Frameworks für responsives Design

Ein **Framework** ist ein **vorgegebenes Gerüst** für Software, das viel Standardarbeit abnimmt: Struktur, typische Bausteine (Komponenten), Regeln/Best Practices und oft auch Tools (Build, Routing, Testing). Frontend-Frameworks helfen beim Bau von Web-UIs.

Das Framework **gibt den Ablauf vor** („Inversion of Control"): *dein* Code "hängt" sich an das Framework (Lifecycle, Components, Routing, etc.).

Wichtigster Unterschied zur Library:

- Man ruft die Library auf („Ich benutze eine Funktion.").

Das bekannteste aktuelle Framework ist jedoch ***Bootstrap* von Twitter**. Es bietet viele vorformatierte Layout-, Typografie- und Formularoptionen, zahlreiche wiederverwendbare Komponenten und auch eine Erweiterbarkeit durch JavaScript.

Vorteile von Frameworks:

Bei so gut wie allen Projekten muss man auch Aufgaben lösen, die an sich nichts Neues sind und bei denen die Wahrscheinlichkeit groß ist, dass sie bereits jemand anderes gelöst hat. Daher kann man zu einem Framework greifen und den Code von anderen weiterverwenden, der oft besser durchdacht und gründlicher getestet ist.
Frameworks bieten aber weitere Vorteile:

- **Schneller starten:** Viel ist schon gelöst (Routing, Komponenten, Build, Struktur).
- **Weniger Fehler / Best Practices:** Vorgegebene Patterns machen Code oft stabiler und wartbarer.
- **Wiederverwendbare Komponenten:** UI in Bausteinen → weniger Copy/Paste, bessere Konsistenz.
- **Großes Ökosystem:** Viele fertige Pakete (UI-Komponenten, Form-Validation, Auth).
- **Teamarbeit & Skalierung:** Einheitliche Projektstruktur hilft in größeren Teams.
- **Performance-Optimierungen "out of the box":** z. B. effizientes Rendering, Code-Splitting.
- **Testing/Tooling integriert:** TypeScript-Support, DevTools usw.
- **Sicherheit** ist heute ein wichtiges Thema. Frameworks bieten oft bewährte Strategien, um die Sicherheit der Anwendung zu gewährleisten.
- **Organisation**: Frameworks geben einen Weg vor, wie der Code zu organisieren und zu schreiben ist. Das reicht von einfachen Vorgaben für Speicherorte von Dateien über Konventionen der Benennung bis hin zur Aufteilung von Aufgaben.

- **Weniger ist mehr**: Die Codemenge, die man selbst schreiben muss, reduziert sich beim Einsatz eines Frameworks; das bedeutet, es geht schneller, und außerdem minimiert sich die Fehlerquote.

Frameworks haben auch Nachteile:

- Jedes Framework braucht **Zeit zur Einarbeitung** und so lohnt sich der Einsatz eines Frameworks als Einzelkämpfer eher, wenn Sie mehrere Projekte damit realisieren.
- Vom Framework vorgesehene Aufgaben realisiert man im Allgemeinen wesentlich schneller. Wenn man aber etwas anderes umsetzen möchte, kann der **Aufwand** größer sein.
- **Lernkurve:** kann groß sein für Konzepte wie Komponenten-Lifecycle, State, Routing
- **Komplexität/Overhead:** Für kleine Seiten manchmal „zu schwer" (Bundle, Setup, Abhängigkeiten).
- **Abhängigkeit/Vendor Lock-in:** man bindet sich an API/Patterns; Wechsel kostet Zeit.
- **Updates & Breaking Changes:** Framework-Versionen können Anpassungen erzwingen.
- **Sicherheits-/Maintenance-Risiko durch Dependencies:** Viele Pakete → mehr Angriffsfläche/Update-Pflicht.

Frameworks stellen Bausteine für typische Layoutsituationen und User-Interface-Elemente bereit. Außerdem **umgehen sie bekannte CSS-Bugs**, insbesondere für ältere Browser.

Welche Auswahl an Frontend-Frameworks gibt es? (grobe Kategorien)

