Introduction to React

Jack Morris, Will Mikel.

Abstract

Introduction to React includes a brief background on React and why a reader may be interested in learning more about this topic. It includes an explanation of how to get up and running so you can start using React as fast as possible. This is done by explaining restrictions placed upon the syntax such as JSX rules, explaining the folder structure of the development environment, gathering requirements to be able to run React applications, as well as demonstrating the fundamentals of React by using pieces of code that we alter as the introduction progresses. This article aims to get as far as possible in introducing React concepts without sacrificing knowledge needed for advanced concepts. This article prepares the reader to start to learn about more advanced concepts such as hooks, custom hooks, Context API, and React Router.

Introduction

This article is to provide an introduction to React to someone familiar with HTML, CSS, and JavaScript. This is done through introducing some background information on React, what you might need to start working on React, and then going through fundamentals in detail to explain how React works and to prepare for advanced concepts.

Background

What is React?

React is a free open-source front-end JavaScript library that is used to build user interfaces using components. Components are basically independent parts of user interfaces. It is the most popular JavaScript library to build user interfaces. React was created by Jordan Walke, a software engineer at Facebook. It is currently being maintained by Meta (formerly Facebook) and other communities and companies. The first prototype of React was 2011. Reacts initial release date was May 29th, 2013.

How is React different from JavaScript?

Differences between JavaScript and React are that JavaScript apps have an initial user interface created on a server as HTML where as React apps start with a blank HTML page and "dynamically create the initial state in JavaScript"(Framer). React wants you to split your user interface into components, while JavaScript apps can be structured in which ever way you want. Other differences include data for JavaScript apps being stored in the DOM (Document Object Model) and therefore have to be found from the DOM before it can be used. React apps use regular JavaScript variables to store data. When you update the user interface in JavaScript, you have to find the DOM node and then add or remove elements. React automatically does this by updating the user interface "based on setting and changing state within the component"(Framer).

How is React similar and different from other programming languages such as Java?

Technically, React is not a programming language, it is a library for JavaScript. Although Java shares some similarities with JavaScript such as the name and some syntax, JavaScript and Java differ greatly.

Why learn React?

You have probably interacted with a website that uses React, in fact you may even interact with it daily. Facebook, Instagram, and Netflix are built using React where Instagram being just a single web page application that is built completely with React (Vinugayathri). Because React is declarative, which means that you basically just need to tell React what you would like to happen to the state, instead of telling the app how to represent the state, which makes developing with React quicker, easier, and have less potential for human error (Medium). React also has a “write once, run anywhere” philosophy which allows for easier development. It has a massive community with lots of tools. Because of React being component based this allows you to be able to save time and money by building reusable components which allow you to build dynamic user interfaces. Since you already have familiarity with JavaScript, React should be relatively easy to learn.

Setting up the development environment

What you need for the developer environment

One way to get started with this Introduction would be to have relatively recent version of Node.js installed. This can be done by downloading through: https://nodejs.org/en/download/ and selecting your respective operating system. You would then need a terminal application, for this tutorial I will use Git Bash, however the type of terminal application is not significant. After you have installed Node.js, you can check the version by doing `node --version` inside the terminal application. This will confirm that you do have Node.js installed and what version you are on. You will also need a Browser, I will be using Chrome. A text editor is also required, I will be using Visual Studio Code. Lastly, You will need React Developer Tools which can be gotten through the Google Chrome Extension store or your respective browser extension store.

Getting familiar

Throughout the setup, you may be using several commands that include “npm” or “npx”. The “npm” command stands for Node Package Manager, which is the default package manager for Node.js. The “npx” command stands for Node Package eXecute, which is also automatically installed as long as the npm version is above 5.2.0. NPX is a npm package runner that is able to execute any package that you want from the npm registry without installing the package (GeeksforGeeks).

Create the folder that you want to work in and then change your working directory in your terminal to that folder. To get the manifest file you are going to type ‘npm init’ into the terminal application. You can name your package, version, and any other information you want to put. Type “yes” at the end to confirm and now you will have a package.json in that folder. I drag the folder with the package.json file in it and place it into my folder structure for Visual Studio Code. You can access the terminal in some text editors, however for this introduction we will be using the terminal application. We will install Bootstrap, a free and open-source CSS framework to help us and help demonstrate some dependency management. Therefore, typing “npm install bootstrap” into the terminal application will allow us to Bootstrap.

