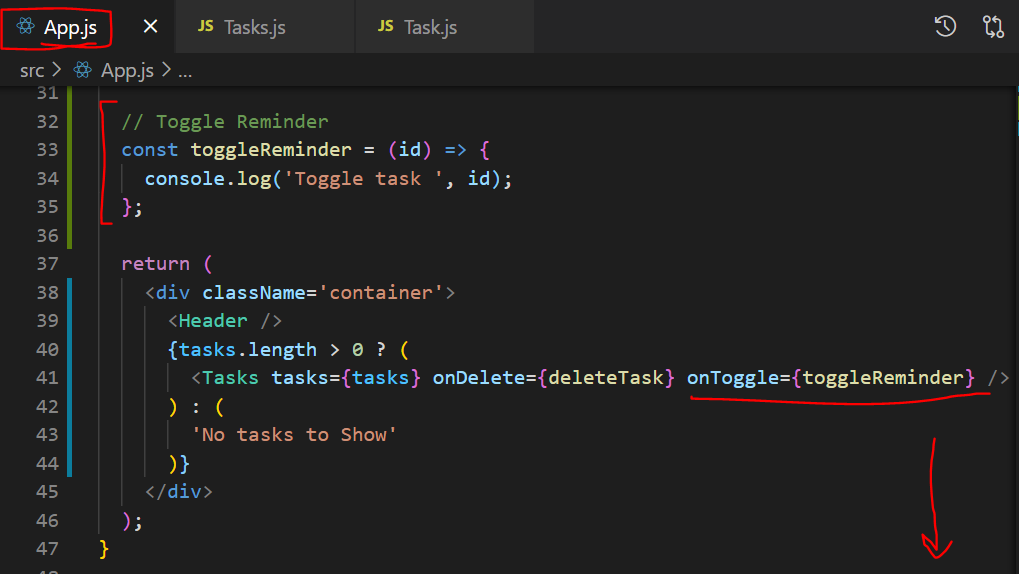
Now we might want to show a message when there are no tasks to show, we could easily do it as it follows using the conditionals:

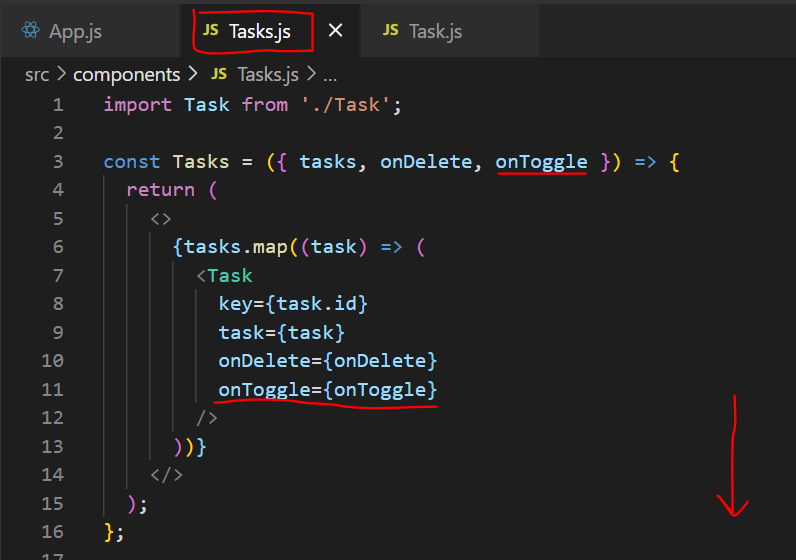


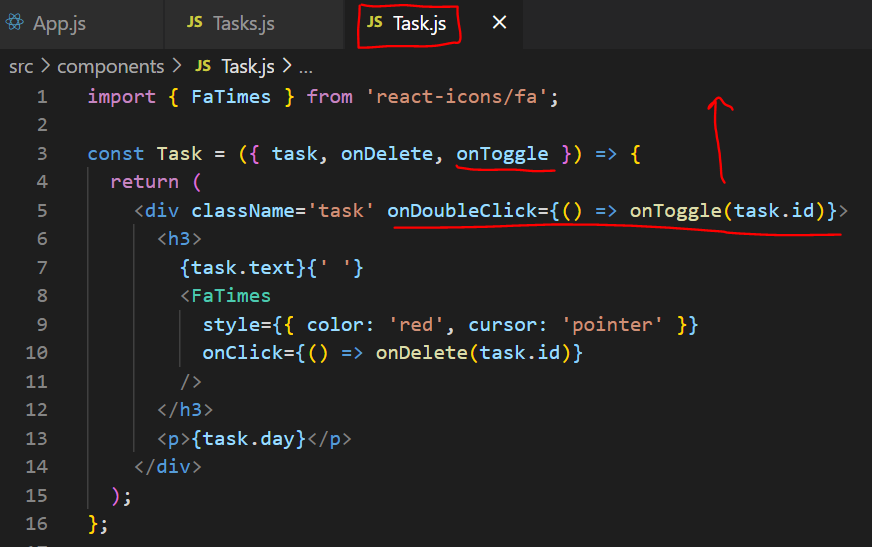
## Changing the state – Toggle reminder

Let’s start by creating a function in the App.js file.

