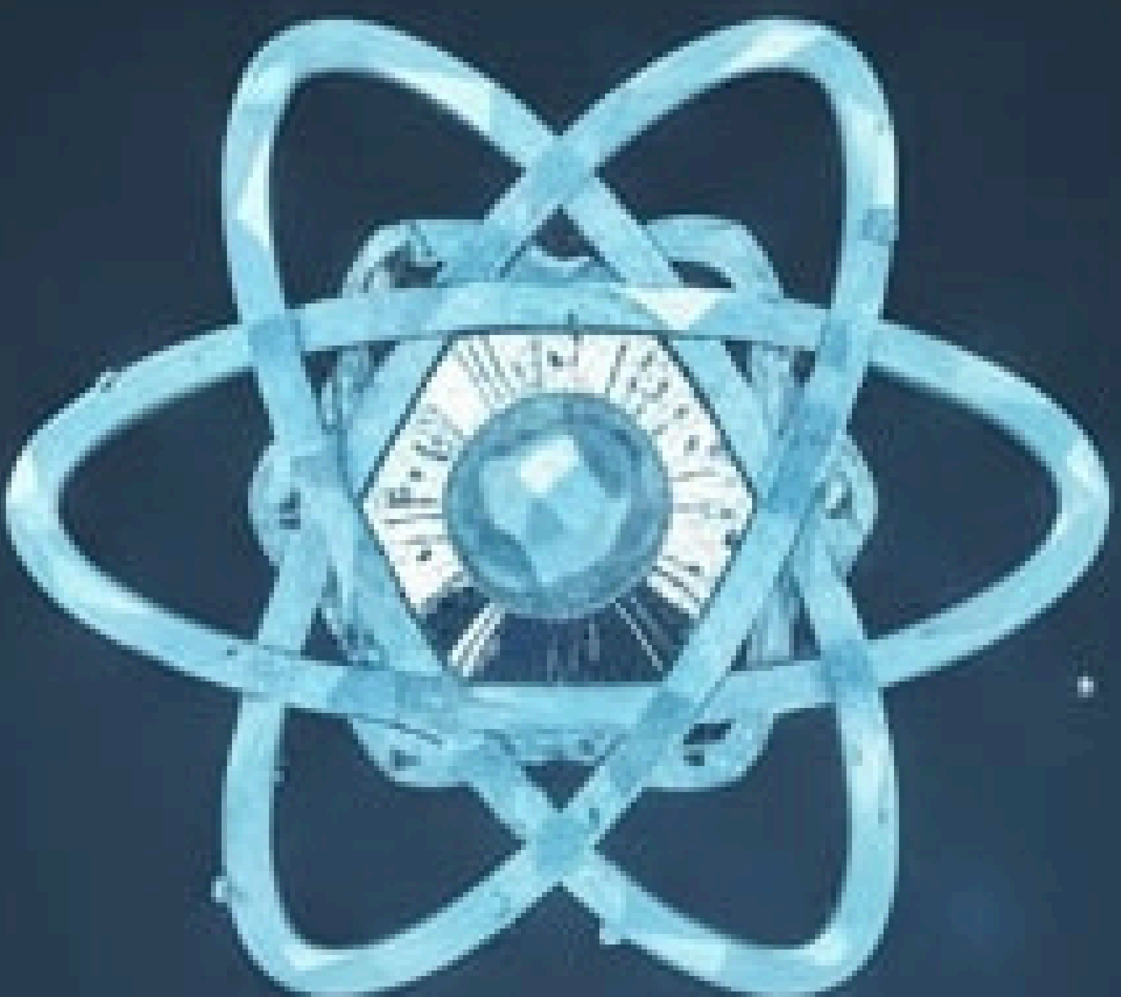

RENDERING REACT Vs VUE

THE ULTIMATE PERFORMANCE SHOWDOWN

DINO MIZINOVIĆ

Introduction

In the modern JavaScript world, React and Vue are basically the Avengers of UI development. Everyone uses them, everyone argues about them, and both are extremely good at saving your web app from becoming slow and ugly.

At a basic level, both frameworks exist to solve the same problem: how do we keep the screen in sync with the data when users are clicking, typing, scrolling, and generally trying their best to break things?

