What is React?

Open source javascript library for building dynamic user interfaces

The “V” in MVC

Maintained by Facebook Instagram and a community for individual developers

Some advantages of react:

-design simple declarative views for each state in your application

Encapsulated components

Dynamic properties state

Virtuval dom

Completely independent of the rest of your application’

Can render on the client or server

Virtual dom:

React abstracts away the dom and creates its own version which is simplified and only includes the things that you need

Helps identify which parts have changed

Determines how to upload the browsers dom more efficiently

Much more lightweight/ works faster

JSX javascript syntax extension

A preprocessor step that adds XML syntax to javaScript

1)looks like XML/html

2)defines a familiar syntax for defining tree structures with attributes

3) is not required but makes things much easier

How to create a react app

Create components

Manage state and properties

Handle events

Work with forms and input

Work with jsx

Life cycle methods

Fetch data from an api

React is a library for building compostable user interfaces. It encourages the creation of reusable UI components which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, giving a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow which reduces boilerplate and is easier to reason about than traditional data binding.

React Features

* **JSX** − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
* **Components** − React is all about components. You need to think of everything as a component. This will help you to maintain the code when working on larger scale projects.
* **Unidirectional data flow and Flux** − React implements one way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
* **License** − React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

## React Advantages

* React uses virtual DOM which is JavaScript object. This will improve apps performance since JavaScript virtual DOM is faster than the regular DOM.
* React can be used on client and server side.
* Component and Data patterns improve readability which helps to maintain larger apps.
* React can be used with other frameworks.

## React Limitations

* React only covers view layer of the app so you still need to choose other technologies to get a complete tooling set for development.
* React is using inline templating and JSX. This can seem awkward to some developers.

React uses JSX for templating instead of regular JavaScript. It is not necessary to use it, but there are some pros that comes with it.

* JSX is faster because it performs optimization while compiling code to JavaScript.
* It is also type-safe and most of the errors can be caught during compilation.
* JSX makes it easier and faster to write templates if you are familiar with HTML.

React uses JSX for templating instead of regular JavaScript. It is not necessary to use it, but there are some pros that comes with it.

* JSX is faster because it performs optimization while compiling code to JavaScript.
* It is also type-safe and most of the errors can be caught during compilation.
* JSX makes it easier and faster to write templates if you are familiar with HTML.

## Using JSX

JSX looks like regular HTML in most cases. We already used it in environment setup tutorial. Look at the code from **App.jsx** where we are returning **div**.

### **App.jsx**

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

Hello World!!!

</div>

);

}

}

export default App;

## Nested Elements

If you want to return more elements, you need to wrap it with one container element. Notice how we are using **div** as a wrapper for **h1**, **h2** and **p** elements.

### **App.jsx**

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

<h2>Content</h2>

<p>This is the content!!!</p>

</div>

);

}

}

export default App;

## Attributes

You can use your own custom attributes in addition to regular HTML properties and attributes. When you want to add custom attribute, you need to use **data-** prefix. In example below we added **data-myattribute** as an attribute of **p** element.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

<h2>Content</h2>

<p data-myattribute = "somevalue">This is the content!!!</p>

</div>

);

}

}

export default App;

## JavaScript Expressions

JavaScript expressions can be used inside of JSX. You just need to wrap it with curly brackets **{}**. Example below will render **2**.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>{1+1}</h1>

</div>

);

}

}

export default App;

## Styling

React recommends using inline styles. When you want to set inline styles, you need to use **camelCase** syntax. React will also automatically append **px** after the number value on specific elements. You can see below how to add **myStyle** inline to **h1** element.

import React from 'react';

class App extends React.Component {

render() {

var myStyle = {

fontSize: 100,

color: '#FF0000'

}

return (

<div>

<h1 style = {myStyle}>Header</h1>

</div>

);

}

}

export default App;

## Comments

When writing comments you need to put curly brackets **{}** when you want to write comment within children section of a tag. It is good practice to always use **{}** when writing comments since you want to be consistent when writing the app.

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

{//End of the line Comment...}

{/\*Multi line comment...\*/}

</div>

);

}

}

export default App;

React js components

## Stateless Example

Our first component in example below is **App**. This component is owner of **Header** and **Content**. We are creating **Header** and **Content** separately and just adding it inside JSX tree in our **App** component. Only **App** component needs to be exported.

### **App.jsx**

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<Header/>

<Content/>

</div>

);

}

}

class Header extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

</div>

);

}

}

class Content extends React.Component {

render() {

return (

<div>

<h2>Content</h2>

<p>The content text!!!</p>

</div>

);

}

}

export default App;

Reactjs- state:

State is the place where the data comes from. You should always try to make your state as simple as possible and minimize number of stateful components. If you have, for example, ten components that need data from the state, you should create one container component that will keep the state for all of them.

