# **React** is a JavaScript library for building user interfaces

## What is React?

- React is a JavaScript library – one of the most popular ones, with **over 100,000 stars on GitHub**.
- React is not a framework (unlike Angular, which is more opinionated).
- React is an open-source project created by Facebook.
- React is used to build user interfaces (UI) on the front end.
- React is the **view** layer of an MVC application (Model View Controller)

One of the most important aspects of React is the fact that you can create **components**, which are like custom, reusable HTML elements, to quickly and efficiently build user interfaces. React also streamlines how data is stored and handled, using **state** and **props**.

# Why React?

## Convenient architecture

Flux – is highly competitive to MVC. One-way data flow provides maintainability and efficient arrangement of data and DOM elements.

## Virtual DOM

React developers suggested using “virtual DOM” in order to solve performance issue for websites with too dynamic DOM. All changes in a document are made there first, and then React looks for the shortest path to apply them in a real DOM tree. This approach makes the framework fast.

## Components

React is fundamentally different than other front-end frameworks in that each asset is made up of many isolated components. Want a button changed across the whole platform? Change it once and voilà it`s changed everywhere.

By making the creation, distribution and consumption of isolated reusable components more straightforward, developers are better able to save time by using and creating common abstractions. This is true of both low level elements like buttons and high level elements such as accordions.

## JSX

React.js uses a special syntax called JSX, which allows to mix HTML with Javascript. Markup and code are composed in the same file. This means code completion gives you a hand as you type references to your component’s functions and variables.

## SEO friendly

React is significantly more SEO friendly than most JavaScript MVC frameworks. As it is based on a virtual DOM you can use it on the server without needing a headless browser on the server such as Phantom.js to render pages to search engine bots.

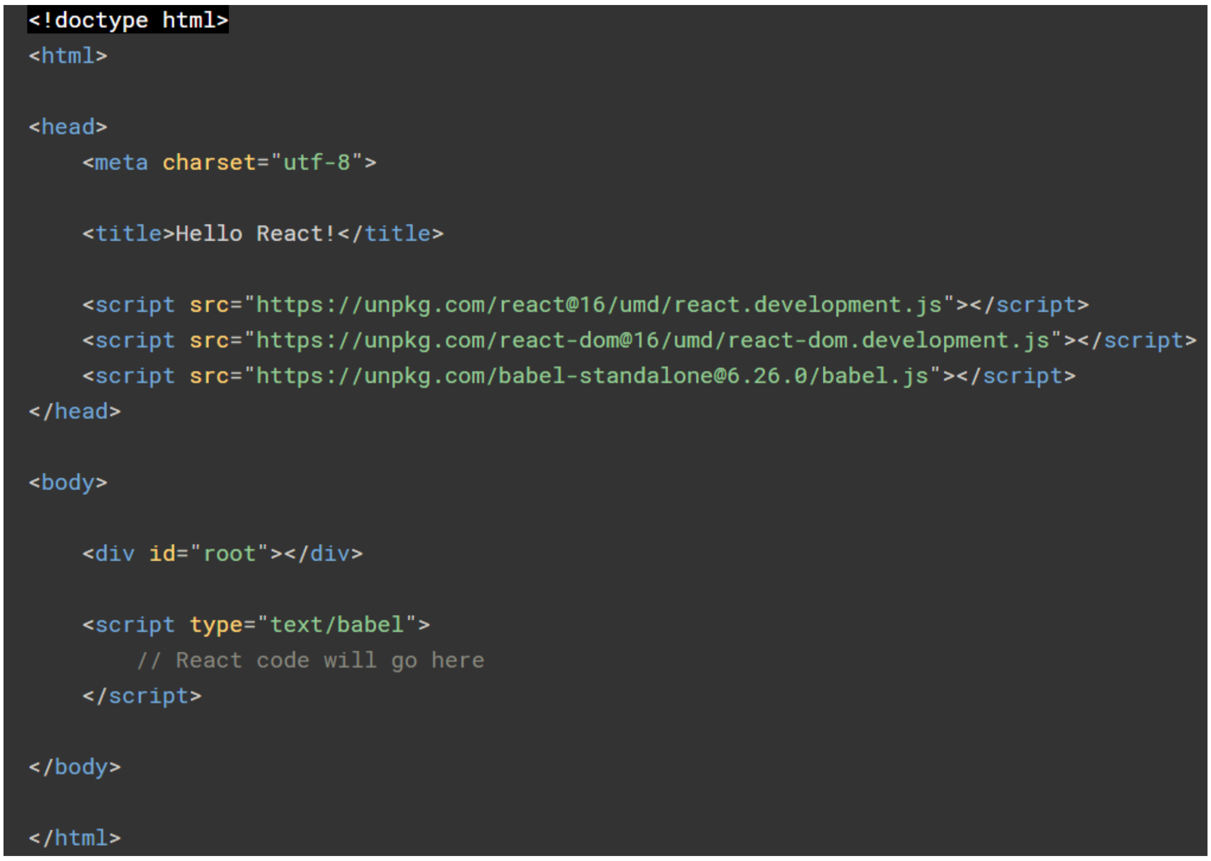
# Setup and Installation

There are a few ways to set up React:

