This is probably a more simple explanation than you're looking for, but for anyone else who may be confused...

***AJAX and Airplanes***

Think about an **Airplane**. The most important part of an airplane is that it flies. But an airplane also has **wheels**. And the wheels serve a very important purpose, because without them the airplane would never fly or land, and despite all the awesome stuff a plane could do in the air, it wouldn't matter without wheels.

This is the same relationship that **React** has with **AJAX**. React is the airplane, and AJAX are the wheels. But, ya know, other things have wheels too. Tractors, cars, even some boats have wheels, and they're all very important, and crippled without wheels. So too is AJAX to other web technologies, but when you're talking about **airplanes**, its **wheels** are usually the farthest thing from your mind.

So **React** is to **AJAX**, what an **Airplane** is to **Wheels**.

But let's talk about AJAX. What is it? Why is it so important? How it is used in websites today. Then I'll show how it's used by React. Then show you what React does that's so impressive, it makes you forget about AJAX - Like an Airplane to its wheels.

***Remember Websites in the 90's?***

When you clicked anything, a new page would have to load to show the effect of your click - even if it was nothing. [Here's an awesome example](http://toastytech.com/evil/). Go to that page and click around... See how clicks whisk you away to a completely different page? That is the Internet before AJAX.

Now, take a look at *this very page*: next to each answer is an Up Arrow... Go ahead and click one of them... Notice the page doesn't reload, but you are given feedback: *the arrow turns Orange*. This may seem insignificant, but it represents big advancements in web technology: AJAX.

AJAX allows that to happen! And this is no big deal now, it's so intrinsic to the web experience now, it's difficult to imagine an Internet without it.

***AJAX and a Clock Face***

A good analogy of AJAX, and how it changed the web is a simple wrist watch, or a wall clock... Imagine the minute, hour and second hands moving around the clock's face to show time. Now, suppose every movement of the second hand caused the entire clock to be destroyed and rebuilt?

*All that effort* of destroying and rebuilding just to show a tiny change. Well, that would be an outrageous waste of resources, and that was the Internet of the 90's. Thankfully, we have AJAX now. Just as a clock seamlessly displays the time, AJAX allows webpages to show changes in data immediately, without the page needing to be refreshed; you click an up arrow, and it turns orange. No page reload needed!

Everything nowadays uses AJAX, and as a developer it's becoming easier to write an app without even knowing what AJAX is. This is because it has become assimilated into the JavaScript technologies we use, like React - which uses [fetch](https://facebook.github.io/react-native/docs/network.html), among other commands, to make AJAX easier and more readable.

And AJAX works just like webpages:

1. A client (A Web Browser like Firefox) requests data from a server (Like the Stack Overflow (SO) Server).
2. The server processes the request.
3. The server sends a response back to the client.
4. and the code of the webpage, based on the response, decides where it will go (HTML) and how it will look (CSS).

All that is hidden from the user so it just seems like the webpage is reacting to their actions.

***Single-Page Applications***

Since we're not rebuilding the entire page with every click, you can keep stuff stored in the browser. This can be used throughout your entire visit and future visits to the site.

The first time you visit Stack Overflow, all of the CSS, JS, and HTML is loaded. These three languages define the style (CSS), behavior (JS), and structure (HTML) of the data sent back and forth from the server. And guess how that data is sent! AJAX.

This is how most of the web works now. Google, Facebook, Amazon, Youtube, Reddit, every site built with WordPress and WIX, even Stack Overflow - they all use this basic paradigm for delivering their sites to users efficiently. The difference comes in how the Single-page application is built and managed...

***React.js***

[React](https://facebook.github.io/react/) is a javascript library for building and maintaining [Single-Page Applications](https://flaviocopes.com/single-page-application/).

But that's not even that big of a deal. The big deal about React is how it allows you to build applications...

Basically, you to build things separately, then put them together: Components come together to form an Application. So take a look at this totally plausible but fake code for all the answers on this page:

answerArray.map(a => <Answer answerData={a}></Answer>)

This is one line that shows most of the information on this page. That is a big deal. The developers at Stack Overflow created their own component, called "Answer" and it's only job is to show an Answer. You run that in a loop, and bam, all the intricacies of all the answers is ***abstracted***, hidden in the Answer Component, which is completely separate from other components.

