JavaScript and ECMAScript

JavaScript's real name is ECMAScript — ECMA-TC39 is the standards body that controls it. So the version numbers of "JavaScript" are ES1, ES2 … ES6 etc which stands for ECMAScript Version 6 etc. Version 6 came out in 2015, and they decided to not have the same numbering version anymore. They decided to call ES6 "ES2015" instead. Since then the version numbers have been based on year. But lots of people still refer to them as ES7, ES8 etc.

DOM

Stands for "Document Object Model". It's an API which allows JavaScript to manipulate the items on a website. HTML is a language for describing the content on a website. When the browser interprets HTML, they create the DOM from it and when we look at a website we're really looking at the instructions that the DOM is telling the browser to make, not so much the HTML. So, when something is changing on the website, it's actually the DOM that changed and JavaScript is the thing that tells the DOM to change (sometimes called DOM manipulation). In browsers, JavaScript is the only programming language that can manipulate the DOM. So, if you want a complex web application with dynamic moving parts and user interactions, you're going to need JavaScript (or a JavaScript library/framework) that can manipulate the DOM.

Node

Around 2009, developers created a "run-time" for JavaScript on the server. A "run-time" is just a technical term for "the ability to parse and execute code". That runtime is called Node. So, Node is essentially JavaScript that runs on the server.

NPM

Stands for "Node Package Manager" and is the de facto standard JavaScript and Node dependency manager.

Dependency (Third-party Dependency)

Any third-party code that your application depends on. If you have a React application, you're going to install React (which is a dependency of your App) with NPM

Module

A JavaScript file that is re-usable in an application. Third-part dependencies are modules but the application code your developers write will also be in modules.

MVC

A paradigm for organizing framework code. It stands for Model, View, Controller — its three logical categories for framework organization.

Reactive Programming

Reactive Programming is a paradigm for building applications. Its premise is that as one thing changes, other things will "react" to those changes. The library React is

created with this paradigm which is why the library's name is React. Reactive programming can be seen as an alternative to older paradigms like MVC.

Monolithic Frameworks

A monolithic framework is one that tries to give the developer all the tools they'll need as one big unit. The tradeoff to that vs giving them a more à la carte approach is that there's a cohesiveness to monolithic approaches but less wiggle room to customize things, where as the more à la carte approach requires the developer to use lots of third-party libraries together but this is highly customizable.

Website vs Web Application (Web App)

Generally speaking, we all know what a website is. It's a collection of pages organized into URLs that we can go to and read content. Technically, a Web Application is no different. You could say that a Web App is a *type* of Website. But we needed a way to differentiate the types of sites that were starting to be built around 2008 which were much much lager and more sophisticated than that of a normal "website". Things like Twitter, Facebook, LinkedIn, Google Docs, Dropbox, Linked and many more were starting to feel less website-like and more application-like. Not only were they big, they were actually doing technical things to make their sites *feel* more like an application, including doing this idea called a Single Page Application. Thus, we started calling these types of sites "Web Applications". They're not "Native Apps" like the ones you install from the App Store on your phone, they're still visited in the browser, but they *feel* like applications.

Single Page Application (SPA)

A technique for building Web Applications so that the browser doesn't do page refreshes in-between page visits but rather uses JavaScript to swap parts of the DOM into and out of the page to give the user a feeling like a page changed. However, since the page didn't actually change, it's one single page that they're viewing the entire time they're on the Web App.

Angular

A JavaScript, DOM manipulation framework which uses a monolithic approach and is written in TypeScript (A superset of JavaScript). Angular is made by Google and it would be considered an alternative to React.

Ember

A JavaScript, DOM manipulation framework which uses a monolithic approach. Made to resemble "Ruby on Rails" - a server side language and framework. It would be considered an alternative to React.

React

A JavaScript, DOM manipulation library that uses a component oriented architecture with state management tools built in. It's also made with the "reactive" paradigm. Made by Facebook.

Vue
A JavaScript, DOM manipulation library made with the "reactive" paradigm. It would be considered an alternative to React.

Svelt
A JavaScript, DOM manipulation library made with the "reactive" paradigm. It would be considered an alternative to React.

Build-tools, Build-process (aka "tooling")
Any tool(s) or processes that are used (usually third-party) to help the developers while they're developing code and are a part of the build-process — the process of taking the code we write as developers and preparing it for production servers. Webpack and Babel are examples of build-tools.

Webpack
A "build-tool" that takes JavaScript modules (files) and turns them into one "build-file" aka a "bundle". The end-result build-file is what gets sent to the production servers.