## Static HTML File

This first method is not a popular way to set up React and is not how we'll be doing the rest of our tutorial, but it will be familiar and easy to understand if you've ever used a library like jQuery, and it's the least scary way to get started if you're not familiar with Webpack, Babel, and Node.js.

Let's start by making a basic index.html file. We're going to load in three CDNs in the head – React, React DOM, and Babel. We're also going to make a div with an id called root, and finally we'll create a script tag where your custom code will live.



```html
<!doctype html>
<html>

<head>
    <meta charset="utf-8">

    <title>Hello React!</title>

    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
</head>

<body>

    <div id="root"></div>

    <script type="text/babel">
        // React code will go here
    </script>

</body>

</html>
```

- React – the React top level API
- React DOM – adds DOM-specific methods
- Babel – a JavaScript compiler that lets us use ES6+ in old browsers

The entry point for our app will be the root div element, which is named by convention. You'll also notice the text/babel script type, which is mandatory for using Babel.

Now, let's write our first code block of React. We're going to use ES6 classes to create a React component called App.

Now we'll add the `render()` method, the only required method in a class component, which is used to render DOM nodes.

```
class App extends React.Component {
    render() {
        return (
            <div>
                <h1>Hello World!</h1>
            </div>
        );
    }
}
```

Inside the return, we're going to put what looks like a simple HTML element. Note that we're not returning a string here, so don't use quotes around the element. This is called JSX.

Finally, we're going to use the React DOM `render()` method to render the App class we created into the **root** div in our HTML.

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

Now if you view your index.html in the browser, you'll see the `h1` tag we created rendered to the DOM.

# Create React App

The method I just used of loading JavaScript libraries into a static HTML page and rendering the React and Babel on the fly is not very efficient, and is hard to maintain.

Fortunately, Facebook has created Create React App, an environment that comes pre-configured with everything you need to build a React app. It will create a live development server, use Webpack to automatically compile React, JSX, and ES6, auto-prefix CSS files, and use ESLint to test and warn about mistakes in the code.

To set up create-react-app, run the following code in your terminal, one directory up from where you want the project to live. Make sure you have 5.2 or higher in Node.js.

```
npm install -g create-react-app
```

Once that finishes installing, move to the newly created directory and start the project.

```
cd react-demo
npm start
```

Once you run this command, a new window will popup at localhost:3000 with your new React app.

If you look into the project structure, you'll see a /public and /src directory, along with the regular node_modules, .gitignore, README.md, and package.json.

In /public, our important file is index.html, which is very similar to the static index.html file we made earlier – just a root div. This time, no libraries or scripts are being loaded in. The /src directory will contain all our React code.

To see how the environment automatically compiles and updates your React code, find the line that looks like this in /src/App.js:

## React Developer Tools

There is an extension called React Developer Tools that will make your life much easier when working with React. Download **React DevTools for Chrome**, or whatever browser you prefer to work on.

After you install it, when you open **DevTools**, you’ll see a tab for React. Click on it, and you’ll be able to inspect components as they’re written. You can still go to the Elements tab to see the actual DOM output. It may not seem like that much of a deal now, but as the app gets more complicated, it will become increasingly necessary to use.