* Create an arrow function that receives an id as a parameter and logs this id.
* Just as we did with the delete function pass this function down as a prop called onToggle.
* Pass it to Tasks and catch it, pass it to Task and catch it.
* In the Task component main div, set an event (onDoubleClick) and pass the function that we created (onToggle), as we need the id we will need to make a function and pass in the param task.id

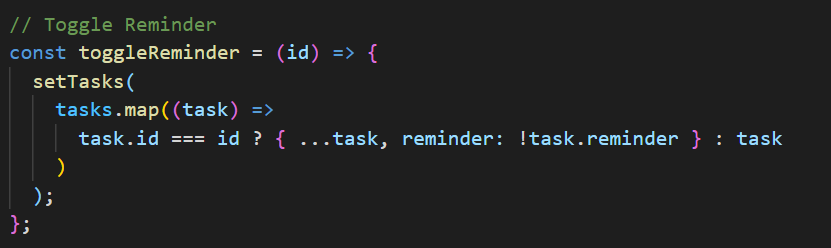




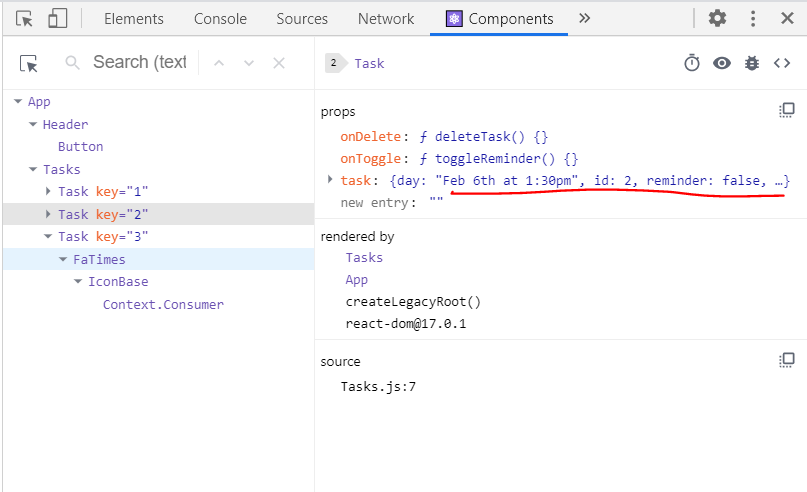


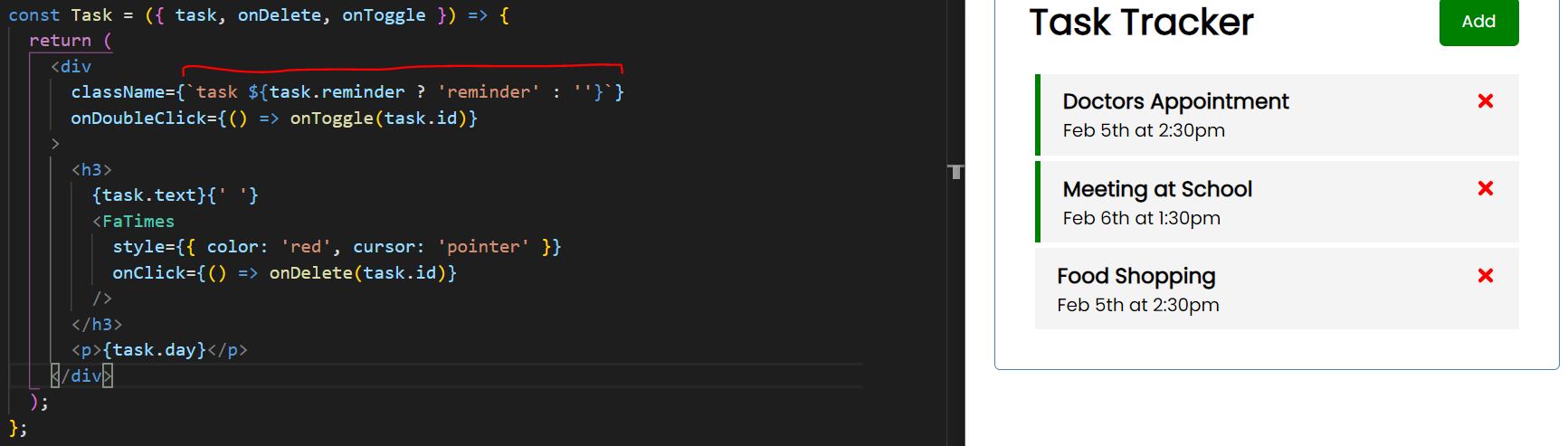
Now we have to actually change the state of the clicked task:

* We will again use setTask and map through the array



Right now nothing changes in the UI we can just check if the task changed its state in the dev tools:

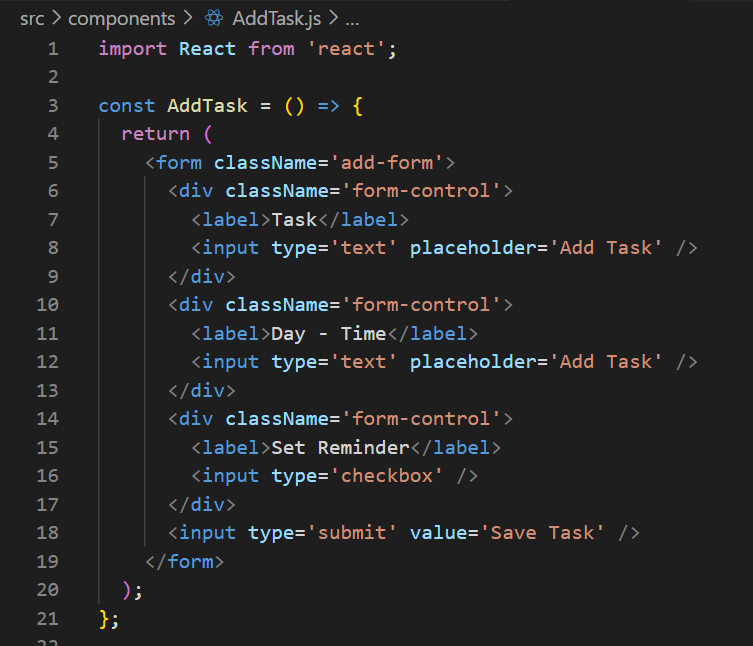




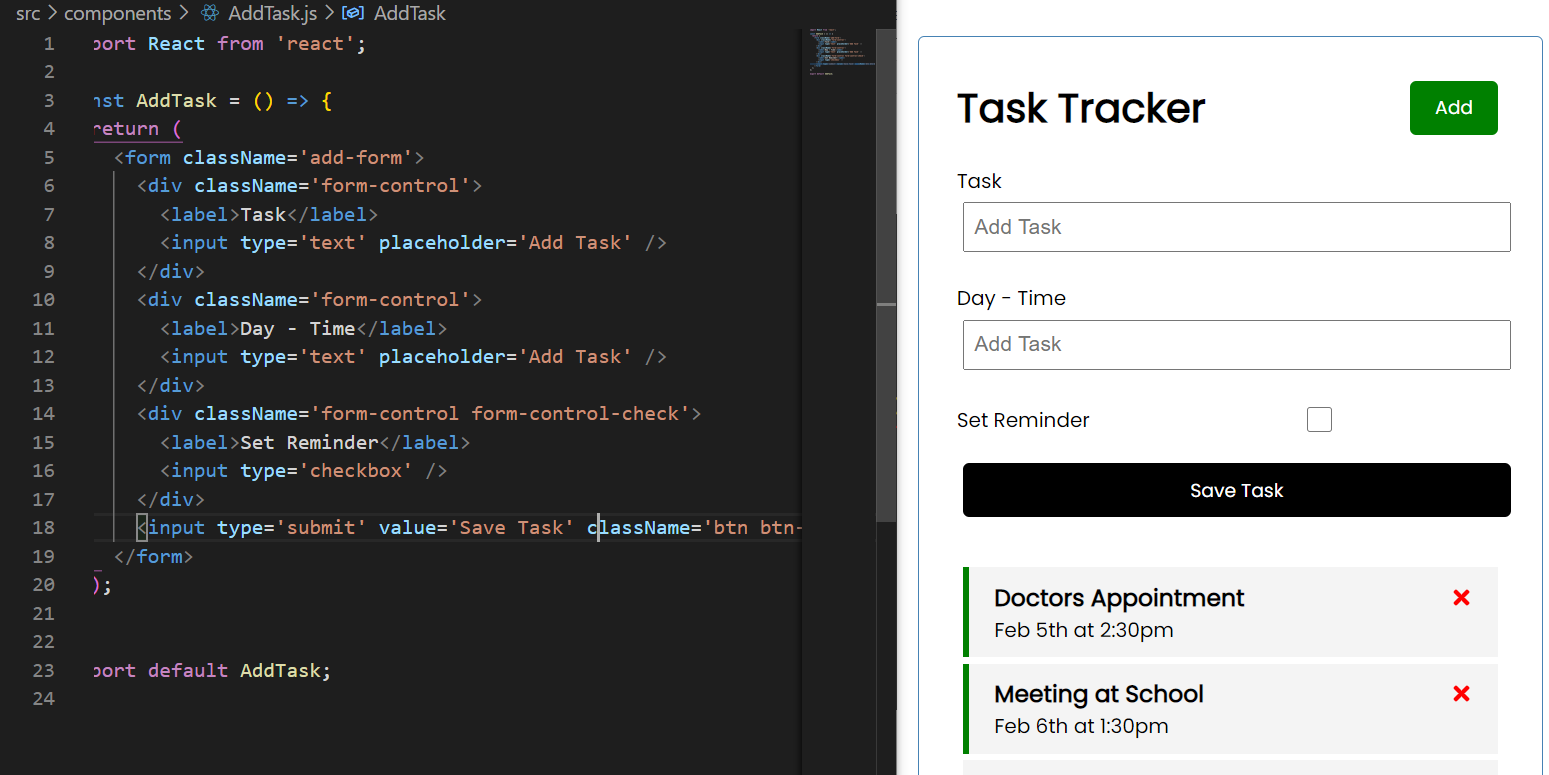
Now we can make use of the reminder class to change the task div according to the task.reminder prop.

