## **REACT INTRODUCTION**

### **Introduction**

This section aims to provide you with all you need to know to get started with React, to feel the power of a frontend framework, and to help you create highly scalable React applications. Before we jump into it, one question must be answered.

### **Why React?**

React is one of the most powerful, widely used frontend frameworks.

The landscape for frontend frameworks has been changing a lot over the last few years, so it is understandable to be “scared” of going for the “wrong” one. [This article](https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/) shows the recent development of frontend frameworks well. Once you start diving deeper into a framework, you will begin to love it. It makes your code easily scalable, more readable, and possibly a thousand times more efficient (just our modest estimation).

Just to name a few reasons on why to learn React.js:

* Reusability of components
* Well supported, due to its popularity
* React is unopinionated, which means that it won’t force you to follow any specific patterns or logic, it’s all up to you.
* Smaller learning curve, especially when you already have a good grasp of JavaScript and HTML from our previous lessons.

If you need more convincing, the recent [2020 StackOverflow Developer Survey](https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-all-respondents2) should be reason enough to get excited about learning React. Still unsure? Check out [this article](https://laptrinhx.com/top-10-reasons-why-you-should-learn-react-right-now-3718263995/).

*Convinced and excited? Great, because it’s going to change your life! Let’s dive right into it.*

### **Components**

Applications built with React are made with (reusable) components. Components are your “building blocks” in React. To gain confidence using React, you should learn to divide your application or project into separate components. The following picture gives you an idea of how to do that with a very basic app.



For example, this simple website could be divided into the following sections (components):

* App, which represents your main application and will be the parent of all other components.
* Navbar, which will be the navigation bar
* MainArticle, which will be the component that renders your content.
* NewsletterForm, which is a simple form that let’s a user input his email to receive the weekly newsletter

In React, each component is defined in an ES6 module. ES6 (ECMA2015) introduced the import statement; you can import code from a module into another module. This allows us to write each component in its own module in a separate file and later import all components to the parent component like so: import ExampleComponent from "./components/ExampleComponent". (In our case that parent would be App.js)

In the beginning, it might be a little bit difficult to figure out the best component structure, especially when state and props come into play. This topic will be discussed in the following sections. For now, don’t worry too much about the component structure, understanding of best practices comes with experience. React components, in general, can and usually do have parent or child components. This system of structuring your applications helps to keep your code structured and makes it easy to keep track of your components’ relationships with each other.

To give you an example of a basic component, see the following code:

import React, { Component } from 'react' class App extends Component { constructor() { super() } {/\* Javascript functions can be written here \*/} render() { return ( <div className="App"> Hello World! </div> ) } } export default App

Does the code look foreign? Don’t worry, the structure isn’t as difficult as it may look. Let’s walk through it step by step.

import React, { Component } from "react";

With the above import statement, we are importing React and Component from the React library, which allows us to create a class component. If you are wondering why we have to wrap Component into curly brackets and not React, this is due to the way they are exported from the react module. Default exports are imported without curly brackets; everything else must be wrapped in curly brackets. Don’t worry about this too much as we will get plenty of exposure to import and export statements soon.

class App extends Component { {/\* Some logic we haven't yet talked about. \*/} }

Secondly, we are declaring the class component, which is just a JavaScript class that extends the Component class we imported earlier. One thing to notice is that React components should always be declared with a capital letter at the beginning. This is a naming convention used by most developers and recommended by the React core team at Facebook.

constructor() { super() }

Next is the constructor. A constructor is not obligatory in a class component, but you will most likely encounter one because it becomes important when concepts like inheritance and state are involved. To get used to seeing it, it has been included here. You will usually see developers passing props as an argument to the constructor and also to the super() call, which must be called in any constructor. However, you will not learn about props here. This concept will be discussed further in the next lesson. The idea here is to expose you to the terminology that we will be using in the future.

{/\* Some logic we haven't yet talked about. \*/}

This syntax may look weird at first, but it is nothing more than a simple comment. In React, you write comments within curly brackets and /\* \*/. There are sections of React components where the comments (denoted by //) we are used to are permissible. For now, we’ll leave that as an exercise for you.

render() { return ( <div className="App"> Hello World! </div> ) }

The most unfamiliar code is likely the render() function, which returns something that looks like HTML, but is actually JSX. One of the primary characteristics and features of React is the ability to combine JavaScript and JSX. JSX is an HTML-like syntax that will be transpiled into JavaScript so a browser will be able to process it. One thing you should know about JSX is that you can’t use the HTML protected words, such as class or onchange. Instead of class you write className and instead of onchange you write onChange. In general, all attributes in JSX are written in camelCase. You should be fairly familiar with that naming convention from the naming of variables in JavaScript. The render() function you see is the most used React lifecycle function (more on that in an upcoming section). The only thing you should know for now is that every React class component needs a render function, which returns *one* JSX element. So whatever you want to return needs to be wrapped in a single parent element.