They both use something called a Virtual DOM, which is a lightweight copy of what the UI should look like. Instead of directly poking the browser's real DOM (which is slow and fragile), they work with this virtual version and then apply only the necessary changes.

However, the way React and Vue decide what needs to be updated is very different. React takes a "recalculate everything and then decide" approach, while Vue takes a "track everything and update only what changed" approach. It's the difference between recalculating your entire budget every time you buy coffee versus just subtracting the price of the coffee.

This difference may sound small, but at scale — thousands of components and constant updates — it becomes the difference between a smooth app and one that makes users stare at loading spinners.

Virtual DOM

React’s Virtual DOM strategy is based on re-rendering. When something changes, React re-runs the render function for that component and all of its children. It then compares the newly generated Virtual DOM tree with the previous one and figures out what actually changed. This process is called reconciliation, and it is powered by React’s Fiber engine.

It is rather like rewriting an entire shopping list just because you realised you forgot milk, and then checking the new list against the old one to find the difference. It works, but it’s a bit… enthusiastic.

Vue, on the other hand, uses dependency tracking. When a component is rendered for the first time, Vue notes which pieces of data were used. Later, if one of those pieces of data changes, Vue already knows exactly which components depend on it and need to be updated.

This means Vue doesn’t have to re-render everything and then compare — it simply updates the parts that are affected. No extra work, no polite overthinking, just efficient, precise updates.

Reactivity

React’s state model is built on immutability. This means you never directly change data. Instead, you create a new version of the data every time something changes. React then compares the old and new versions to determine what happened.

For simple data, this is fine. But if you have deeply nested objects — like a huge profile with settings, preferences, and history — those comparisons can become expensive. It’s like copying your entire hard drive just because you edited one text file.

Vue 3 uses a Proxy-based reactivity system. It wraps your data and intercepts reads and writes. When you change a property, Vue knows immediately, because it was watching that exact property. No guessing, no comparing, no scanning the whole object.

This fine-grained reactivity allows Vue to respond very quickly to changes, which is especially useful in apps with lots of small, frequent updates.

Re-rendering

React gives developers great power, but also great responsibility. By default, components re-render whenever their parent renders. This can lead to unnecessary work unless the developer actively uses tools like React.memo, useMemo, and useCallback to prevent it.

If you forget to optimise, React will happily re-render components that didn't really need to change — rather like a British person apologising to a chair they bumped into.

Vue, however, uses its reactivity system to determine exactly when a component should update. If the data it uses hasn't changed, it simply won't re-render. You don't have to sprinkle optimisation hooks everywhere just to get decent performance.

This makes Vue much more predictable and easier to keep fast as applications grow.

Bundle

Bundle size matters because it affects how fast users see something on their screen. Vue’s runtime is smaller than React’s, and it also includes more features built in, such as state management and transitions. That means fewer extra libraries and less code to download.

React often requires additional libraries for things like state management, which increases bundle size. It’s powerful, but you’re carrying more gear.

In runtime performance, Vue tends to handle frequent updates more efficiently, while React does very well when rendering large component trees for the first time. So Vue is great for live dashboards and chats, while React shines in complex, dynamic applications.

Template

Vue uses templates that are compiled at build time into highly optimized render functions. The compiler analyses which parts are static and which are dynamic and generates code that skips unnecessary work during updates.

React’s JSX is more flexible, but that flexibility comes at the cost of fewer compile-time optimizations. React must evaluate more at runtime to decide what changed.

Vue’s approach allows it to do more work ahead of time, resulting in faster updates when the app is running.

Real-World

In large applications with thousands of components, Vue often uses less CPU and feels smoother during frequent updates. This makes it excellent for real-time applications like chats, dashboards, and collaboration tools.

React offers more explicit control over rendering, which can be useful for complex state and highly dynamic UIs, but it requires careful optimisation.

So Vue is efficient by default, while React rewards developers who really know how to tune it.

Conclusion

Vue provides excellent performance out of the box with minimal effort. React provides maximum flexibility but requires more attention to performance.

Both are powerful, mature frameworks. The best choice depends on your team, your project, and how much control you want over rendering.

One prefers elegance and automation, the other prefers precision and engineering — and both can build fantastic applications when used well.