**#1 What is React and How it works - Learn React Series -** [Ruckaiya Mira](https://medium.com/@ruckaiya.awf5?source=post_page-----de7bc2a3284b--------------------------------) Jun 29, 2022

Why do we code in React in simple English.

React JS — Currently the most popular web library. I am launching a *Learn React Series* with this article on Medium.

In this series, I will try to explain the basic things of React JS in simple English language from the beginning.

In this article, I have tried to explain what React is, why we will learn React and the basic concept of React with examples. Hopefully, this article will be useful for any beginner who has a basic idea about JavaScript. Also, if you follow the next articles of this series continuously, I hope you will understand React JS and will be able to work.

React JS, in short, we call it react.

What will be there in this article?

Everything about the react library, step by step from the start till the end, including examples so that you can understand it practically.

According to the Stack Overflow Developer Survey 2020, React’s position and popularity among the most popular web frameworks’ was after JQuery and it was only increasing day by day since then that in 2021’s survey React topped that list.

“What is React?” should not be our first question but “why learn React?”. What is so special about this JavaScript library that according to the 2021 survey, **40.14 percent** of respondents reported to be using React.

Well if we see the official website of React JS, it states that React is a JavaScript library for building user interfaces. Here two things are important 1) JavaScript library and 2) User Interfaces.

Library means a collection of codes that are reused to make our applications. In the same way, React is a collection of vanilla JavaScript codes which we use to make websites, mobile applications and even desktop applications’ user interfaces.

When we talk about an application, there are two parts 1) front-end or user interface 2) back-end (where all the logic is).

Front-end is where the user interacts such as browsing, clicking, submitting forms etc and it is this the front-end that responds to these interactions with the help of the back-end. In the back-end, we basically have the data and there is where complex business logic like authentication, authorization, data manipulation etc happens.

React is not concerned with back-end but front-end, which means its work is to respond to user interactions as well as display and render information.

Now if we talk about the web then we know JavaScript is the web’s default programming language. React is not a programming language but just a JavaScript library, which means React somehow speaks in JavaScript language. It is an open source project, that means if we want we can see its source code and you will see that there is nothing else except pure JavaScript.

That’s why it is important to know Vanilla JavaScript and Modern JavaScript syntaxes to work with React. You don’t need to be a boss in JS in order to learn React but at least need to know the basics.

Now the question is React was introduced in 2013 so before that didn’t we build websites?

Of Course we did using vanilla JS and JQuery. We used to handle user interfaces and user interactions with this two. So what was the need for React when JQuery was popular since the work was getting done. To answer that question we will build a small UI with vanilla JS and discover why React is useful and what extra benefits we get using React.

**Create a folder** on your desktop and inside it create 2 more files — **index.html** and **script.js**.

In index.html

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>React Tutorial</title>  
 <link rel="stylesheet" href="style.css" />  
 </head>  
 <body>  
 <div class="container">  
 <div>  
 <h1 id="display">0</h1>  
 <div>  
 <button id="button">Increment +</button>  
 </div>  
 </div>  
 </div>  
 <script src="./script.js"></script>  
 </body>  
</html>

Our goal is to click on the increment button and the counter number will increase by one.

For that in the **script.js** file, first, we select the button and then in the button’s click event, we run a function that will increase the counter number by one.

To do this:

1. Write a variable name number and set the initial value of the counter number to zero.
2. Then using the query selector method we will select the increment button.
3. Now to listen this button’s click event, use addEventListener and in its callback increase the number by one by writing number++.
4. Next, select the p tag and set its textContent value to the number.
5. Now when you click on the increment button, the counter number increases by one.

In script.js

let number = 0;const display = document.querySelector("#display");  
const button = document.querySelector("#button");button.addEventListener("click", () => {  
 number++;  
 display.textContent = number;  
});

Now the question is what is the problem here when everything is working fine.

Well, the problem is, for larger applications it is not scalable.

For example, at the moment we have **only one** counter. That’s why it doesn’t seem to be a problem but in practical applications, there will be many HTML files and we might need to use similar counters like this in multiple places.

Let’s say, we need another counter here in this example.

What would we have done to add this second counter? We’d have had to take another one of this div in fig:1 and **change the ids** of them. In the JS file, we’d have **needed two number variables** as well as**two buttons and p tags**. Finally, we’d have **needed two addEventListener for two buttons to work**. You see, in order to do the same thing twice, we have to change so many things. Maybe you can maximize it by taking a single function and passing the parameter there but no matter what you do, you have to select these Html elements separately and this can be error-prone. This does not scale well for larger applications and we have to duplicate the same code if more counters are needed and our application will not be managed faster.

