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](https://reactjs.org/docs/jsx-in-depth.html), 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](https://code.angularjs.org/1.0.8/docs/guide/directive#reasonsbehindthecompilelinkseparation).

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](https://jsfiddle.net/2h6th4ju/) 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](https://reactjs.org/) to check out what we have built.

# The Virtual DOM

React builds a representation of the browser Document Object Model or DOM in memory called the virtual DOM. As components are updated, React checks to see if the component’s HTML code in the virtual DOM matches the browser DOM. If a change is required, the browser DOM is updated. If nothing has changed, then no update is performed.

As you know, this is called the **reconciliation** process and can be broken down into the following steps:

**Step 1:** The virtual DOM is updated.

**Step 2:** The virtual DOM is compared to the previous version of the virtual DOM and checks which elements have changed.

**Step 3:** The changed elements are updated in the browser DOM.

**Step 4:** The displayed webpage updates to match the browser DOM.

As updating the browser DOM can be a slow operation, this process helps to reduce the number of updates to the browser DOM by only updating when it is necessary.

But even with this process, if a lot of elements are updated by an event, pushing the update to the browser DOM can still be expensive and cause slow performance in the web application.

The React team invested many years of research into solving this problem. The outcome of that research is what’s known as the React Fiber Architecture.

The Fiber Architecture allows React to incrementally render the web page. What this means is that instead of immediately updating the browser DOM with all virtual DOM changes, React can spread the update over time. But what does "over time" mean?

Imagine a really long web page in the web browser. If the user scrolls to the bottom, the top of the web page is no longer visible. The user then clicks a button on the bottom of the web page that updates some text on the top of the web page.

But the top of the page isn’t visible. Therefore, why update it immediately?

Perhaps there is text currently displayed on the bottom of the page that also updates when the button is clicked. Wouldn’t that be a higher priority to update than the non-visible text?

This is the principle of the React Fiber Architecture. React can optimize when and where updates occur to the browser DOM to significantly improve application performance and responsiveness to user input. Think of it as a priority system. The highest priority changes, the elements visible to the user, are updated first. While lower priority changes, the elements not currently displayed, are updated later.

While you’re unlikely to interact with the virtual DOM and Fiber Architecture yourself, it’s good to know what’s going on if issues occur during the development of your web application.

There are many tools available to help you investigate how React is processing your webpage. The official React Developer Tools web browser plugin developed by Meta will be one of the key tools in your developer toolbox. So, if you do have to look deeper into the code, you’ll have the right toolbox available to help you. These tools will be explored later on.

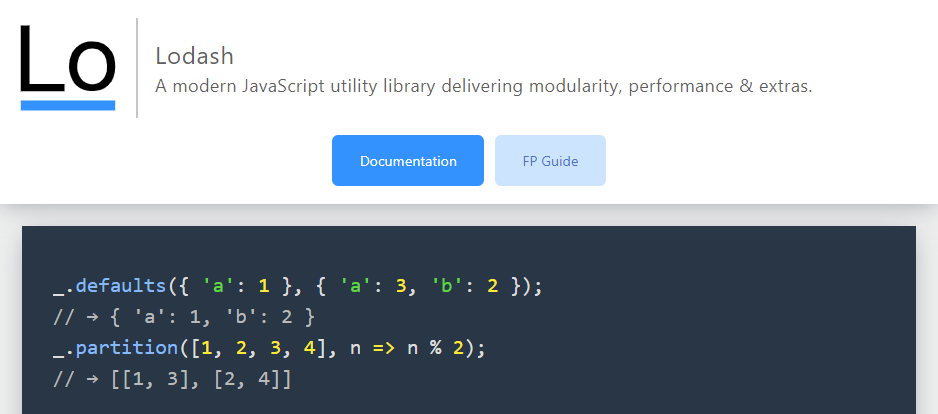
# Alternatives to React

React is a library and not a framework. This means you'll often use other JavaScript libraries with it to build your application. In this reading, you will be briefly introduced to some JavaScript libraries commonly used with React.

