SIDEBAR

Web Development

React vs Angular vs Vue

Sidebar

Web Development · React vs Angular vs Vue

Sidebar is where this site keeps content that's genuinely worth a real, standalone write-up, but too narrow to justify a multi-chapter course of its own. This lesson is a practical comparison of three well-established answers to the same problem — building interactive UIs from reactive, composable components. They differ mainly in **how much they decide for you** and **what they ask you to learn**, and having a course on each of them already on this site, it's worth pulling the comparison out into its own page rather than repeating it three times.

Where to Go Deeper on Each Framework

> **THIS IS A COMPARISON OVERVIEW, NOT A REPLACEMENT**
>
> This lesson stays deliberately high-level. For real depth on any one of the three, see *React Fundamentals*, *React Intermediate*, *React Advanced*, and *React Projects*; *Angular*; and *Vue* — each already covers its own framework in far more depth than a single comparison page reasonably could.

The Short Version

All three solve the identical core problem: keep the DOM in sync with changing data, using reusable components. **React** is a *library* — it renders UI and leaves routing, state, and data-fetching to your choice of separate packages. **Angular** is a *full framework* — routing, forms, HTTP, and dependency injection all built in, one official way to do each. **Vue** sits deliberately *between* them — flexible and light like React, but with template directives and official router/state packages like Angular.

At a Glance

	REACT	ANGULAR	VUE
Category	Library	Full framework	Progressive framework
Language	JS / JSX (TS optional)	TypeScript (mandatory)	JS or TS (your choice)

	REACT	ANGULAR	VUE
Markup	JSX (in JS)	HTML template + directives	HTML template + directives (SFC)
Reactivity	Explicit (`useState`, re-render)	Signals / Zone change detection	Fine-grained (`ref` proxies)
Reusable logic	Custom hooks	Services + DI	Composables
Routing	React Router (separate)	Built in	Vue Router (official)
Global state	Context / Zustand / Redux	Services / NgRx	Pinia (official)
Data fetching	React Query / fetch	HttpClient + RxJS	fetch in a store / composable
Scoped styles	CSS Modules / lib	Automatic	`<style scoped>`
Opinionatedness	Low — you assemble it	High — batteries included	Medium — official options, not forced
Learning curve	Gentle core, sprawling ecosystem	Steep upfront (TS, RxJS, DI)	Gentlest on-ramp
Backed by	Meta	Google	Community / Evan You

The Same Component, Three Ways

A counter — state, a derived value, and a click handler — captures most of the day-to-day flavor of each:

REACT

```
function Counter() {
  const [count, setCount] = useState(0);
  const double = count * 2;

  return (
    <div>
      <p>{count} ({double})</p>
      <button onClick={() =>
        setCount(count + 1)}>
        +1
      </button>
    </div>
```

```
    );
  }
```

ANGULAR

```
@Component({
  selector: 'app-counter',
  template: `
    <p>{{ count() }}
       ({{ double() }})</p>
    <button (click)="inc()">
      +1
    </button>`,
})
export class Counter {
  count = signal(0);
  double = computed(
    () => this.count() * 2);
  inc() {
    this.count.update(n => n+1);
  }
}
```

VUE

```
<script setup>
import { ref, computed }
  from 'vue';
const count = ref(0);
const double = computed(
  () => count.value * 2);
</script>

<template>
  <p>{{ count }}
     ({{ double }})</p>
  <button @click="count++">
    +1
  </button>
</template>
```

Notice the convergence: Angular's `signal` / `computed` and Vue's `ref` / `computed` are now strikingly similar — both fine-grained reactive systems. React stands a little apart with its "re-

run the whole function on each render" model, which is conceptually simpler but is exactly why it needs `useMemo` / `useCallback` for the optimizations the other two get more automatically.