Now take a look at this:

<App>

<Header />

<LeftSidebar />

<Question>

{ answerArray.map(a => <Answer answerData={a} /> )}

</Question>

<RightSidebar />

<Footer/>

</App>

This is the whole Stack Overflow site.

Each tag (Header, Question, Answer, etc.) is a component. These components are completely separate and have self-contained code, but here they are used together to build the more complex application.

***Composition***

An important concept of React is composition, and we just defined it above. "Composition allows you to build more complex functionality by combining small and focused functions" ([flaviocopes](https://flaviocopes.com/react-composition/)). Our Application is composed of smaller components.

It's also important to note that each component contains its own functionality. That means if the user clicks a button and a warning appears, the button and the code that makes the warning appear are in the same component.

***Functional Programming***

Surprise, we already defined this too. Functional programming, for our purposes, means 1. objects; and 2. how they behave; are in the same place. Like the button example above. Click a button, get a warning. And that's all in the same file.

This is different than pre-React development where all the buttons would be in one file, and all effects of the buttons would be in another. And this isn't necessarily a wrong way to do things, but for web development, it is easier to think in terms of self-contained building blocks, rather than widely dispersed tools that don't work by themselves.

***Why you shouldn't care about Moustache and Handlebars***

These two technologies have been cannibalized by React. Similar to how React uses AJAX, but makes it easier, Moustache and Handlebars is already inside React, and you'll using them all the time without even knowing it. And to me, that's ok. There are arguments to the contrary, and knowledge is never a bad thing, so investigate further if you want, but this is already long enough, so that's all I'll say about that.

Instead I will tell you about 3 technologies you should care about.

***What you should care about instead...***

**Node**

The main point of Node.js is that it executes JavaScript outside a browser. Big whoop, right? Well, it turns out this is one of the most influential advancements for web developers ever. In fact, downloading Node is often done before downloading React.

Node is important for 2 huge reasons:

1. It lets you download other stuff
2. It lets you process JavaScript before sending it to a browser

I could write pages and pages about Node, but your takeaway from this should be "Node is important, I should be on the lookout more knowledge about Node and how it relates to React and web development."

**NPM**

[NPM](https://en.wikipedia.org/wiki/Npm_(software)) does not stand for "Node Package Manager", but it should, because that's exactly what it does. React, SASS, Angular, Vue, pretty much everything mentioned here you will probably use npm to install and keep updated.

**Webpack**

[Webpack](https://webpack.js.org/) is a "module bundler". It takes all your js and css files and writes them to one file so you only have to worry about writing one <script> tag.

Each React component will have at least one js file associated with it. Each component should have it's own file too. Keeping track of all those files is very demanding. Webpack does it for you, it just makes life easier, so learn about it early and don't shy away from it.

This is something so inherent to React Apps that most of the time it will just be working and you won't even know it. For instance [create-react-app](https://github.com/facebook/create-react-app#creating-an-app) installs it automatically, and does not require you to do anything - same with Babel...

**Babel**

Translates all your code to ECMA5 so it can be read by the most browsers and the most versions of those browsers.

Again, this can be installed with npm, or if you just want to play around with React and not get too bogged down with the minutia like this, you can run create-react-app, and this will just work with automatic settings and will be out of your hair while you learn.

**They make stuff easier**

NPM, Webpack, Babel, and many other Node packages are only there to make your life easier. Building web apps requires a lot of maintenance - or small, non-programming annoyances that typically you don't even need to think about.

Try not to be intimidated by new packages because wielding their power can mean countless hours devoted to more interesting things.

***Conclusion***

Hopefully this post has helped you learn the difference between React, AJAX, and the ongoing nature of web application development. React and AJAX are not comparable, but React uses AJAX, or rather you - the developer - use AJAX in React to get data without the page needing to reload.

AJAX and other technologies were monumental to the advancement of web applications, but because of how absolutely essential they were to applications, they were assimilated into new technologies so much so that you don't even have to know about them to reap their benefits.

My goal was to correct some misconceptions on your path of learning; explain the "why" of the current state of web dev; and introduce technologies you didn't mention but should know about: Node, npm, Babel.

If you want to continue learning, I highly recommend doing a tutorial in React. I have done some at [platform.ui.dev/](https://platform.ui.dev/), and enjoy their approach to learning and their payment structure (I haven't been paid to say this). Good luck out there, and I hope this was helpful.