### Lodash

[Official Website](https://lodash.com/)

As a developer, there's a lot of logic you'll commonly write across applications. For example, you might need to sort a list of items or round a number such as **3.14** to **3**. Lodash provides common logic such as these as a utility library to save you time as a developer.



### Luxon

[Official Website](https://moment.github.io/luxon/#/)

You'll be working with dates and times often as a developer. Think of viewing a list of orders and when they were placed, or displaying a calendar schedule for an event. Dates and times are everywhere.

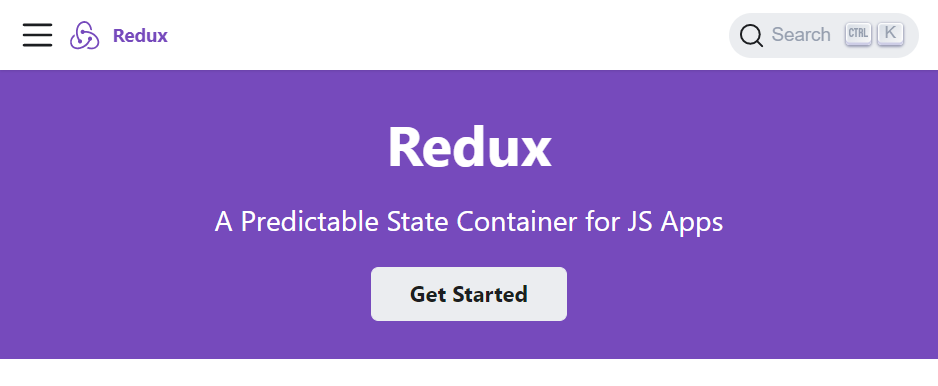
Luxon helps you work with dates and times by providing functions to manipulate and display them. For example, think of how dates are formatted in different countries. In the United States the format is **Month Day Year** but in Europe it is **Day Month Year**. This is one area where Luxon can help you display the date in the user's local format.



### Redux

[Official Website](https://redux.js.org/)

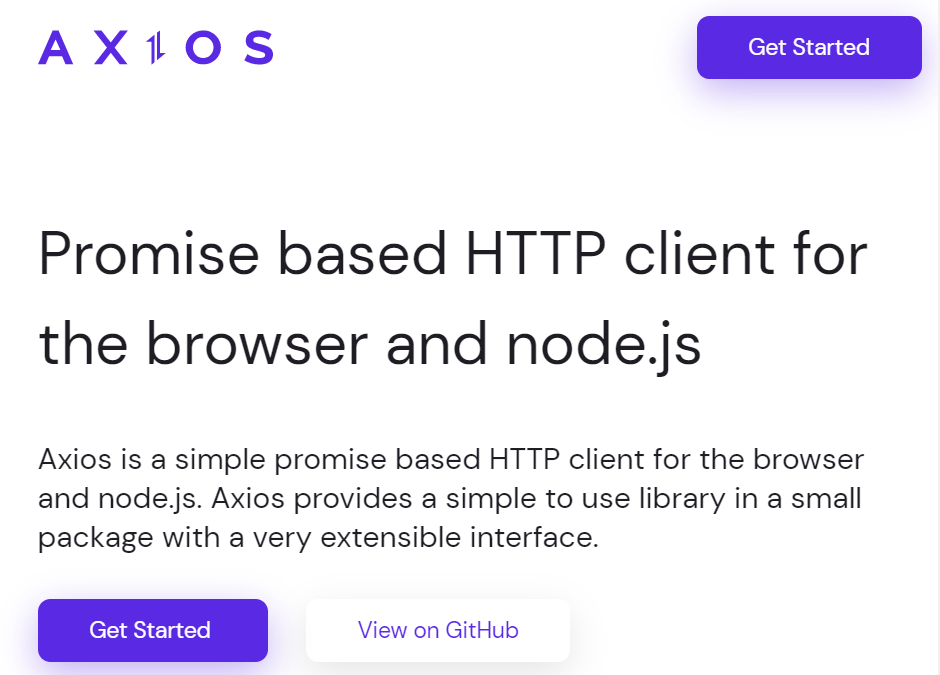
When building a web application, you'll need to keep track of its state. Think of when you shop online. The web application tracks items currently in your shopping cart. When you remove an item from the cart, the application needs to update what displays on the screen. This is where Redux comes in. It helps you manage your application state and even has advanced features such as undo and redo.



