**Simple React User Guide**

**What is React?**

React -created by Facebook (now Meta)- is a JavaScript library that creates user interfaces. React was originally created for building single page web applications, and making the build process simpler by giving developers the ability to write reusable components (blocks of code that can be used in multiple ways and multiple times). With the growth and development of React, additional packages were added to make it even easier for developers to build applications that can be simple to run, understand, and maintain. For instance, some of these packages, are React-Bootstrap, which is a library that comes with some prebuilt styles, making some CSS styles easier to program, and React-Router, which gives React the ability to have multi-page applications that can be faster and easily navigated without the need for page reload.

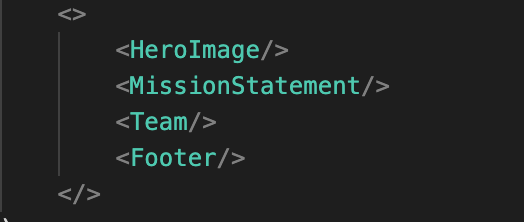
For a full tutorial:

[React Tutorial](https://www.w3schools.com/react/default.asp)

**How does React work?**

React works in declarative code. This means the developer tells the application what to do, not how to do it. Declaring what each element of the application is.

For example:



Declarative code is used by developers to create each component and it’s how the information is displayed. As you see in the code snippet above, each piece of code above is a component being rendered to the virtual DOM. It tells React to render the HeroImage component, the MissionStatement, Team, and Footer components, but it doesn’t say how. It’s up to the developer to figure out the exact order in which to write the code in order for it to render in a specific way.

Now. When looking at the code snippet, what do you notice? If you said HTML-like code or tags, you’re absolutely right. The React library is built on the ES6 Javascript syntax and JSX (this means HTML and JS), as a result, React uses Babel to convert JSX into JavaScript. JSX stands for JavaScript XML, which is like the HTML extension of JavaScript. Babel is a JavaScript compiler that translates markup or programming languages to JavaScript. However, it is possible to use React without ES6 and JSX, which removes the need for Babel, but may make the code base slightly more complicated. It is also possible to use React with other build systems, such as Gatsby or NextJS.

[More on React without ES6 and JSX](https://reactjs.org/docs/react-without-es6.html)

[Different React build systems](https://reactjs.org/docs/create-a-new-react-app.html)

**What are React components?**

We’ve mentioned components quite a bit so far. But what are components?

React components are simply reusable UIs that allow developers to split the application into separate blocks of code. These separate blocks of code act independently from each other, but can interact with each other via props (an arbitrary input with data) to return a React element that declares what is to appear on the screen.

But how exactly do these components work and interact with each other? Each component refers to the state’s representation of specific data. In other words, the state is “the representation - or snapshot - of the app at any one time”. This means that the application will only render data based upon what the state represents at any given time. If the state or props (explained below) change, so does the information rendered.

A component can also have props, which are variables passed to the component by a parent component. Props make the components reusable because they can be modified and receive information from their parent component only. Think of it as a global variable with limits. Whereas state, though also a variable, is directly initialized and managed by the component it’s used in and cannot be accessed or modified outside of its component. Think of it as a local variable.

With the structure of the state and props in the components, we can have more control of the data flow and what gets rendered onto the page. It allows for certain responsibilities to be assigned to a specific component, as well as passed on to a parent or child component only when needed.

For a deeper understanding of how state and props work within a component, visit [State vs Props in React](https://flaviocopes.com/react-state-vs-props/)

**Starting a React application**

Starting your first React application is actually quite simple. You’ll need to have [Node.js](https://nodejs.org/) installed to use the create-react-app command.

On your terminal, change to the directory where you’d like to create your application and then run the create-react-app followed by the name you’d like to give your application (for example purposes, we’ll call it, my-react-app.

Example:



This command will begin running some things on your terminal and installing all the necessary packages to run your React application.