# Virtual DOM in ReactJS

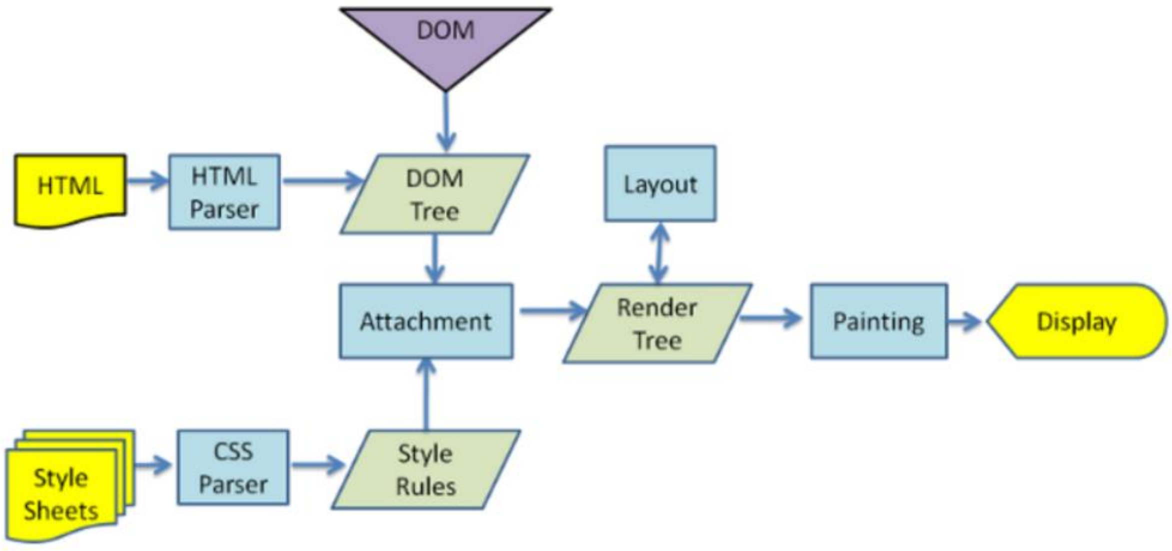
ReactJS does not update the Real DOM directly but it updates the Virtual DOM.

This causes a great performance benefit for ReactJS. In this article we will see why updating the Real DOM is slow, what is Virtaul DOM, and how updating Virtual DOM increase the performance?

## Why updating Real DOM is slow:

Updating a DOM is not slow, it is just like updating any JavaScript object; then what exactly makes updating Real DOM slow?

Let's look at the below image from html5Rocks to see how exactly browser renders a web page



Rendering engines which is responsible for displaying or rendering the webpage on the browser screen parses the HTML page to create DOM. It also parses the CSS and applies the CSS to the HTML creating a render tree, this process is called as attachment.

Layout process give exact co-ordinates to each node of the render tree, where the node gets painted and displayed.

So when we do,

```
document.getElementById('elementId').innerHTML = "New Value"
```

Following thing happens:

1. Browser have to parses the HTML
2. It removes the child element of elementId
3. Updates the DOM with the “New Value”
4. Re-calculate the CSS for the parent and child
5. Update the layout i.e. each elements exact co-ordinates on the screen
6. Traverse the render tree and paint it on the browser display

Recalculating the CSS and changed layouts uses complex algorithm and they effect the performance.

Thus updating a Real DOM does not involves just updating the DOM but, it involves a lot of other process.

Also, each of the above steps runs for each update of the real DOM i.e. if we update the Real DOM 10 times each of the above step will repeat 10 times. This is why updating Real DOM is slow.

## What is virtual DOM?

Virtual DOM is in-memory representation of Real DOM. It is lightweight JavaScript object which is copy of Real DOM.

Updating virtual DOM in ReactJS is faster because ReactJS uses

1. Efficient diff algorithm
2. Batched update operations
3. Efficient update of sub tree only
4. Uses observable instead of dirty checking to detect change

AngularJS uses dirty checking to find the models which has changed. This dirty checking process runs in cycle after a specified time. As the application grows, checking the whole model reduces the performance and thus makes the application slow.

ReactJS uses observable’s to find the modified components. Whenever setState() method is called on any component, ReactJS makes that component dirty and re-renders it.

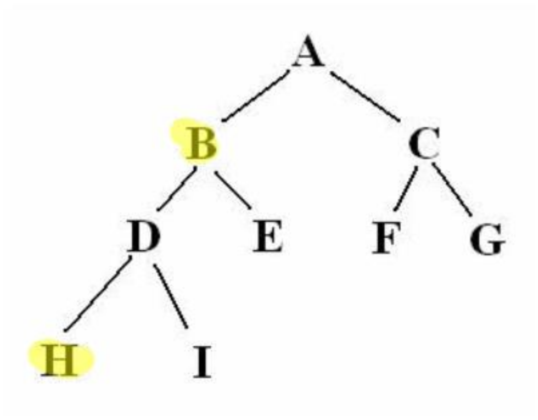
Whenever setState() method is called, ReactJS creates the whole Virtual DOM from scratch. Creating a whole tree is very fast so it does not affect the performance. At any given time, ReactJS maintains two virtual DOM, one with the updated state Virtual DOM and other with the previous state Virtual DOM.