After this command, you will notice a “node_modules” folder, a new package-lock.json file, and if you click on package.json file, you will notice a new dependencies listed for Bootstrap. The node_modules folder contains a list of our dependencies. Once we start using React more, the node_modules folder will get much bigger as more is added to it. If we were to add this folder to Git Hub or similar, we should not share the node_modules folder because it will become very large and you can get these dependencies back easily by doing “npm_install” after you fork or clone this or a similar project to install all dependencies.

Create-React-App and Folder Structure

A better way to learn React is with the command `npx create-react-app my-app` command in your terminal, keep in mind the `my-app` portion is where you are going to name your file, so you should name it something completely lowercase like `my-apptest` or whichever name you prefer. This command provides the latest JavaScript features, a neater presentation, and optimizes the app for production. After you run this command you may notice more folders that may look similar to folder structures of other programming languages. If you look at the public folder, you will notice an index.html file. This is the html file that is displayed as you will see upon further reading. Click on it and find the `<div id=”root”></div>` line. Inside of this root id is where our entire JavaScript application is going to be. The SRC folder is where we are going to do most of our work. The .gitignore folder specifies which files we will ignore by our source control. App.css is where css would go for your application, App.js is where your components would meet. You may have a README file which should just be a markdown file that contains information about your project. All of these names are optional, as it is only by convention that we call it App.js. In this introductory text we are not concerning ourselves with testing files such as app.test.js. A lot of these files are considered boilerplate code in the SRC folder. If you run this code in the terminal application after changing the folder you’re in using `cd yourappfilename` by doing `npm start` you can see a nice single page application in React. To exit the app, and disable the dev server, you should press `CTRL + C`. It should run on localhost:3000 and you should see a spinning atom with a hyperlink and some text. An interesting note here is that if make some changes in the App.js, such as altering the paragraph text to some other text and save it you will notice changes right away with the latest version of your app, unless it has some bugs. If you click on your package.json inside your app file, you should notice a package.json file. This manifest file will list useful information such as the dependencies we are using for this project. It also lists scripts that we can use, such as `start` for `npm start`. There is also `npm run build` which will set up a production ready build. This build folder is just static assets. However, to create an introductory project using React, we will need to delete everything from the SRC folder except for the index.css file and the index.js file for this project. Keep in mind that you still have the index.html file in the public folder.

Explaining fundamentals of React using a small project.

Since this project already assumes that you have knowledge of HTML, CSS, JavaScript, I will leave the reader to implement their own CSS for the project. Import your css that you can work on throughout the project using:

```
import ./index.css;
```

The first thing to do is to go to where our project starts, which is the index.js file you have and type:

```
import React from ‘react’
```

If you want to create a component, you need to import this. The next thing we are going to do is create a function. This function is a component and is special and therefore you must capitalize the name of the function.

```
function Helloworld( ) {

}
```

By capitalizing the function name, React will know that this function is special and it is a component. A big part of React is the ability to use components which are essentially pieces of a user interface. It is possible to use a component with just an HTML element or you can put everything into one component, however putting everything into one component is not recommended as it defeats the purpose of using React. Components allow you to essentially split the user interface into independent, isolated, and reusable pieces. From this component, we want to return what is considered HTML from some people, however technically this is not actually HTML even though it can look like it. It is actually something called JSX, a syntax extension to JavaScript. It stands for JavaScript XML and this allows us to "write HTML in React inside of JavaScript code" (GeeksforGeeks).

```
function Helloworld( ) {
     return <h1>This will be displayed on your webpage</h1>
}
```

To get this component to display on our web page, we are going to have to get this component into our index.html file's <div> tag with the id of root. To be able to do this we are going to have to add another important statement to our code:

```
import ReactDom from 'react-dom'
```