Once your terminal is finished creating the React application, you will likely see a message that says “happy hacking”. At this point, run this command to move into the my-react-app directory:

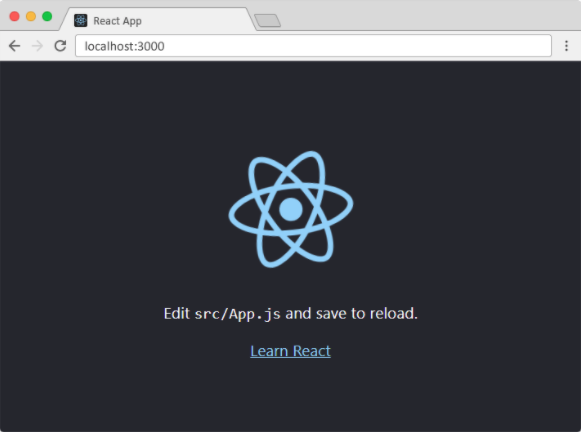


Once you’ve moved into the my-react-app directory, simply run this command to start the server and execute the React application my-react-app:



After running that command, your server will start and a new browser window will open and display your new React App. If that doesn’t automatically happen, you can also just open your browser and type localhost:3000 in the address bar.

At this point you should be seeing this on your browser:



**Class based vs Functional Components**

When working with React, you must choose whether you want to use class based components or functional components.

Class based components are the original method that React was created with. A stateful component controls the changes of the state and the logic implementation. Classes extend the React component and the JSX you want to render will be returned in the return method. Like so:



Functional components are the new modern method that everyone is shifting to. The syntax is different and it is more simplified, there is slightly less code to write and, as the name suggests, it is more function-like, also making it more easily testable; whereas stateful components require much more coding and the classes must be defined early on. Some might consider stateful components easier to understand, as it is more clear where each class is defined and how to refer back to it, in contrast to functional components, where you can’t really immediately see what the class is, where it comes from, or how to callon it. Functional components are a plain JavaScript function that return JSX (no render here). It can also be written with or without the arrow function. Like so:

With arrow function



Without arrow function



Knowing how class components and functional components work is very important to React. It is how React handles specific information; therefore, knowing how and when to use them becomes imperative.

In Class components we initialize the state by giving it one or more variables and then call on those variables with this.state. Before functional components, class components were the only method by which certain React features could be accessed, such as the state and React lifecycle methods. In functional components, though some may disagree, composing the components is actually more straightforward, and gives us the option to implement state and other React features right within the component function. The state is accessed via hooks. The most commonly used hooks are the state hook (useState) and the effect hook (useEffect).

**React Packages**

Now that you know a little bit about React and its powerful use for building great front-end interfaces, let’s talk about some of it’s packages, which essentially make using React even simpler and much more fun.

“Npm packages are a paramount when it comes to developer productivity and happiness.” Many of these packages are incredible time saving solutions to some of the problems we face as developers. Problems like testing, animations, tables/graphs, API calls, etc.

Some of those packages are:

1. [Axios](https://www.npmjs.com/package/axios) - A promise based HTTP client for the browser and Node.js
2. [React-router-dom](https://www.npmjs.com/package/react-router-dom) - Allows you to build a multi-page React application.
3. [React-responsive-carousel](https://www.npmjs.com/package/react-responsive-carousel) - A responsive carousel component for React.
4. [React-awesome-button](https://www.npmjs.com/package/react-awesome-button) - A button component for React.
5. [React-calendar](https://www.npmjs.com/package/react-calendar) - A calendar component for React.
6. [React Testing Library](https://github.com/testing-library/react-testing-library) - A lightweight solution for testing React components.
7. [Framer Motion](https://www.framer.com/docs/) - A production ready motion library for React.

And many more!

In my opinion, it would be extremely difficult, maybe even impossible, to build a full react application without the use of at least one npm package. Many of these packages add that extra oomph the application needs.

For a bigger list of React npm packages, including their use, description, and how-to, please visit.

[A list of Useful npm Packages for React Developers](https://dev.to/manindu/a-list-of-useful-npm-packages-for-react-developers-3dhg)

[14 Useful Packages Every React Developer Should Know](https://betterprogramming.pub/14-useful-packages-every-react-developer-should-know-55b47a325d3)

**More on React**

As the title suggests, this is just a simple guide, designed to give you a brief introduction into React and getting started. For more information and more detailed guides, please visit any or all of the following links:

[React.js Docs - React Tutorial](https://reactjs.org/tutorial/tutorial.html)

[React Tutorial](https://www.w3schools.com/react/default.asp)

[Create a New React App](https://reactjs.org/docs/create-a-new-react-app.html)