Getting Started React

This section covers prerequisites to get started with React. You should have a good understanding of the following technologies:

- ***HTML***
- ***CSS***
- ***JavaScript***

1. What is React?

React is a JavaScript library for building a reusable user interface(UI). It was initially released on May 29, 2013. The current version is 18.2.0 and somehow it is stable. React was created by Facebook. React makes creating UI components very easy. When we work with React we do not interact directly with the DOM. React has its own way to handle the DOM (Document Object Model) manipulation. React uses its virtual DOM to make new changes and it updates only the element, that needs changing. Do not directly interact with DOM when you build a React Application and leave the DOM manipulation job for the React virtual DOM. In this challenge, we will develop 10-15 web applications using React. A web application, or a website, is made of buttons, links, forms with different input fields, header, footer, sections, articles, texts, images, audios, videos and boxes with different shapes. We use react to make a reusable UI components of a website.

To summarize:

- React was released in May 2013
- React was created by Facebook
- React is a JavaScript library for building user interfaces
- React is used to build single page applications - An application which has only one HTML page.
- React allows us to create reusable UI components
- React latest release is 18.2.0
- React versions

Why we choose to use React? We use it because of the following reasons:

- fast
- modular
- scalable
- flexible
- big community and popular
- open source
- High job opportunity

3. JSX

JSX stands for JavaScript XML. JSX allows us to write HTML elements with JavaScript code. An HTML element has an opening and closing tags, content, and attribute in the opening tag. However, some HTML elements may not have content and a closing tag - they are self-closing elements. To create HTML elements in React we do not use the createElement() instead we just use JSX elements. Therefore, JSX makes it easier to write and add HTML elements in React. JSX will be converted to JavaScript on browser using a transpiler - babel.js. Babel is a library which transpiles JSX to pure JavaScript and latest JavaScript to older version.

4. Rendering a React Element

To render a JSX element to HTML document, we should first create an index HTML. The index.html is the only HTML file you will have in any React Application. That is why we say that every React Application is a single page application. Let us create an index.html file. We can get started with React in two ways - either by using CDN or create-react-app. The create-react-app creates a React project boilerplate outbox and because of that, many people do have a hard time to understand how React works. In order to make things clear for absolute beginners I would like to start with a CDN. We use CDN only in this section and we will use the create-reap-app in the rest of the challenge and I also recommend you to use only create-react-app all the time.

Code:-

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Welcome to React</title>
  </head>

  <body>
    <div class="root"></div>

    <script
      crossorigin
      src="https://unpkg.com/react@16/umd/react.development.js"
    ></script>
    <script
      crossorigin
      src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
    ></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/babel">
      // To get the root element from the HTML document
      const rootElement = document.querySelector('.root')

      // JSX element
      const jsxElement = <h1>I am a JSX element</h1>

      // we render the JSX element using the ReactDOM package
      // ReactDOM has the render method and the render method takes two
arguments
      ReactDOM.render(jsxElement, rootElement)
    </script>
  </body>
</html>
```

I am a JSX element

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Welcome to React</title>
  </head>

  <body>
    <div class="root"></div>

    <script
      crossorigin
      src="https://unpkg.com/react@16/umd/react.development.js"
    ></script>
    <script
      crossorigin
      src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
    ></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script type="text/babel">
      // To get the root element from the HTML document
      const rootElement = document.querySelector('.root')

      // JSX element, header
      const header = (
        <header>
          <h1>Welcome to React</h1>
          <h2>Getting Started React</h2>
          <h3>JavaScript Library</h3>
          <p>Varniraj</p>
          <em>April 3, 1837</em>
        </header>
      )

      // JSX element, main
      const main = (
        <main>
          <p>Prerequisite to get started react.js:</p>
          <ul>
            <li>HTML</li>
            <li>CSS</li>
            <li>JavaScript</li>
          </ul>
        </main>
      )
```

```
      // JSX element, footer
      const footer = (
        <footer>
          <p>Copyright 2022</p>
        </footer>
      )

      // JSX element, app, a container or a parent
      const app = (
        <div>
          {header}
          {main}
          {footer}
        </div>
      )

      // we render the JSX element using the ReactDOM package
      // ReactDOM has the render method and the render method takes two
argument
      ReactDOM.render(app, rootElement)
      // or
      //  ReactDOM.render([header, main, footer], rootElement)
    </script>
  </body>
</html>
```

Welcome to React

Getting Started React

JavaScript Library

Varniraj

April 3, 1837

Prerequisite to get started react.js:

- HTML
- CSS
- JavaScript

Copyright 2022

Components

A React component is a small, reusable code, which is responsible for one part of the application UI. A React application is an aggregation of components. React can help us to build reusable components. The following diagram shows different components. All the components have different border colors. In React we assemble different components together to create an application. We use JavaScript functions or classes to make components. If we use a function, the component will be a functional component, but if we use a class, the component will be a class-based component.