ReactJS using diff algorithm compares both the Virtual DOM to find the minimum number of steps to update the Real DOM.

Finding minimum number of modifications between two trees have complexity in the order of O(n^3). But react uses heuristic approach with some assumptions which makes the problems to have complexity in the order of O(n).

ReactJS uses following steps to find the difference in both the Virtual DOM's

- **Re-render all the children if parent state has changed**. If the state of a component has changed, then ReactJS re-renders all the child components even if child components are not modified. To prevent the unwanted re-render of the child components we can use `shouldComponentUpdate()` component life cycle method. This will further help in boosting the performance.
- **Breadth First Search**. ReactJS traverse the tree using BST. Consider the below tree. States of element B and H have changed. So when using BST ReactJS reached element B it will by default re-render the element H. This is the reason to use BST for tree traversal



**3. Reconciliation.** It is the process to determine which parts of the Real DOM need to be updated. It follow below steps:

- Two elements of different types will produce different trees.
- The developer can hint at which child elements may be stable across different renders with a key prop.

ReactJS using the diff algorithm to find the minimum number of steps to update the Real DOM. Once it has these steps, it executes all the steps in one event loop without involving the steps to repaint the Real DOM. Thus, if there are more element which gets updated

ReactJS will wait for the event loop to finish then, in bulk will updated the real DOM with all the updated elements.

Once all the steps are executed, React will repaint the Real DOM. This means during the event loop, there is exactly one time when the Real DOM is being painted. Thus all the layout process will run only on time for updating the real DOM.

# JSX: JavaScript + XML

We’ve been using what looks like HTML in our React code, but it’s not quite HTML. This is **JSX**, which stands for JavaScript XML.

With JSX, we can write what looks like HTML, and also we can create and use our own XML-like tags. Here’s what JSX looks like assigned to a variable.

```
const heading = <h1 className="site-heading">Hello, React</h1>;
```

Using JSX is not mandatory for writing React. Under the hood, it’s running `createElement`, which takes the tag, object containing the properties, and children of the component and renders the same information. The below code will have the same output as the JSX above.

```
const heading = React.createElement('h1',
      {className: 'site-heading'},
      'Hello, React!');
```

JSX is actually closer to JavaScript, not HTML, so there are a few key differences to note when writing it.

- `className` is used instead of class for adding CSS classes, as class is a reserved keyword in JavaScript.
- Properties and methods in JSX are camelCase – onclick will become `onClick`.
- Self-closing tags *must* end in a slash – e.g. `<img />`

JavaScript expressions can also be embedded inside JSX using curly braces, including variables, functions, and properties.

JSX is easier to write and understand than creating and appending many elements in vanilla JavaScript, and is one of the reasons people love React so much.

## Components

Almost everything in React consists of components, which can be **class components** or **simple components**.

Most React apps have many small components, and everything loads into the main App component. Components also often get their own file.

A class component must include render(), and the return can only **return** one parent element.

## Props

One of the big deals about React is how it handles data, and it does so with properties, referred to as **props**, and with state.

Props are an effective way to pass existing data to a React component, however the component cannot change the props – they're read-only.

## State

With props, we have a one-way data flow, but with state we can update private data from a component.

You can think of state as any data that should be saved and modified without necessarily being added to a database – for example, adding and removing items from a shopping cart before confirming your purchase.

The object will contain properties for everything you want to store in the state.

```
state = {
        characters: []
     };
```

To retrieve the state, we'll get `this.state.characters` using the same ES6 method as before. To update the state, we'll use `this.setState(),` a built-in method for manipulating state.

You must use `this.setState()` to modify an array. Simply applying a new value to `this.state.property` will not work.

## Submitting Form Data

If we wanted to be able to add new data to state.

In a real world application, you'd more likely start with empty state and add to it, such as with a to-do list or a shopping cart.

## Keys

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

## Refs

Refs provide a way to access DOM nodes or React elements created in the render method.

**When to Use Refs**

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Avoid using refs for anything that can be done declaratively.
For example, instead of exposing open() and close() methods on a Dialog component, pass an isOpen prop to it.