Finally, to be able to reuse this App component we created in other files of our project, we have to export the component. In our example, we export the component as the file’s default export:

export default App;

If you have multiple components in one file, you could export each component separately by adding the export keyword before the declaration of the component. However, if you export a component as a default, you can import it without wrapping curly brackets around it. If you export multiple components, you have to import them inside of curly brackets.

So far, so good! We have already learned a lot about components in React, and let me tell you, class components are just one way of defining components. In React, another way is to use functional components.

A basic functional component looks like this:

import React from "react"; const App = () => { return <div className="App">Hello World!</div>; }; export default App;

As you can see, there are a couple of differences between functional and class based components.

1. We don’t have to import and extend “Component” from React;
2. We don’t need a constructor.
3. We don’t need the render function, but instead a return statement

There are more differences, which we will encounter when discussing props and state.

For further understanding of React components, [this short article](https://dev.to/sarah_chima/an-introduction-to-react-components-cke) provides a great overview.

For further understanding of the difference between functional and class based components, read [this article](https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108). They discuss concepts like state and lifecycle methods, which we haven’t talked about yet. However, keep them in mind because it will prove useful later on in the course. Rest assured, we will expand on “Which one should I use?” later.

### **Create-react-app**

Developers at Facebook came up with a great tool called create-react-app, which sets up a complete React application for you. By running one command, it does all the necessary setup and configuration for you to immediately start working on your project.

If you want to see all the things we have discussed in action, go ahead and run npx create-react-app my-first-react-app in your terminal, cd into the project by typing cd my-first-react-app, and then open it in your text editor of choice. If you want, you can view the project in the browser via the command npm start.

### **Index.js and App.js**

Two of the most important files create-react-app includes for you are index.js and App.js. index.js is the entry point of your application by default. Open up your index.js file and check out this line:

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

In short, this line of code tells React to render the App component into the DOM, exactly into the element with the id “root”. Every create-react-app project has a root div, which is visible in the index.html file in your public directory. If you decide to name your main application component differently than App.js, make sure to change its name in the index.js as well.

If you want to get a better understanding of how create-react-app works and which files it creates for you, make sure to check out [this article](https://blog.logrocket.com/getting-started-with-create-react-app-d93147444a27/) and watch [this video](https://www.youtube.com/watch?v=rUdtgnwrA14) to really understand the file system create-react-app sets up for you.

### **Learning Outcomes**

* Why should you learn React?
* What is JSX?
* What is a React Component?
* What is the difference between a functional and a class based component
* How can you structure your application into components?
* Understanding create-react-app and the files it creates for you

### **Assignment**

1. Read the first couple of pages of the react.js documentation. In general their documentation is a great resource for coming back at a later point if you have to get more familiar with certain concepts or have to freshen up something. So let’s get started. Read [this](https://reactjs.org/docs/hello-world.html), [this](https://reactjs.org/docs/introducing-jsx.html), [this](https://reactjs.org/docs/rendering-elements.html) and [this](https://reactjs.org/docs/components-and-props.html) article. In the last one they already introduce the concept of props. We haven’t looked at that yet, so don’t worry too much about it, but it can’t hurt having a look at it.
2. Watch [this video](https://www.youtube.com/watch?v=JPT3bFIwJYA&list=PL55RiY5tL51oyA8euSROLjMFZbXaV7skS) to get another quick explanation of React. And then watch these ([one](https://www.youtube.com/watch?v=G40iHC-h0c0&list=PL55RiY5tL51oyA8euSROLjMFZbXaV7skS&index=4), [two](https://www.youtube.com/watch?v=9wK4gHoOh1g&list=PL55RiY5tL51oyA8euSROLjMFZbXaV7skS&index=5)) videos from the same series, which focus on components. Feel free to code along with the whole course if you enjoy it.

### **Additional Resources**

* [This video series](https://www.youtube.com/watch?v=QFaFIcGhPoM&list=PLC3y8-rFHvwgg3vaYJgHGnModB54rxOk3&index=1) really provides it all. Watch it for a greater understanding of the most important React concepts.
* [This video](https://www.youtube.com/watch?v=deyxI-6C2u4) shows you how to set up a React application, without using create-react-app