### Axios

[Official Website](https://axios-http.com/)

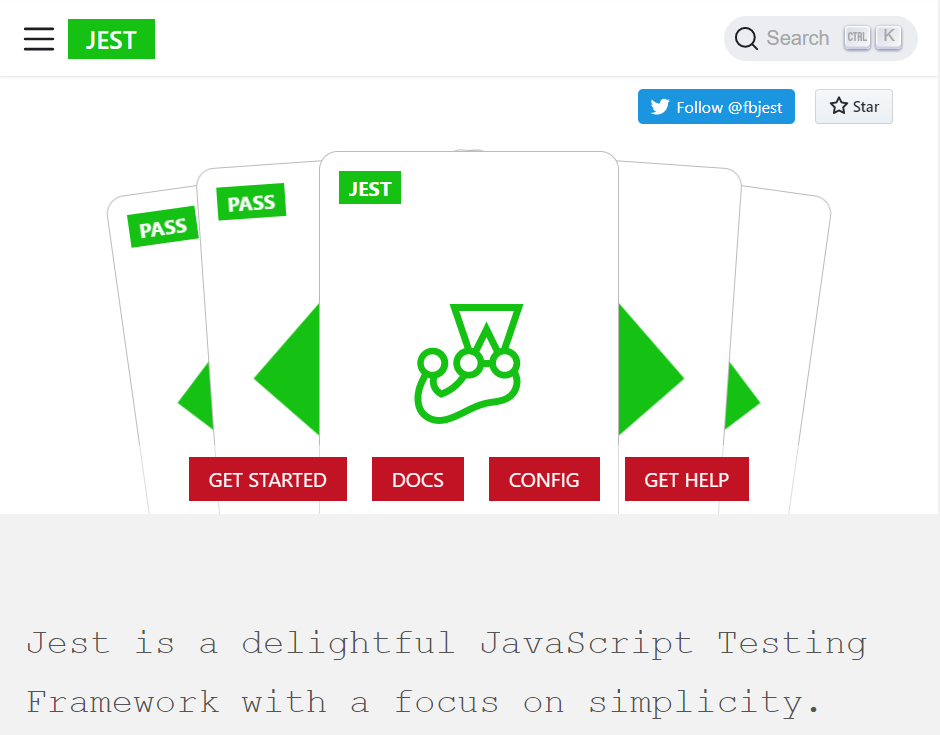
As a developer you'll be communicating with APIs over HTTP frequently. The Axios library helps to simplify sending HTTP requests and processing the response. It also provides advanced features allowing you to cancel requests and to change data received from the web server before your application uses the data.



### Jest

[Official Website](https://jestjs.io/)

It is good practice to write automated tests for your code as a professional developer. The jest library helps you to do this and works with many libraries and frameworks. It also provides reporting utilities such as providing information on how much of your code is tested by your automated tests.



### Conclusion

If you're curious to learn more about these libraries, their websites feature setup guides, tutorials and documentation to get started. These libraries will be covered later on.

Mark as completed

Like

Dislike

Report an issue

Top of Form

Bottom of Form

# Additional Resources

**Learn more​** Here is a list of resources that may be helpful as you continue your learning journey.

### **React Official Website**

<https://reactjs.org/>

**Choosing between Traditional Web Apps and Single Page Apps (Microsoft)**

<https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>

**React Source Code (Github)**

<https://github.com/facebook/react>

**Introduction to React.js**

The original video recorded at Facebook in 2013.

<https://youtu.be/XxVg_s8xAms>