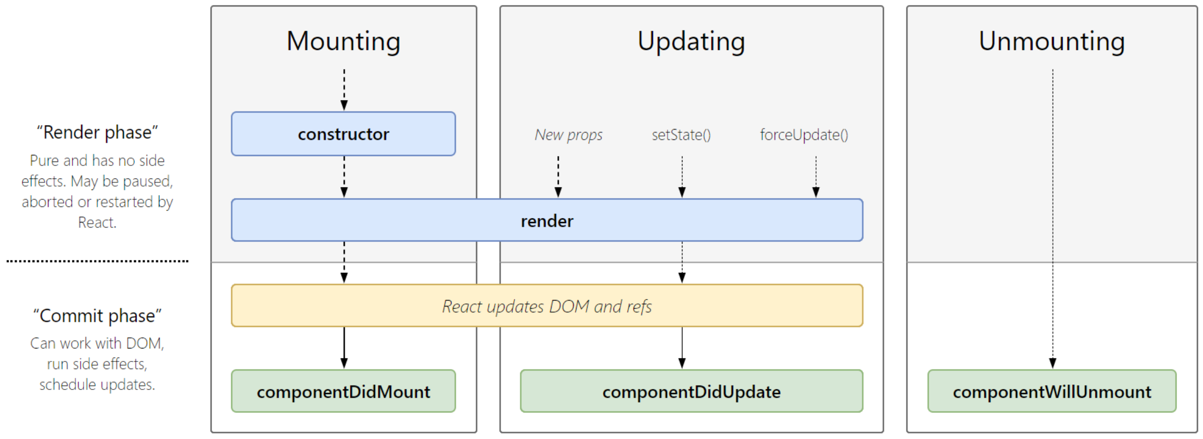
# Adding Lifecycle Methods

We can declare special methods on the component class to run some code when a component mounts and unmounts:

These methods are called "lifecycle hooks"

## Commonly Used Lifecycle Methods

The methods in this section cover the vast majority of use cases you'll encounter creating React components. **For a visual reference, check out this lifecycle diagram.**



## render()

The render() method is the only required method in a class component.
When called, it should examine this.props and this.state and return one of the following types:

- **React elements.** Typically created via JSX. For example, **<div /> and <MyComponent />** are React elements that instruct React to render a DOM node, or another user-defined component, respectively.
- **Arrays and fragments.** Let you return multiple elements from render. See the documentation on fragments for more details.
- **Portals**. Let you render children into a different DOM subtree. See the documentation on portals for more details.
- **String and numbers.** These are rendered as text nodes in the DOM.

- **Booleans or null**. Render nothing. (Mostly exists to support return test && <Child />pattern, where test is boolean.)

The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

If you need to interact with the browser, perform your work in **componentDidMount()** or the other lifecycle methods instead. Keeping **render()** pure makes components easier to think about.

## Note

render() will not be invoked if shouldComponentUpdate() returns false.

## constructor()

**If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.**
The constructor for a React component is called before it is mounted. When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs.

Typically, in React constructors are only used for two purposes:

- Initializing local state by assigning an object to this.state.
- Binding event handler methods to an instance.

You **should not call** setState() in the constructor(). Instead, if your component needs to use local state, **assign the initial state to** this.state directly in the constructor:

Constructor is the only place where you should assign this.state directly. In all other methods, you need to use this.setState() instead.

## componentDidMount()

componentDidMount() is invoked immediately after a component is mounted (inserted into the tree). Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.
This method is a good place to set up any subscriptions. If you do that, don't forget to unsubscribe in componentWillUnmount().
You **may call** setState() **immediately** in componentDidMount(). It will trigger an extra rendering, but it will happen before the browser updates the screen. This guarantees that even though the render() will be called twice in this case, the user won't see the intermediate state. Use this pattern with caution because it often causes performance issues. In most cases, you should be able to assign the initial state in
the constructor() instead. It can, however, be necessary for cases like modals and tooltips when you need to measure a DOM node before rendering something that depends on its size or position.

## componentDidUpdate()

componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.

Use this as an opportunity to operate on the DOM when the component has been updated. This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed).

## componentWillUnmount()

componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created
in componentDidMount().

You should not call `setState()` in `componentWillUnmount`() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again.

# React Router :

The easiest way to get started with a React web project is with a tool called Create React App, a Facebook project with a ton of community help.

First install create-react-app if you don’t already have it, and then make a new project with it.

React Router DOM is published to npm so you can install it with either npm or yarn. Create React App uses yarn, so that’s what we’ll use.

```
npm install react-router-dom
```

## Building and Deploying a React App

Everything we’ve done so far has been in a development environment. We’ve been compiling, hot-reloading, and updating on the fly. For production, we’re going to want to have static files loading in – none of the source code. We can do this by making a build and deploying it.

Now, if you just want to compile all the React code and place it in the root of a directory somewhere, all you need to do is run the following line:

```
npm run build
```

This will create a build folder which will contain your app. Put the contents of that folder anywhere, and you’re done!