Code-Splitting
Instead of having a building tool like Webpack generate one giant build file, code-splitting is the idea to configure the app so that Webpack creates separate smaller build files based on some logic.

Babel
A JavaScript compiler that can turn alternative types of JavaScript into normal JavaScript. Or it can turn JavaScript that is too modern for most browsers into an older version of JavaScript. Babel can be a plugin to Webpack and can be used to turn React's special JSX syntax into normal JavaScript. Babel is also used to turn TypeScript into JavaScript.

CDN
Stands for "Content Delivery Network" which is just a way of saying you're importing third-party code into your site via <script> or <link> includes. This is an alternative to using a builder like Webpack where the third-party code would be installed with NPM and then "bundled-into" the project.

TypeScript

A superset of JavaScript that brings “types” to JavaScript. What this translates to is, JavaScript is not a “typed” language. Some developers who like typed languages (like Java and C#) might not like JavaScript because it’s not “typed”. TypeScript is the exact same thing as JavaScript but with types. Made by Microsoft.

Server Side Rendering (SSR)

Ordinarily, React is only rendered in the client (the browser). But React can also be rendered on the Server with Node. In this case we call that Server Side Rendering

Static Site Generator

A framework or tool that allows you to build a website in a way that feels dynamic but through a build-process generates a static website.

Create React App (CRA)

A code generator that gives developers a head-start for their Webpack and Babel setups for React. It does not give the developer a way to organize React or any SSR tools.

NextJS

A React Framework which creates a hybrid of an SPA and SSR. The SSR is by virtue of dynamic pages. It has a prescribed way to organize code which has some limitations regarding layouts

Gastby

A React Framework which creates a hybrid of an SPA and SSR. The SSR is by virtue of its Static Site Generator. It has a prescribed way to organize code which has some limitations regarding layouts

Remix

A React Framework which creates a hybrid of an SPA and SSR. The SSR is by virtue of dynamic pages. It has a prescribed way to organize code which rely’s on React Router which doesn’t have the same limitations as NextJS and Gatsby. It also does code-splitting out-of-the-box

React Router

A third-part module for React that helps React become a Single Page Application.

Component

Originally, the term component meant a re-usable piece of UI such as a Calendar Component or a Button Component. When web developers look at a website, we tend to see it in terms of a big collection of components all being used together. React is a JavaScript library to manipulate the DOM, but it does so by creating a system where you defined your code into these logical groups that they call “components”. In React,

components are the fundamental building blocks for everything — kind of like small lego bricks that can be used to make anything. Sometimes you might have a React component that perfectly aligns with the traditional idea of what it meant to be a component, like a React Button or Calendar. But since components are the fundamental building blocks for all things in React, we actually use them to build lots of other stuff that someone like a graphic designer would never consider to be a "component" in their line of work. In React, a component is a piece of re-usable code.

Function Component

A component made from a JavaScript function. Before "Hooks", Function Components were inferior to Class ones because they couldn't have state

Class Component

A component made from a JavaScript class. Before "Hooks", Class Components were the only way to have state and lifecycle methods for components.

Lifecycle Methods

Methods of Class Components which help the component do things at certain intervals of time, like when the component first mounts, updates, or unmounts.

Higher Order Components (HoC's)

This is not a third type of React Component, but rather a design pattern for creating re-usable code abstractions with Class Components. This pattern was mostly popular in the 2015-2017 era in React. Redux largely helped to popularize this pattern.

Render Props

This is a pattern for creating re-usable code abstractions. It's an alternative to HoC's and starting around 2017 started to be considered a better alternative by many because it solved several shortcomings that HoC's had. However, Render Props have some shortcomings of their own that HoC's didn't have to begin with.

Hooks

In October of 2018, the React team revealed a new way to write React components. A "hook" is a feature for the Function Components (that we've had for a while already) to have access to the full API of React just like how Class Components have always had. This means that Function Components are no longer inferior to Class ones. Hooks solve a number of problems that Class components had to begin with and they give us a new primitive way to do code sharing abstractions so we don't have to rely on the HoC and "Render Props" patterns which had shortcomings. Generally speaking, when a project is using Function Components with Hooks instead of Class Components, the code in each component will be smaller, more bug free (solving some common bugs found in classes) and the abstractions are much cleaner than HoC and Render Props.

It's also worth mentioning that Class-based Components are not deprecated and a project with Class-based ones can slowly migrate one component at a time to Function Components with Hooks since they can co-exist.

A "Pure" or "Memoized" Component

These are not other "types" of components. A "Pure" component is a Class Component that has some optimizations turned on. A "memoized" component is a Function Component with similar optimizations turned on.