For a long time, JavaScript developers had to face this problem as the complexity of the user interface increased and managing the code became difficult.

The same problem was faced by Facebook engineers because a huge website like facebook’s user interface is very complex and was getting very difficult to manage with vanilla JavaScript.

That’s when they started to work on it and in 2011 Facebook’s software engineer Jordan Walke created the first prototype of React JS and named it Fax js. It was first used in 2011 on Facebook’s news feed and on Instagram in 2012.

Jordan was influenced by an HTML component framework XHP-JS of PHP and made react.

In 2013 Fax.js was renamed to React.js and was made open source.

So from then on, react.js started to become popular and big companies started to use react actively in their front-end, including Netflix, Whatsapp, Instagram and many more.

There are currently over 3 million websites that use React world wide.

Now we will see in the case of our example how React solves the problem. We will create the same UI using React instead of vanilla JS.

First, [go to react.js](https://reactjs.org/) website then in the [Docs](https://reactjs.org/docs/getting-started.html) section go to [***Add React to a Website***](https://reactjs.org/docs/add-react-to-a-website.html)***.***There you can see three steps on how we can add react to our website.

1. Create another HTML file in your folder called **react.html.**
2. Add a div with an id root: *inside this div our react application will be generated.*
3. Then add the required script tags at bottom of the HTML file.

**In the script tags you can see two things have been obtained *1) react***and***2) react dom***

react.html

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>React Tutorial</title>  
 <link rel="stylesheet" href="style.css" />  
 </head>  
 <body>  
 <div id="root"></div><!-- Load React. --><!-- Load React. -->  
 <!-- Note: when deploying, replace "development.js" with "production.min.js". -->  
 <script src="<https://unpkg.com/react@18/umd/react.development.js>" crossorigin></script>  
 <script src="<https://unpkg.com/react-dom@18/umd/react-dom.development.js>" crossorigin></script><!-- Load JSX -->  
 <script src="<https://unpkg.com/babel-standalone@6/babel.min.js>"></script><!-- Load our React component. --><script type="text/babel" src="./Increment.js"></script>  
 </body>  
</html>

React library will generate a HTML library for us and React dom library will render it in the browser DOM.

**Where will it generate and render it?**

***In root HTML div block.***

Graphical user interface, application

Description automatically generated

HTML div block with id=”root”

That’s all you need to write in HTML.

Now we will only write JavaScript.

To write JavaScript create a new js file named **increment.js.**

In the HTML file import the increment.js file.

**The increment.js file has access to react because in react.html file the increment.js file is included after react.**

You might be wondering why React and React DOM are separate?

That’s because react is not only made for browsers. React is a platform agnostic library. When we use react library in mobiles we use another library called React Native instead of React DOM and are able to render the UI made by the same react library in mobile interfaces. That’s why the two packages are kept separately.

In the browser, we use React DOM. In fact, 90% of work is done by react, React DOM will only be used for printing. Let's see how.

const domContainer = document.querySelector("#root");ReactDOM.render(<h1> Hello World </h1>, domContainer);

First, we will select the root div element and store it in a **domContainer** variable.

Then **call the render method** of ReactDom and this method receives two parameters.

In the **first parameter**, we have to mention ***what you want to print*** so I gave ‘Hello World’ and in the **second parameter**, we have to mention ***where you want to print***— and for that give domContainer (which is the root div).

React DOM will just render the text ‘Hello World’ in the root div. You can see ‘Hello World’ on the screen. If you inspect this, you can see the text is present in the root div. Here we are only rendering text but you can render anything if you want.

Using the React library we can create useful things plus render them and these are called **React elements**.

React library provides some functions using which we can make React elements, for example **React.createElement()** function.

Let’s create a React element.

*Comment the code you wrote previously*

First create a variable name **myElement** and equate it to **React.createElement()**.

In the **first parameter**, ***we mention what element to create*** so I will go with a div. In the **second parameter**, ***we mention the data that you want to pass*** in the div, at the moment let’s consider no data so we are passing null. In the **third parameter**, ***we mention what content will be there inside the element*** meaning inside the div and I am writing ‘Hello world’. Using React we created a react element and the content inside the element is ‘Hello world’.

Notice we have created a React element but not an HTML element and then instructed the ReactDOM to render **myElement** in root div.

const domContainer = document.querySelector("#root");const myElement = React.createElement("div", null, "Hello React Element");ReactDOM.render(  
 myElement,  
 domContainer  
);

If you inspect and see inside the root div there is another div with the content ‘Hello world’.