This allows us to use the render method from ReactDom. The render method is looking for what we want to render which is the component we made and it is also looking for where we want to render it. Therefore, we are going to pass in our component and where we want to render it, which is where the root id is.

```
ReactDom.render(<Helloworld/>, document.getElementById('root'));
```

If you have already started your server, you can reload the page and see what you did, if you have not, you can type into the terminal: `npm start` It should display the content between the header tags inside the component. One important thing to mention here is that in React, you must have a closing tag. So for the `<Helloworld/>`, you cannot have `<Helloworld>`, it must be either `<Helloworld/>` or `<Helloworld></Helloworld>`. Also, remember that the component is returning what looks like HTML, but is officially JSX. Currently our component is just a stateless functional component. It always needs to return something, which means it always needs to return JSX. You can see this requirement by removing the return statement.

JSX

An interesting thing to mention about returning what looks like HTML in our return statement, however under the hood it is actually calling functions. For example, our component is actually doing something like this under the hood:

```
const Helloworld = () => {
     return React.createElement('h1', {}, 'under the hood');
};
```

The createElement function is looking for what element we want to return, props, and what is going to rendered inside the element. One of the things it is looking for is props which are properties. What props do is “allow us to pass information to a component” (GeeksforGeeks). Why are we not using this syntax instead of the JSX? Take something like this:

```
<div>
     <h1>
          test
     </h1>
</div>
```

and it’s equivalent using the under the hood functions:

```
React.createElement(‘div’, {}, React.createElement(‘h1’, {},
‘test’));
```

Which one do you think is easier to read? Now imagine if there was even more tags involved in the first example, how do hard to read would that be if it was converted to the under the hood way? JSX allows a much more cleaner and readable approach. There are some rules that you will need to follow with JSX. You must return a single element, which means you could have your entire project in a return statement with tons of tags, however, it must be enclosed by 1 tag and only that enclosing tag should be returned. It is also a good idea to follow the semantics of the document when trying to enclose the tag. It is possible to create a React Fragment by wrapping your returning content by `<React.Fragment></React.Fragment>` or <></> for short. However, we will not use React fragments in this introduction. Another rule is that you should use camelCase for any html attributes, so for example, in html you have the `onsubmit` attribute which will need to be `onSubmit` in JSX. You also are unable to use the class keyword. The reason for this is because there is already a keyword called `class`, therefore you need to use the keyword of `className`. So if you create a `<div>` tag for example with an attribute of class, it will give you an error. You must use the `<div>` tag with an attribute of `className` instead. It will look like this:

```
return (
          <div className=’’>
)
```

For closing elements, you need to close every tag even the ones that are not technically required by HTML5, even the ones that do not have closing tags such as the img tag. This is what closing an img tag looks like:

```
<img src=’’ alt=’’ />
```

You may have noticed that my return statement has parenthesis. You can omit the parenthesis, however it’s not recommended to do so because if you do something such as create a new line after return without the parenthesis and then enter a tag such as `<h2> test </h2>,` React will interpret the return as:

```
return;
```

and the rest of the information will be grayed out. So for this to work, you need to make sure that your opening tag is in the same line as your return. So for example,

```
return
<h2>
     test
</h2>
```

will NOT work.

```
return <h2>
          test
     </h2>
```

will work.

You may find it easier to just have the parenthesis so you never run into this problem. Before we continue onto the rest of the fundamentals, there are some interactions you should be aware of between JSX, CSS, and JavaScript. When you write `{}` inside of JSX, whatever is inside is essentially JavaScript. Also, if you write `{{}}` inside of a JSX, you should know that the outer brackets indicate that whatever is inside is JavaScript. The inner brackets mean there is an object in JavaScript. This means you can have things in your `return` statement like:

```
const name = "John Doe"
const nick = "John Deer"

return (
<h1>{name}</h1>
<h2>{nick}</h2>
);
```