## Creating the form

Go ahead and create a new addTask.js file and create a component.

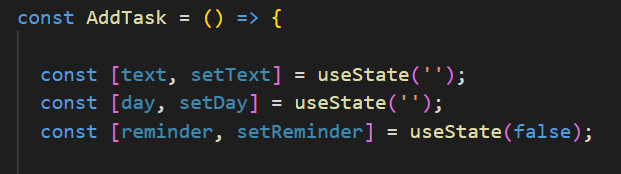


Add some more style classes:



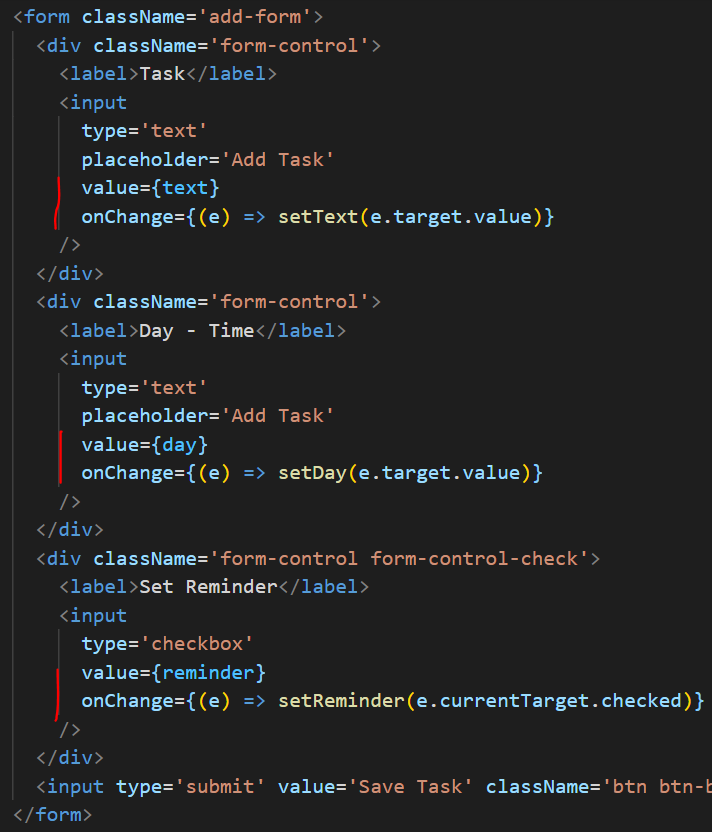
Each input is going to have its own state, **component level state**, not app level state. Bring in useState and create a variable to hold the state. Remember:

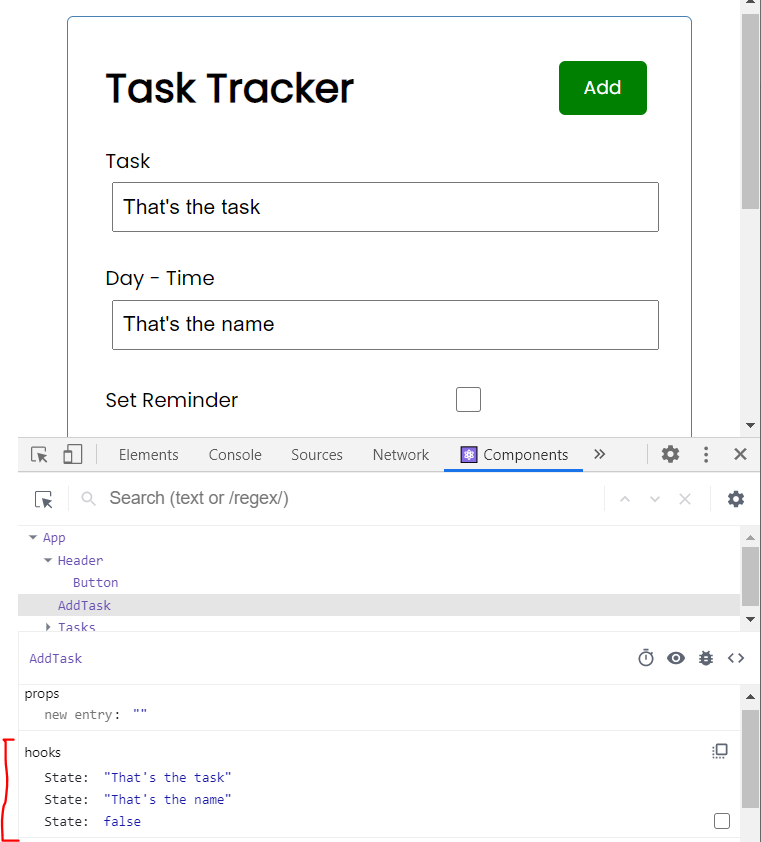
* Create a variable to hold the state and declare [ variableName, updateFunction]
* Set the default value of the state variable by calling useState(value)



Now in each input we’re going to set the value of the input to the value of the state. We also need to declare an onChange event to change the state.

* Declare a onChange event with a function that receives as a param the event object and calls setText function to update the state.
* It access es the event object e.target.value to get the actual value.

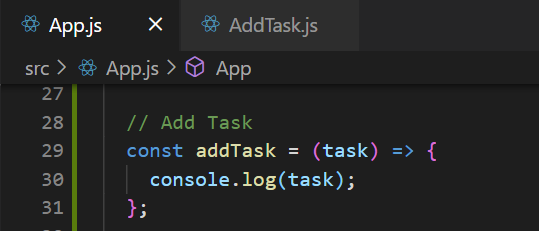
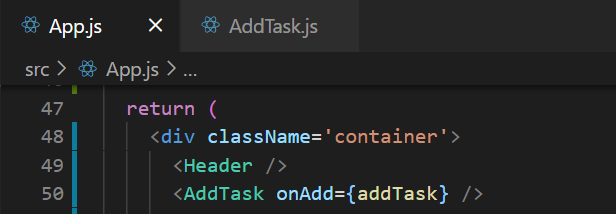


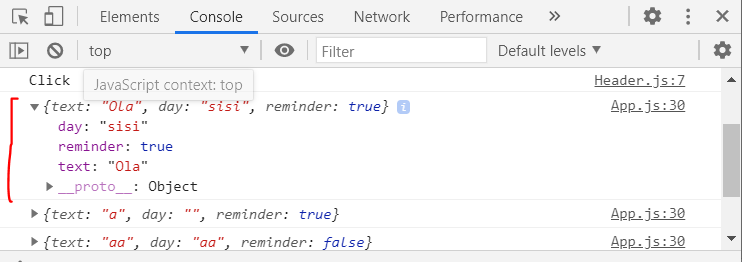


## Submit the task

Now, to submit the task we’re going to have to create a function in our App state, App.js file.

