Introduction to React

Introduction

Nowadays, there are many JavaScript libraries and frameworks focused on making Front-end development easier and faster. However, among other great characteristics, ReactJS or React has a business-forward mindset and a strong concept of code reusability. This article will introduce you to the key features of this great JavaScript tool.

Pre Requirements

The requirements to understand and start building your web application with React are familiarity with HTML and JavaScript. It is useful to have a basic knowledge of programming concepts like objects, arrays functions, and classes.

React also uses some features from the ES6 version of JavaScript, such as arrow functions, classes, let, and const statements. So, if you need a review of JavaScript you can check the following link from MDN web docs: A re-introduction to JavaScript (JS tutorial). You can use the Babel REPL to check what ES6 code compiles to.

What is React?

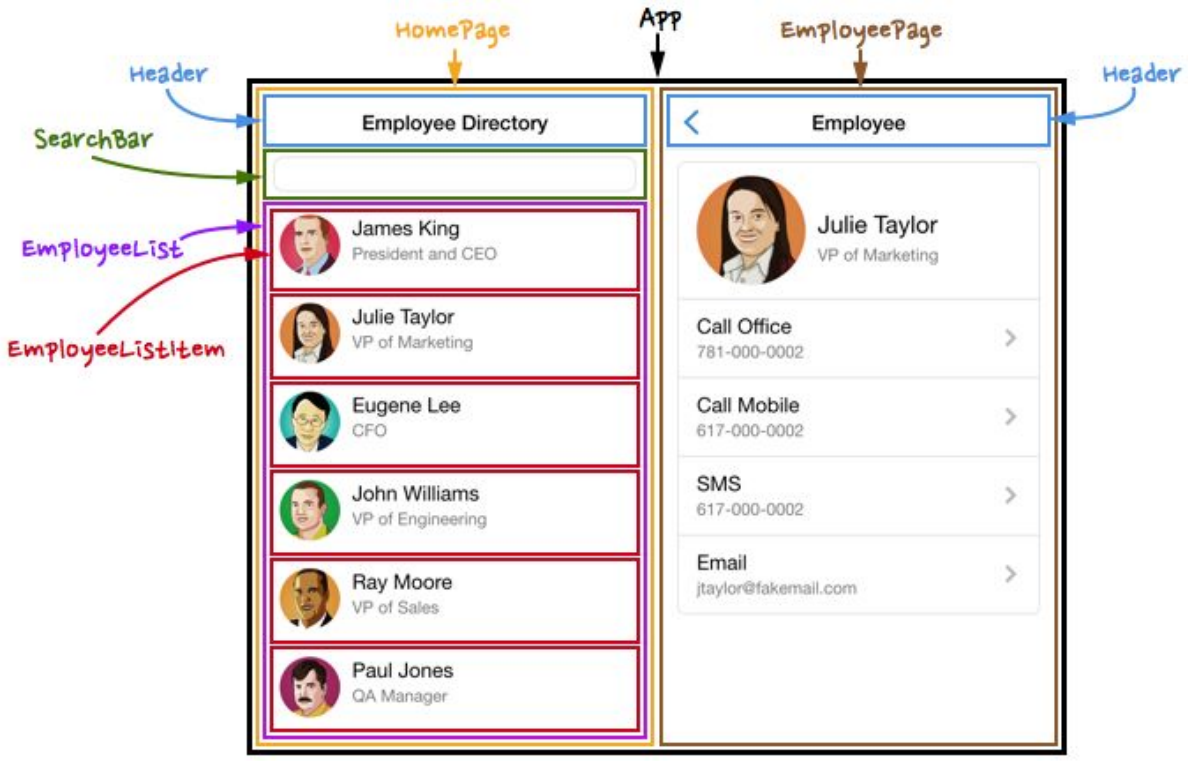
React is a component-based JavaScript library for building user interfaces. It was created by Jordan Walke and it was initially released in May of 2013. Currently, React is maintained by Facebook and an active community of developers and companies. It can be used as a foundation for single-page or mobile applications and it is ideal for reflecting data in real time.

When using React, your pages will not reload and only the necessary parts will be re-rendered. It happens because React has a sophisticated algorithm that is sensitive to changes in the components' state and immediately replicates that in the screen. To reduce the complex procedure of comparing each state attributes, React uses the concept of immutability.

The following topics in this article will discuss those features and other key points that makes React so efficient.

Components

In React, each element should be a component or be part of one. It allows the developer to split the user interface into small reusable pieces and treat them in isolation. The following image is an example of how to structure an employee list in components:



http://coenraets.org/blog/wp-content/uploads/2014/12/uimockscript.png

Each section of the application is a component and each one is responsible to render a specific content. We can consider components as JavaScript functions, where they receive inputs (called “props”) and return React elements specifying what should be displayed on the screen.