Continuation of fundamentals

Since a big part of React is components, we can do things like nest components inside other React components and have multiple instances of components. Here we are going to make a list of food items and display them with properties using components. You can delete everything except the import statements and the `ReactDom.render` method. Change the first element of the render method to `<FoodList />`.First let us create an array of foods:

```
const foods = [
{
     img:
"https://upload.wikimedia.org/wikipedia/commons/thumb/2/26/Six_eggs_v
iews_from_the_top_on_a_white_background.jpg/1920px-
Six_eggs_views_from_the_top_on_a_white_background.jpg",
name: "egg",
},
{
```

```
img:
“https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/Broccoli_and_cross_section_edit.jpg/330px-Broccoli_and_cross_section_edit.jpg”,
name: “broccoli”,
},
{
img:
“https://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Glass_of_Milk_%2833657535532%29.jpg/1280px-Glass_of_Milk_
%2833657535532%29.jpg”,
name: “milk”,
},
];
```

The img links can be whatever you want, but try to make them respective to the name or create your own. Next we need to create a component with properties of img and name and a component that is the food list.

```
function FoodList() {
     return (
     <div className=”yourfoodlistname”>
     {foods.map((food) => {
     const { img, name } = food;
     return <Food food={food}></Food>;
})}
</div>
);
}

const Food = (props) => {
const { img, name} = props.food;
return (
<div className=”yourfoodclassname”>
     <img src={img} alt=””/>
     <h1>{name}</h1>
</div>
);
};
```

Here you can see that the `FoodList` component is returning the content of our array using a component of `Food`. Its taking every item from our `foods` array and iterating over it in the JSX. The parameter of the map method is `food` which is pointing to each object in our iteration. Then we use destructuring to get variables from `food`. Now we return a `Food` component where we have `food` prop that is equal to a `food` we are passing into it. In the second component we can access the appropriate properties by doing `props.food` and there we have the ability for our properties to be applied the `Food` component which is used in the `FoodList`.

Conclusion

I enjoyed my time learning about React while making an introduction/tutorial about React. I plan on continuing learning about React, but also further practicing HTML, CSS, and JavaScript. I was unable to cover as much as I wanted mostly because of the development environment section and making sure everything was set up correctly, but I think covering the development environment is still important. Despite the word limitation, I still think I was able to provide a good introduction to React for beginners. I think exposing beginners to the create-react-app and going through small little programs through my introduction to demonstrate aspects of React is a manageable and less stressful way since I also am a beginner to React and understand what might confuse beginners before they even start.

References

Lloyd, J.W., *Practical Advantages of Declarative Programming* "declarative language". *FOLDOC*. 17 May 2004. Retrieved 2 December 2021.

React – a JavaScript library for building user interfaces. – A JavaScript library for building user interfaces. (n.d.). Retrieved December 4, 2021, from https://reactjs.org/.

Team. React. (n.d.). Retrieved December 4, 2021, from https://reactjs.org/community/team.html.

React vs. plain javascript. Framer. (n.d.). Retrieved December 4, 2021, from https://www.framer.com/blog/posts/react-vs-vanilla-js/.

Why did we build react? – react blog. – React Blog. (n.d.). Retrieved December 4, 2021, from https://reactjs.org/blog/2013/06/05/why-react.html.

Vinugayathri. (2021, August 24). *Top 10 websites built on react.js*. Top Remote Indian Talent for US Businesses. Retrieved December 4, 2021, from https://www.clariontech.com/blog/top-10-websites-built-on-react.js?hs_amp=true.

Ayub, M. (2019, March 10). *Top 10 reasons why you should learn react right now*. Medium. Retrieved December 4, 2021, from https://medium.com/@SilentHackz/top-10-reasons-why-you-should-learn-react-right-now-f7b0add7ec0d.

JSX full form. GeeksforGeeks. (2021, June 3). Retrieved December 6, 2021, from https://www.geeksforgeeks.org/jsx-full-form/.

ReactJS: Props - set 1. GeeksforGeeks. (2021, January 22). Retrieved December 6, 2021, from https://www.geeksforgeeks.org/reactjs-props-set-1/#:~:text=React%20allows%20us%20to%20pass,of%20global%20variable%20or%20object.

FreeCodeCamp.org (2020, October 6). Full React Course 2020 - Learn Fundamentals, Hooks, Context API, React Router, Custom Hooks [Video]. YouTube https://www.youtube.com/watch?v=4UZrsTqkcW4