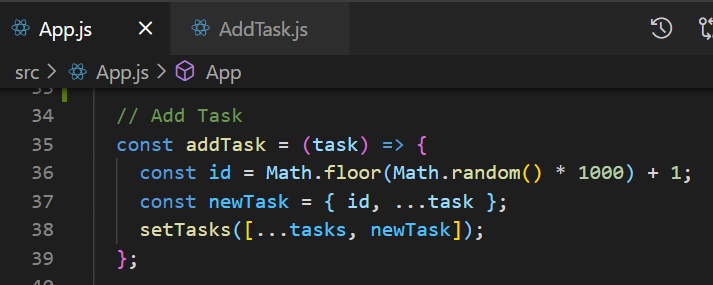
* Create the function addTask
* Pass the function as a prop named onAdd to the AddTask component.
* Take in the prop in the AddTask Component.
* Create an onSubmit event in the form and create a function.
  + Arrow function that takes in the event
  + E.preventDefault to avoid redirecting to a page
  + Add some validation.
  + Call the onAdd function recived as a prop
  + Clear the form

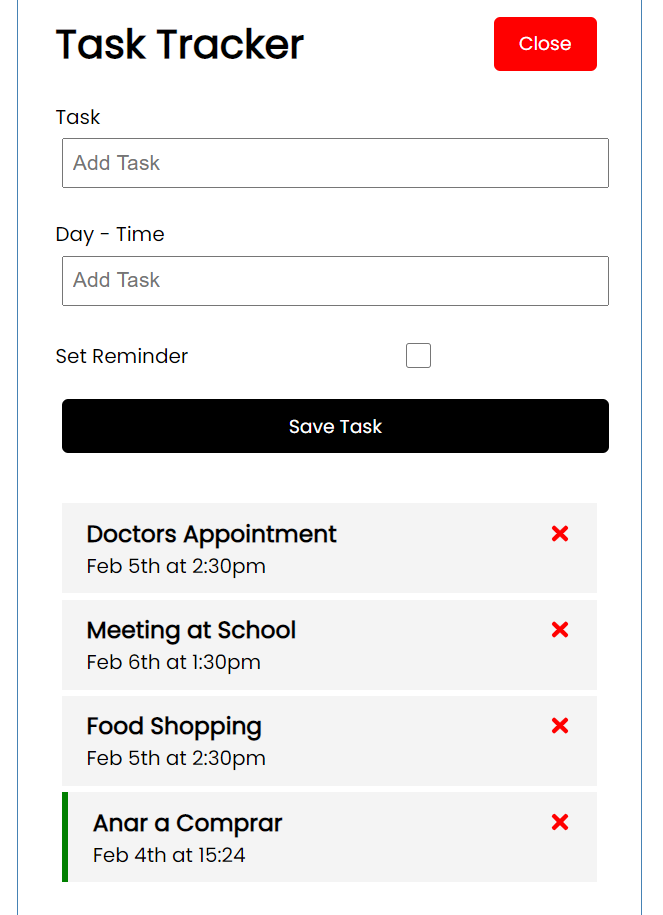
 

Note how we revieve the data and it logs in the console. Now we want to add the task to the App state. There are a lot of ways to do that.

* Add an id to the task and create the object
* Call setTasks an add the new task to the array



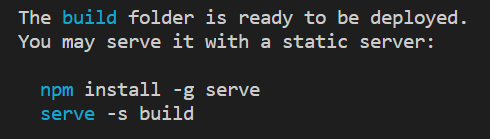


## Production Build

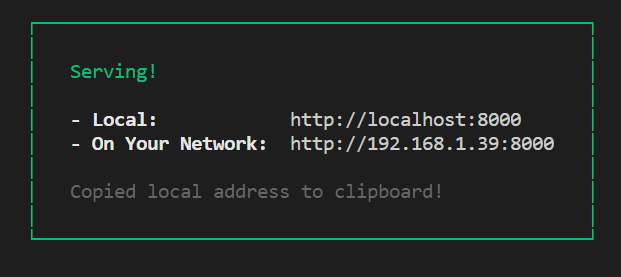
To build the static assets when we’re ready to deploy, we can use ***npm run build***. With this command we will create an optimized production build and a folder inside a project called build.