Component Hierarchy

In React, Components are organized into a hierarchy of parent and child components which ends up creating somewhat of a "tree-structure" similar to a family tree diagram. Components can pass each other information via "props" or "context"

Props

In React, a "prop" (short for property) is a basic way to pass information from parent to child components. When props are sent down through many levels of components, this can sometimes be referred to as "prop-drilling". Prop-drilling isn't supposed to be a good thing or a bad thing, but just a term that describes what's happening. Sometimes developers think prop drilling might be getting excessive and they might prefer to change to "context" for the particular situation.

Context

In React, props are probably the easiest way to pass information around. But in a large application with many components, it can be a little cumbersome with lots of "prop-drilling". So "context" is an alternative way to pass information around where you can pass information from one component to another all while skipping all the components in-between. Technically, components can't pass data directly between each other except for parents passing props to children, so context works by creating a PubSub pattern (publish-subscribe patter) whereby the publishers are called "providers" and the subscribers are called "consumers"

A Context Provider is a component sending data out. A Context Consumer is a component that is "subscribed" to that provider and receiving the data.

State

In React, state means information that can possibly change. If a component loads some user information, we would store that in "state". If we have some piece of UI that toggles on and off or has multiple "UI states" like a dropdown menu being open or closed, then we store that data about the toggle or menu in React's "state". For the most part, React's State is similar to what a "variable" is in programming but it also has some nuanced details that are React-specific.

Local State

When a component has state which originates inside of it, we call that “local state” — as in, local to that component.

When Component A has local state and needs to share that information with Component B, Component A could pass that information via Props or Context to component B.

Application State (Global State)

When your application needs data to be shared among all components, we might call that application state. Redux is a third-party tool which in the past has been used as a way to do Application State in React. More recently, React has built-in features that help us create application state which we call Context.

Redux and MobX

Redux and MobX are third-party tools for doing application state. They are framework agnostic so they’re not meant just for React.

JSX

Stands for “JavaScript and XML”. It’s a syntax that React created in order to help developers author DOM manipulation instructions. JSX looks very similar to HTML which developers are already familiar with. Since browsers don’t understand JSX though, developers use a build-process with Babel to compile the JSX code into ordinary JavaScript. So the files we write as developers have JSX in them, but the code that gets sent to production does not. JSX is largely credited as being the number one aspect of React that has made it so successful.

Mounting vs Rendering

In React, we say that a component is “mounted” if the component is actively being used to create the DOM. In fact, we would say the component is “mounted to the DOM”. In order to mount, the component has to be “rendered” for the first time. After this first “render and mount” a component can experience a number of “re-renders” which is just a way to get updated instructions from the component do to some sort of change that occurred in the code.

Reconciliation

When a React component renders (or re-renders), it’s instructions for creating or updating DOM are examined by React so that React can optimize the actual DOM changes for performance.

Virtual DOM

React keeps track in memory of what it thinks the real DOM should look like, this “copy” of the DOM that React keeps is called the Virtual DOM (virtual because it’s not the real one). When React is doing reconciliation, it uses its Virtual DOM and compares

it to the changes being sent over from the components optimize what needs to be changed in the real DOM - this is reconciliation.

Effects (Side Effects)

As a general programming term, “side effects” is when a function reaches outside of itself to accomplish its task. In React, we say a component has “side effects” if the component reaches outside of itself (usually outside of JavaScript) to accomplish a task. Some examples would be: Doing Network API requests to communicate with the server, working with Cookies or Local Storage, etc.

Asynchronous Code

This is a technique for writing code such that a long-running task can be happening outside of JavaScript and allowing JavaScript to continue to run and not put everything on hold while it waits for the task to finish.

Promises and “Async/Await”

A “promise” is a JavaScript pattern that helps to do asynchronous code. “Async/Await” is a more modern syntax for doing promises.

JavaScript Threads

JavaScript does not have threads. I just put this in here so that we’re clear about this. Instead, JavaScript solves the concept of “doing many things at once” with asynchronous code.

AJAX, XHR, and Fetch

AJAX is really just unofficial “slang” for what is really an XHR Request. XHR is the tool in browsers that’s existed since the early 2000’s that allows JavaScript and browsers to communicate over the network with servers to get responses without doing full-page refreshes. XHR is asynchronous and was created before promises. Because of the XHR API being so old, spec creators wanted to make a more modern alternative to XHR based on promises called Fetch. Today, developers still use XHR and/or Fetch to build complex “application-like” websites that don’t have to do refreshes — like Single Page Applications. There’s also a bunch of third-party libraries that wrap around these technologies. One that is popular is Axios which is a promise-based abstraction for XHR with other value-added concepts.