Now, what if we want to add a p element inside the div? Then in the third parameter instead of the text, another React element is given as content.

const domContainer = document.querySelector("#root");const myElement = React.createElement("div",   
 null, React.createElement("p", null, "Hello <p> Element"));ReactDOM.render(  
 myElement,  
 domContainer  
);

What if we want to add two p elements inside the div?

Then we **can give an array of elements in the third parameter**. Refer fig:8

const domContainer = document.querySelector("#root");const myElement = React.createElement("div",   
 null,   
 [React.createElement("p", null, "Hello <p> Element"),  
 React.createElement("p", null, "Hello 2nd <p> Element")  
 ]);ReactDOM.render(  
 myElement,  
 domContainer  
);

That’s the job of React Library. It just creates any React element and the ReactDOM receives the React elements which makes them HTML elements for browser DOM, meaning it renders the elements in the browser. What is the benefit to create elements like this with React?

Conversely, the syntax is more complex. Well, forget React, when we write markup in the HTML file then DOM creates HTML elements this way for us for example, I made a p element using document.createElement which has a content of ‘Hello world’ and rendered it in the root div. If we inspect it you can see the root div has the p element with the text ‘Hello world’.

const domContainer = document.querySelector("#root");let p = document.createElement("p");  
p.innerHTML = "Hello p element";  
domContainer.appendChild(p);// const myElement = React.createElement("div",   
// null,   
// [React.createElement("p", null, "Hello <p> Element"),  
// React.createElement("p", null, "Hello 2nd <p> Element")  
// ]);// ReactDOM.render(  
// myElement,  
// domContainer  
// );

This is the way we write it in Html because this syntax is easy for us. Though we write in this way browser parses HTML and creates its DOM by calling createElement.

Html just makes our work easy by giving easy syntax so that we can make our UI faster. In the same way, React makes elements using its**createElement function** and **then creates a separate DOM for itself using these elements**.

*Remember it is a****separate DOM****and in React language we call it Virtual DOM.*

Normally we make HTML elements indirectly by using markup syntax. In the same way React provides us with a markup syntax which we can use to make React elements and the syntax’s name is **JSX** - JavaScript XML, which looks 90% like HTML but it is not HTML.

If we write this HTML using JSX then **we write it in first brackets and store it in a variable** that’s it. This will ultimately call the createElement and make React elements for us.

const domContainer = document.querySelector("#root");const myElement = (  
 <div>  
 <h1 id="display">0 </h1><button id="button">Increment +</button>  
 </div>  
)ReactDOM.render(myElement, domContainer);

The above code is equivalent to the code below.

const domContainer = document.querySelector("#root");const myElement = React.createElement("div", null,   
 React.createElement("h1", {  
 id: "display"  
 },   
 "0 "),   
 React.createElement("button", {  
 id: "button"  
 },   
 "Increment +"));ReactDOM.render(myElement, domContainer);

Thank God we don’t have to write code like this, we write JSX.

Now the question is we are writing the JSX syntax in js file meaning in JavaScript but JavaScript does not understand this syntax neither is it available in ECMAScript specification, then?

Well, that’s why we have to use the transpiler: **Babel.**

Babel will transform this JSX syntax into vanilla JavaScript.

On Babel’s website go to [***Try it out***](https://babeljs.io/repl#?browsers=defaults%2C%20not%20ie%2011%2C%20not%20ie_mob%2011&build=&builtIns=false&corejs=3.21&spec=false&loose=false&code_lz=Q&debug=false&forceAllTransforms=false&shippedProposals=false&circleciRepo=&evaluate=false&fileSize=false&timeTravel=false&sourceType=module&lineWrap=true&presets=env%2Creact%2Cstage-2&prettier=false&targets=&version=7.18.7&externalPlugins=&assumptions=%7B%7D) and paste your JSX code. You can see Babel transpile the code into vanilla JavaScript.

Text

Description automatically generated

Babel transpiles JSX into vanilla JavaScript

That’s why if you’ve noticed I’ve added 3rd script in **react.html**.

Let’s use babel in our code for that. Go to [Docs](https://babeljs.io/docs/en/babel-standalone) and in the Babel script section copy the babel script tag and paste it into the HTML file.

And in our main js file, which we will traspile using Babel, write type=“text/babel” in the script tag.

<!-- Load JSX -->  
 <script src="<https://unpkg.com/babel-standalone@6/babel.min.js>"></script><!-- Load our React component. --><script type="text/babel" src="./Increment.js"></script>

We know by default that the type of js file is text/JavaScript but when we write text/babel the js file goes to babel first then it transpiles the code and ultimately we get JavaScript.

If we see the output of the increment button then you can see it works fine but still we did not get the answer to why we are using React. We will get there soon.

This counter is not a dumb counter. It has functionalities. It should work when clicked so where should we add them?