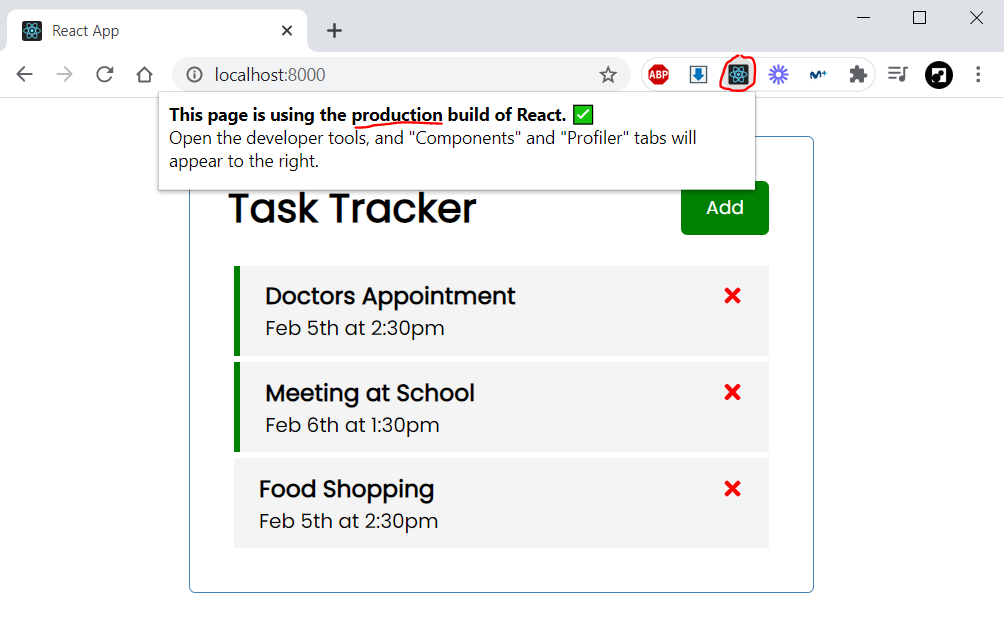
In this folder we will have all the static assets. In case we want to deploy the app, this will be the only folder we need. All the other folders are meaningful only in a development environment.

To check the production build we can make use of the serve package, go ahead and install it globally:



We can also specify which port to show the build: ***serve -s build -p 8000***

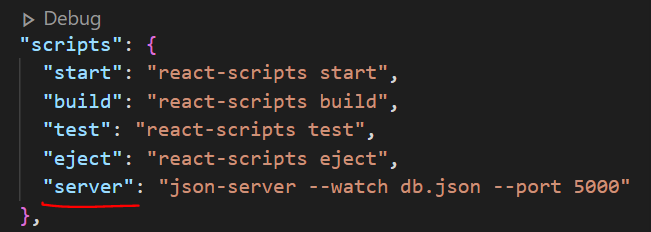




## Mock backend rest API

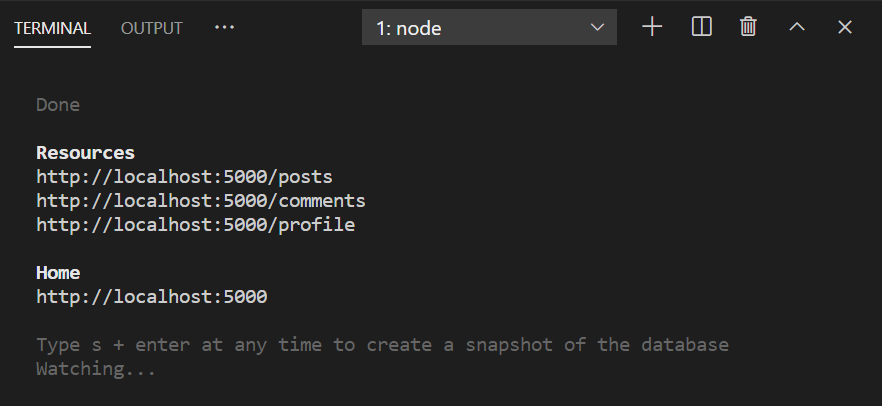
We can now use a simple package to fake an API with json objects. This way we can easily simulate API calls from our front-end.

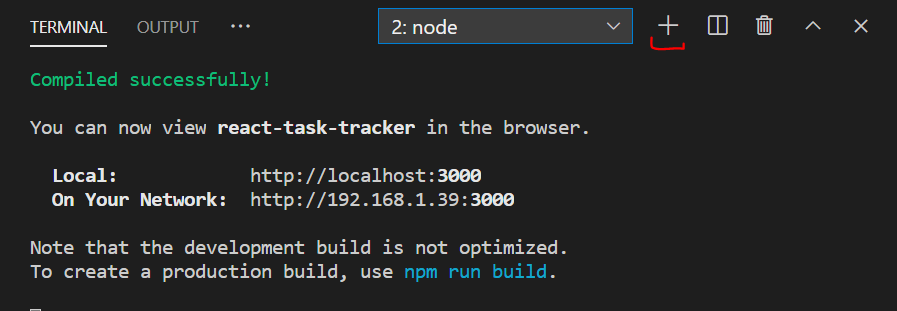
* Install the package by running: ***npm i json-server***
* Create a new script in the package.json file to run the following command: ***json-server - -watch db.json - -port 5000***