Strengths & Weaknesses

React

The flexible default

STRENGTHS

- Largest ecosystem and job market by far
- Unmatched flexibility — pick your own stack
- JSX keeps logic and markup together in plain JS
- React Native shares skills with mobile

WEAKNESSES

- "Just a library" means many decisions are yours
- Manual memoization for performance tuning
- Ecosystem churn — best practices shift often
- Easy to assemble an inconsistent codebase

Angular

The all-in-one

STRENGTHS

- Everything built in — one official way per task
- TypeScript & structure enforced from day one
- Excellent for large teams & long-lived apps
- Powerful DI and RxJS for complex async

WEAKNESSES

- Steepest learning curve (TS, RxJS, DI upfront)
- More verbose / more boilerplate
- Can feel heavyweight for small projects
- Smaller hiring pool than React

Vue

The approachable middle

STRENGTHS

- Gentlest learning curve of the three
- SFCs keep template/logic/style tidy in one file
- Official router & Pinia — guidance without lock-in
- Excellent docs; fine-grained reactivity

WEAKNESSES

- Smaller ecosystem & job market than React
- No single corporate backer (a risk to some)
- Two API styles (Options vs Composition) in the wild
- Less common in large enterprise shops

How to Choose

There's no universally "best" one — the honest answer is "it depends," and here's what it depends on:

React	You want the biggest hiring pool and ecosystem, maximum flexibility, or plan to share code with React Native mobile.
Angular	A large team or enterprise app where consistency, structure, and one-official-way-per-task matter more than flexibility; TypeScript and strong conventions are a plus, not a cost.
Vue	You want the fastest on-ramp, a clean single-file component model, and official-but-not-forced router/state — great for small-to-mid teams and solo builders.
Any	For most ordinary apps, all three are entirely capable. Team familiarity and existing codebase usually outweigh technical differences — the best framework is often the one your team already knows.

A NOTE ON META-FRAMEWORKS

In production, each is often used through a meta-framework that adds server-side rendering, routing, and build conventions: React → Next.js, Vue → Nuxt, Angular → Angular itself (SSR built in). If SEO, fast first-load, or full-stack structure matter, the comparison often shifts to *these* rather than the bare libraries — this site's own *Website Rebuild with Next.js*, *Website Rebuild with Astro*, and *Website Rebuild with Express* courses are the natural next stop.

The Honest Takeaway

THE THREE HAVE CONVERGED MORE THAN THE DEBATES SUGGEST

Components, props, one-way data flow, reactive state, derived values, lifecycle, reusable logic units, official routing and state stories — all three now have all of these. Having learned one well, the second and third are mostly a matter of translating syntax. So the practical advice is less "which is best" and more: **learn one deeply, understand the shared concepts underneath, and you can move between all three** — the differences from here are details, not paradigm shifts.

Hands-On Exercises

EXERCISE 1

A large enterprise team building a long-lived internal app wants strict conventions, enforced structure, and TypeScript from day one, even at the cost of a steeper learning curve. Using this lesson's own "How to Choose" section, which framework fits best, and why?

View solution

EXERCISE 2

This lesson says Angular's `signal` / `computed` and Vue's `ref` / `computed` have "converged," while React's model "stands a little apart." What specific difference in how each framework re-renders causes this, and why does React need `useMemo` / `useCallback` as a direct consequence?

View solution

EXERCISE 3

A team wants server-side rendering, better SEO, and full-stack build conventions on top of their chosen framework. Name the meta-framework this lesson associates with each of React, Angular, and Vue, and explain why production comparisons often shift to these instead of the bare framework/library.

View solution

QUICK REFERENCE

- **React** — a library, JSX, explicit re-render (`useState`), largest ecosystem, pairs with Next.js for SSR/SEO
- **Angular** — a full framework, TypeScript mandatory, signals + built-in DI/routing/HTTP, steepest learning curve, SSR built in
- **Vue** — a progressive framework, HTML templates (SFCs), fine-grained `ref` reactivity, gentlest on-ramp, pairs with Nuxt for SSR/SEO
- All three now share the same shared concepts (components, props, reactive state, derived values, official routing/state) — differences are mostly syntax, not paradigm
- For real depth: see *React Fundamentals/Intermediate/Advanced/Projects*, *Angular*, and *Vue* on this site