Components can be:

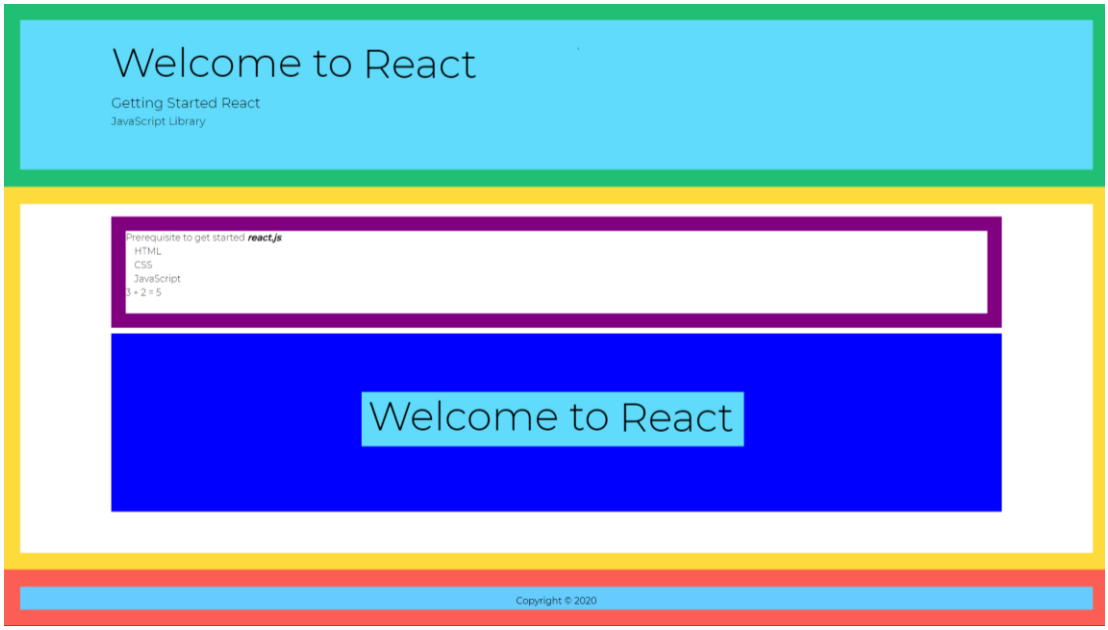
- Functional Component / Presentational Component / Stateless Component / Dumb Component
- Class Component / Container Component/ Statefull Component / Smart Component

The classification of components above does not work for the latest version of React, but it is good to know the former definition and how the previous versions work.

So, let us change all the JSX to components. Components in React are JavaScript functions or classes, that return a JSX. Component name must start with an uppercase, and if the name is two words, it should be CamelCase - a camel with two humps.

Big picture of components

In the previous section we agreed, that a website or an application is made of buttons, forms, texts, media objects, header, section, article and footer. If we have a million-dollar button, we can use this button all the time, instead of recreating it all over again, whenever we need a button. The same goes for input fields, forms, header or footer. That is where the power of the component comes. In the following diagram, the header, main and footer are components. Inside the main there is also a user card component and a text section component. All the different colors represent different components. How many colors do you see? Each color represents a single component. We have five components in this diagram.



JavaScript function

A JavaScript function could be either a regular function or an arrow function. These functions are not exactly the same there is a slight difference between them.

```
const getUserInfo = (firstName, lastName, country, title, skills) => {
  return `${firstName} ${lastName},  a ${title} developer based in ${country}.
He knows ${skills.join(
    ' '
  )} `
}
// When we call this function we need parameters
const skills = ['HTML', 'CSS', 'JS', 'NodeJS', 'MongoDB', 'React']
console.log(
  getUserInfo('Neelkanth', 'Patel', 'India', 'FullStack Developer', skills)
)
```

JavaScript Class

A class is a blueprint of an object. We instantiate a class to create different objects. In addition, we can create children, by inheriting all the methods and properties of the parent.

```
class Parent {
    constructor(firstName, lastName, country, title) {
      // we bind the params with this class object using this keyword
      this.firstName = firstName
      this.lastName = lastName
      this.country = country
      this.title = title
    }
    getPersonInfo() {
      return `${this.firstName} ${this.lastName},  a ${this.title} developer
base in ${this.country} `
    }
    parentMethod() {
      // code goes here
    }
  }

  const p1 = new Parent('Neelkanth', 'Patel', 'India', 'FullStack Developer',
skills)


  class Child extends Parent {
    constructor(firstName, lastName, country, title, skills) {
      super(firstName, lastName, country, title)
      this.skills = skills
      // we bind the child params with the this keyword to this child object
    }
    getSkills() {
      let len = this.skills.length
      return len > 0 ? this.skills.join(' ') : 'No skills found'
    }
    childMethod() {
      // code goes here
    }
  }

  const skills = ['HTML', 'CSS', 'JS', 'NodeJS', 'MongoDB', 'React']

  const child = new Child('Neelkanth', 'Patel', 'India', 'FullStack
Developer', skills)
```