There are two types of components: function and class. The code below declares a function component.

Function component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

https://reactjs.org/docs/components-and-props.html

This is a function component because it is literally a JavaScript function, where it accepts a single props object with data and returns a React element. The class component can be seen below:

Class component

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

https://reactjs.org/docs/components-and-props.html

Regarding class components, they require the React.Component extension and the implementation of a method called render, which returns what the component will display. Consequently, class components have more features, where they can alter their state and use Lifecycle Hooks.

The Lifecycle Hooks are very useful functions that are called in certain phases of the component 's life cycle. The most used are the following:

constructor: it is called when the component is initialized. In this phase, It did not render anything yet.

componentDidMount: it is called when the component is rendered to the DOM for the first time.

componentWillUnmount: it is called when the component is removed from the screen (destroyed).

render: it is called everything the component's state change.

State and Props

State and props (abbreviation of properties) are responsible to hold variables. **The state** contains the variables of the component and should be initially declared in the constructor. The following image illustrates the initialization of a state containing one variable called “date”.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

https://reactjs.org/docs/state-and-lifecycle.html

You can use the “this.state.variableName” to access defined variables. In the previous example is displaying the formatted current date in the <h2> tag.

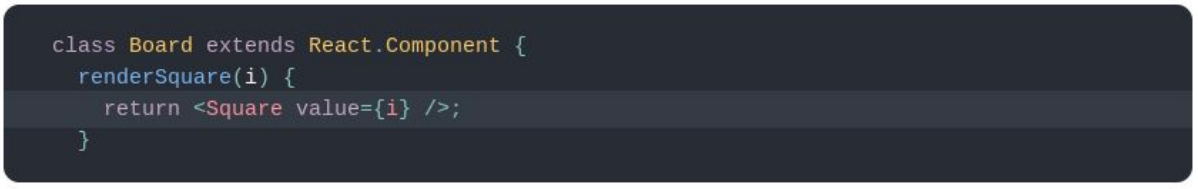
To change the value of the state, it is mandatory to use the **setState** method. The following example is setting a new state inside a method called tick():

https://reactjs.org/docs/state-and-lifecycle.html

It will create an entirely new state, so it is necessary to specify the variables that you want to keep and their values. To know more about how the setState method works, check this link. When React notice that the value of the variable changed, it will call the render() method and it will display the new value in the screen.

Now, to pass information to another component, we need to use the concept of parent and child relationship. When inside a component we call another component, the current component is the parent and the called component is the child. For example, if we take a look at the structure of the Employee List, the EmployeeList component is the parent and the EmployeeListItem is the child in this relationship. A component can have one parent and many child components.

Consequently, the parent component calls a component, specifying the name and value of the variables that will be available inside the child. The following example illustrates a Board component calling a Square component and defining the variable “value”:



https://reactjs.org/tutorial/tutorial.html

Now, to use the variable inside the Square component (child) we need to use **the props**, which holds the variables passed to it by the parent component. The code below demonstrates the Square component using the “value” variable:

```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

https://reactjs.org/tutorial/tutorial.html

So, it is necessary to use “this.props.propertyName” to access the passed properties at the child component. The Square component can also have other child components and the way to pass information to them is the same.

JSX

The JSX(**J**ava**S**cript e**X**tension) is a syntax extension to JavaScript, which allows us to write JavaScript code that looks like HTML. It makes easier to design the output of the render() method. React does not require JSX, but it is very useful when putting markup and logic in the same file. The following example declares a variable called name and uses it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

https://reactjs.org/docs/introducing-jsx.html

After compilation, the expressions made using JSX become regular JavaScript function calls. This means that is possible to use JSX inside if statements, loops, accept it as arguments, assign to variables and return it from functions. The example below demonstrates a function return and JSX expression:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

https://reactjs.org/docs/introducing-jsx.html

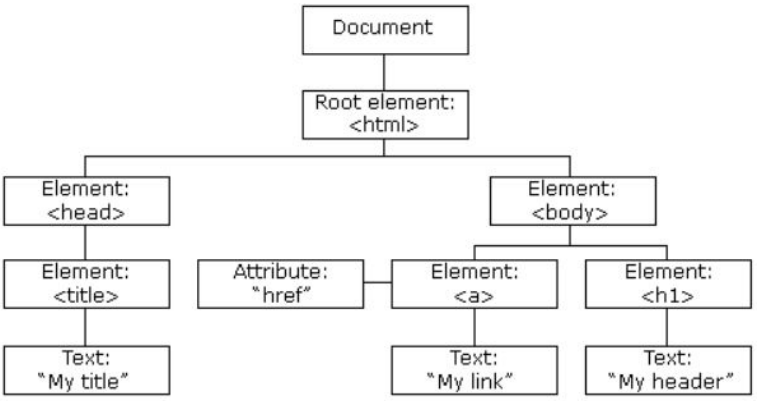
Furthermore, JSX provides protection against injection attacks, by escaping any value before rendering it. The escaping technique replaces all characters of the given input using a specific scheme, allowing only the programs that knows the scheme to interpret the data. The following code demonstrates that is safe to use given information:

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

https://reactjs.org/docs/introducing-jsx.html

Virtual-DOM

In the software web development world, DOM stands for Document Object Model and represents the structure of the elements in a web page. The HTML DOM is constructed as a tree of objects:



https://www.w3schools.com/js/js_htmldom.asp

Some JavaScript libraries, like jQuery, manipulate the DOM elements directly, changing their attributes, adding or removing components. However, instead of changing the DOM directly, React uses a Virtual-DOM, which is a virtual replication of the current DOM tree. Consequently, React can manipulate the Virtual-DOM countless times and comparer its state with the real DOM. This way, React knows exactly which element changed and then updates only that specific element in the screen. This is possible due to the React's **memory reconciliation algorithm**, also known as reconciliation, which compares the real and Virtual-DOM in a heuristic of O(n).

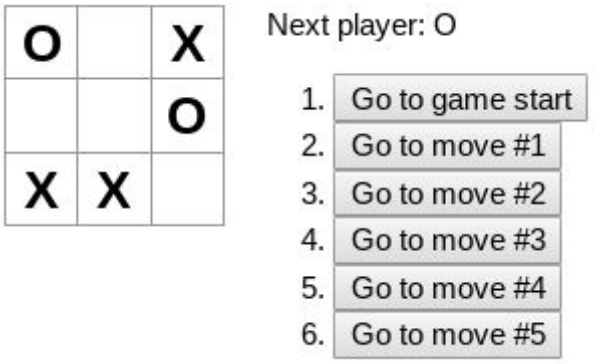
The Virtual-DOM provides the following advantages:

1. Efficiency: It is faster to update the Virtual-DOM instead of the real DOM.
2. Simplicity: React and its Virtual-DOM provide an easier approach to develop reactive JavaScript. The data binding is not attached to the application.

Immutability

Immutability in software development means that a variable can not change its value or state. So, to modify an attribute in a state, it is necessary to replace the state with a new version containing the desired change. As a result, React does not need to implement a complex algorithm to know exactly what changed, it only requires a simple equality operation between the old and new object to see if they are different. This is a very optimized technique because it is not mandatory to verify each attribute of the object. Therefore, it is possible to store different states and compare them.

This approach is very useful when you want to keep track of different phases of a game and load one of them. For example, on each move in a Tic Tac Toe game, we can save the state of the game's board and restore that state when necessary. The following image illustrates a Tic Tac Toe React application with buttons to load previous moves:



https://reactjs.org/tutorial/tutorial.html

Conclusion

To summarize, React is an efficient and easy-to-learn JavaScript library which allows the development of scalable applications focused on the user interface. The component and state concepts help in the designing of the web pages and in the definition of entities relationships. The JSX extension allows us to insert makeup and logic in the same file and helps to develop thinking in the output. Behind this library, there is a complex reconciliation algorithm and a Virtual-DOM feature that makes this tool so efficient. Furthermore, React uses the concept of immutability well, making the comparison between variables much faster. Even if you are not into Front-end development, React is a technology worth to play with.

Resources

- https://reactjs.org/tutorial/tutorial.html
- https://www.angularminds.com/blog/article/7-top-reactjs-features-which-makes-it-best-for-development.html
- https://reactjs.org/docs/introducing-jsx.html
- https://jsx.github.io/
- https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108
- https://flaviocopes.com/react-state-vs-props/
- https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom
- https://medium.com/@hamzamahmood/advantages-of-developing-modern-web-apps-with-react-js-8504c571db71
- https://www.rigelnetworks.com/using-virtual-dom-react-js-top-5-benefits/
- https://reactjs.org/docs/reconciliation.html
- https://reactkungfu.com/2015/08/pros-and-cons-of-using-immutability-with-react-js/
- https://reactjs.org/docs/introducing-jsx.html
- https://blog.logrocket.com/immutability-in-react-ebe55253a1cc