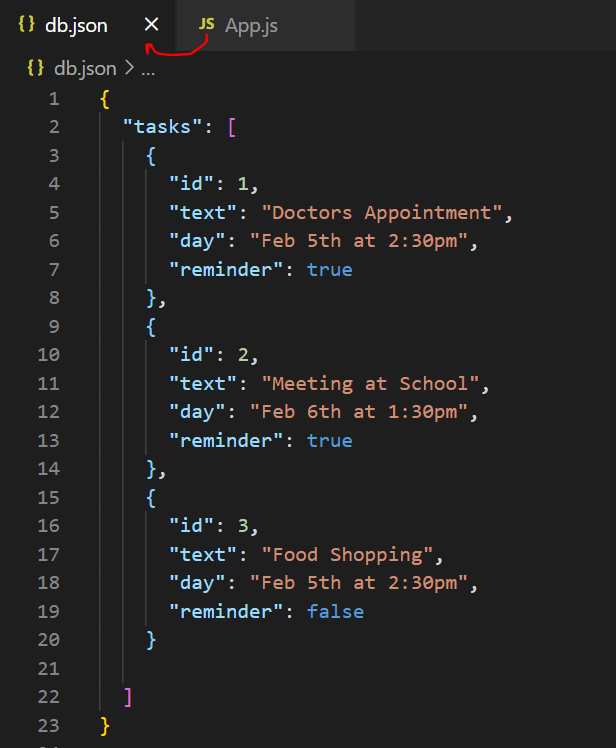
Then we can run the following command to start the mockup server: ***npm run server***

* Open a new terminal to run the react production development environment
* Now we can have both environments running at the same time.

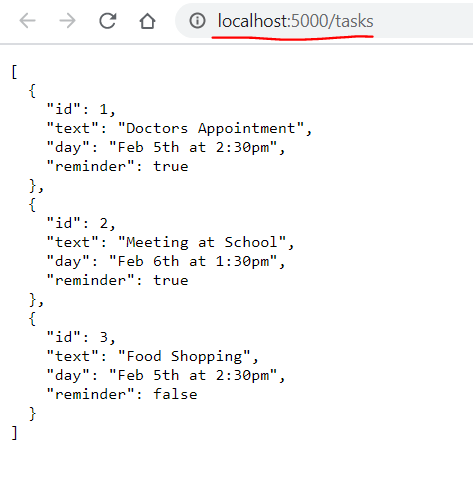




Now we have a new file db.json with some mockup data, we can change it and copy our data:

We can access it now in the mockup server: <http://localhost:5000/tasks>



## Load data from the server

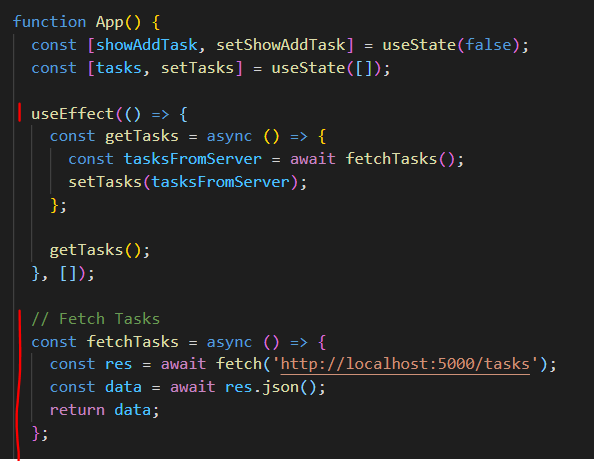
To load data when the page loads we’re going to use a hook called useEffect. This hook takes in an arrow function as a param and it’s called right when the page loads.

Effect documentation: <https://reactjs.org/docs/hooks-effect.html>

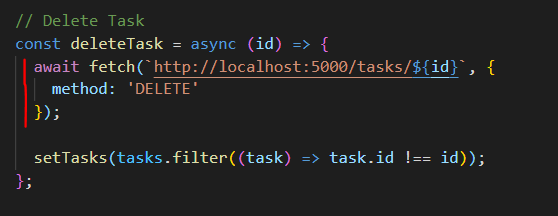
We can declare another function fetchTasks, this time async, so we can wait for the data.

* Await for a response by calling **await fetch**(‘<http://localhost:5000/tasks>’)
* Await for the data to be parsed into a json object by calling **await res.json()**
* Return the data object.

Then inside the useEffect call this function and set the result to the state.



## Delete Tasks from the server



## Add Tasks to the server