## Using Props

Code sample below shows how to create stateful component using EcmaScript2016 syntax.

1st component creation:

Index.js:

**import** React **from "react"**;  
**import** { **render** } **from "react-dom"**;

**import** {Header} **from "./components/Header"**;  
**import**{ Home } **from "./components/Home"**;

**class** App **extends** React.*Component*{  
 render() {  
 **return**(  
  
 <**div className="contanier"**>  
 <**div className="row"**>  
 <**div className="col-xs-10 col-xs-offset-1"**>  
 <**h1**>Hello!</**h1**>  
 </**div**>  
 </**div**>  
 </**div**>  
 );  
 }  
}  
  
**render**(<**App**/>,**window**.**document**.getElementById(**"app"**));

Index.html

<!doctype **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="utf-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico"**>  
  
 <**title**>New app</**title**>  
 </**head**>  
 <**body**>  
 <**div id="app"**></**div**>  
  
  
  
 </**body**>  
</**html**>

Adding new components from other pages

Header.js

**import** React **from "react"**;  
  
**export class** Header **extends** React.*Component*{  
 render(){  
 **return**(  
 <**nav className="navbar navbar-default"**>  
 <**div className="container"**>  
 <**div className="navbar-header"**>  
 <**ul className="nav navbar-nav"**>  
 <**li**><**a href="#"**>home</**a**></**li**>  
 </**ul**>  
 </**div**>  
 </**div**>  
 </**nav**>  
 );  
  
 }  
}

Home.js

**import** React **from "react"**;  
  
**export class** Home **extends** React.*Component*{  
 render(){  
 **return**(  
 <**div**>  
 <**p**>in a new component</**p**>  
 </**div**>  
 );  
 }  
}

Real time practice:

* 1. Npm rum build:- this will concatenate all html css and js and create a folder called build directory then we will upload to a server to serve project
  2. Push-server build push code to server

Npm install react-bootstrap –save

React native is a library  
---------  
If you want to work with ReactJS, you need to have solid knowledge of JavaScript, HTML5 and CSS. Even though ReactJS  
doesn't use HTML, the JSX is similar so your HTML knowledge will be very helpful.  
--------  
One way data-binding:  
React is a library for building composable user interfaces. It encourages the creation of reusable UI components  
which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you,  
giving a simpler programming model and better performance. React can also render on the server using Node,  
and it can power native apps using React Native. React implements one-way reactive data flow which reduces  
boilerplate and is easier to reason about than traditional data binding  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
React Features  
JSX − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.  
  
Components − React is all about components. You need to think of everything as a component.  
This will help you to maintain the code when working on larger scale projects.  
  
Unidirectional data flow and Flux − React implements one way data flow which makes it easy to reason about your app.  
Flux is a pattern that helps keeping your data unidirectional.  
  
--------------------------  
React Advantages  
React uses virtual DOM which is JavaScript object. This will improve apps performance since JavaScript virtual DOM is faster than the regular DOM.  
  
React can be used on client and server side.  
  
Component and Data patterns improve readability which helps to maintain larger apps.  
  
React can be used with other frameworks.  
-----------------------------  
JSX is faster because it performs optimization while compiling code to JavaScript.  
It is also type-safe and most of the errors can be caught during compilation.  
JSX makes it easier and faster to write templates if you are familiar with HTML.  
-----------------------------  
State  
tate is the place where the data comes from. You should always try to make your state as simple as possible and minimize  
number of stateful components.  
\_-----------------------------------  
ReactJS - Props  
The main difference between state and props is that props are immutable. This is why container component should define  
 state that can be updated and changed, while the child components should only pass data from the state using props.  
--------------------------------------  
ReactJS - Component Life Cycle  
  
componentWillMount: is executed before rendering, on both server and client side.  
  
componentDidMount is executed after first render only on the client side. This is where AJAX requests and DOM or state  
updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with  
delayed execution like setTimeout or setInterval. We are using it to update the state so we can trigger the other  
lifecycle methods.  
  
componentwillupdate(just before rendering)-->componentWillmount(excute before render)  
-->componentwillreceivepros-->should componentupdate-->componentdidupdate-->componentdidmount  
-->componentwillunmount  
  
-------------  
  
componentwillmount --- immediately before intial rendering  
componentdidmount -- immediately after intial rendering  
componetwillreceiveprops -- when component receives new props  
shouldcomponentupdate -- before rendering after receiving new props or state  
componentwillupdate -- before rendering after receiving props or state  
componentdidupdate --after components updates are flushed to dom  
componentwillunmount immediately before removing component from dom  
------------------------------