Case Study: Why did Facebook engineers create React?
There are a lot of JavaScript Model-View-Controller (MVC) frameworks out there. Why did we build React and why would you want to use it?
React isn't an MVC framework.
React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time.
React doesn't use templates.
Traditionally, web application UIs are built using templates or HTML directives. These templates dictate the full set of abstractions that you are allowed to use to build your UI.
React approaches building user interfaces differently by breaking them into **components**. This means React uses a real, full-featured programming language to render views, which we see as an advantage over templates for a few reasons:
JavaScript is a flexible, powerful programming language with the ability to build abstractions. This is incredibly important in large applications.
By unifying your markup with its corresponding view logic, React can actually make views **easier to extend and maintain**.
By baking an understanding of markup and content into JavaScript, there's **no manual string concatenation** and therefore less surface area for XSS vulnerabilities.

We've also created JSX, an optional syntax extension, in case you prefer the readability of HTML to raw JavaScript.
React updates are dead simple.
React really shines when your data changes over time.

In a traditional JavaScript application, you need to look at what data changed and imperatively make changes to the DOM to keep it up-to-date. Even AngularJS, which provides a declarative interface via directives and data binding requires a linking function to manually update DOM nodes.
React takes a different approach.

When your component is first initialized, the **`render`** method is called, generating a lightweight representation of your view. From that representation, a string of markup is produced and injected into the document. When your data changes, the **`render`** method is called again. In order to perform updates as efficiently as possible, we diff the return value from the previous call to **`render`** with the new one and generate a minimal set of changes to be applied to the DOM.
The data returned from **`render`** is neither a string nor a DOM node — it's a lightweight description of what the DOM should look like.
We call this process **reconciliation**. Check out this jsFiddle to see an example of reconciliation in action.

Because this re-render is so fast (around 1ms for TodoMVC), the developer doesn't need to explicitly specify data bindings. We've found this approach makes it easier to build apps.
HTML is just the beginning.
Because React has its own lightweight representation of the document, we can do some pretty cool things with it:

Facebook has dynamic charts that render to **`<canvas>`** instead of HTML.

Instagram is a "single page" web app built entirely with React and **`Backbone.Router`**. Designers regularly contribute React code with JSX.
We've built internal prototypes that run React apps in a web worker and use React to drive **native iOS views** via an Objective-C bridge.
You can run React on the server for SEO, performance, code sharing and overall flexibility.

Events behave in a consistent, standards-compliant way in all browsers (including IE8) and automatically use event delegation.

Head on over to https://reactjs.org to check out what we have built.