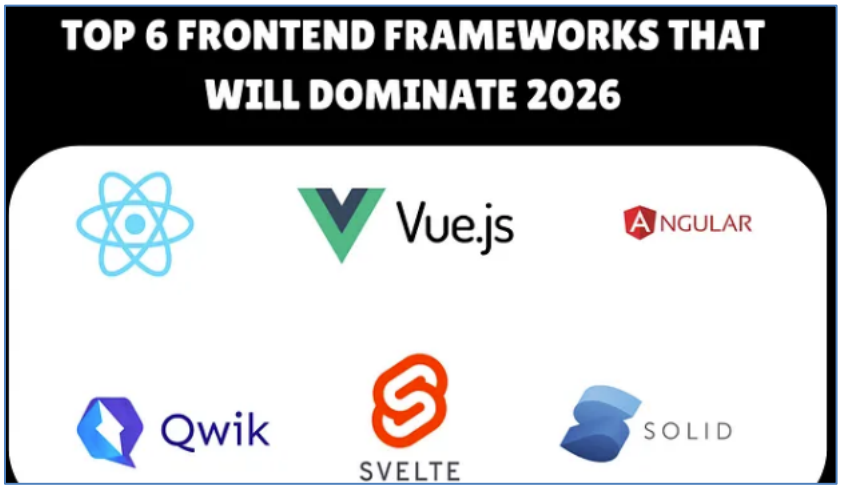
- **UI-Frameworks/-Libraries:** React, Angular, Vue, Svelte, Solid
- **Meta-Frameworks (SSR/SSG, Fullstack-Frontend):** Next.js (React), Nuxt (Vue), SvelteKit (Svelte), Astro
- **CSS/UI-Frameworks (anderer Typ "Framework"):** Bootstrap, Tailwind CSS, Material UI etc. (die sind eher Styling/UI-Komponenten, nicht "App-Frameworks")

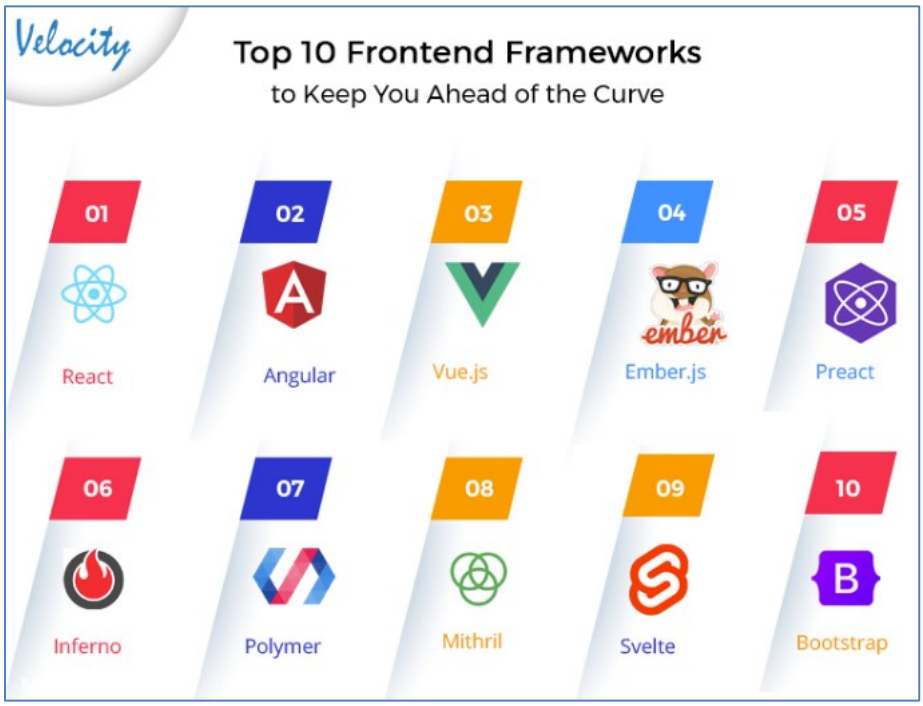
Die 4 wichtigsten Frontend-Frameworks

In großen Entwickler-Umfragen und Nutzungsindikatoren dominieren weiterhin diese vier:

1. **React** (streng genommen eher Library, aber de-facto das wichtigste UI-Stack)
2. **Angular** (vollständiges, „batteries-included" Framework, oft Enterprise)
3. **Vue.js** (sehr beliebt, „leichtgewichtig", gutes Ökosystem)

4. **Svelte** (Compiler-Ansatz, kleiner Footprint, starkes Momentum)





Beispiel Angular

An **Angular** ist vor allem gut, dass es ein **"Full-Framework" (App-Framework)** ist: Man bekommt sehr viele Dinge **standardisiert und integriert**, was bei größeren Projekten und Teams extrem hilft.

Damit meint man: Angular liefert dir nicht nur UI-Bausteine, sondern **das komplette Gerüst für eine Anwendung**, inkl. typischer App-Funktionen und klarer Architektur.

Was Angular besonders stark macht

- Klare, einheitliche Architektur („opinionated")
  Angular liefert dir ein stabiles Grundgerüst und Konventionen, die Code in großen Projekten wartbarer machen.
- Router, Forms, HTTP, Testing, Tooling sind inkludiert
  Viele Standardbausteine sind offiziell dabei (statt alles selbst zusammensuchen zu müssen).
- TypeScript-first
  Angular ist auf TypeScript ausgelegt, was Fehler früher sichtbar macht und Refactoring erleichtert.

Beispiel Bootstrap

Bootstrap ist ein Frontend Toolkit / CSS-Framework.
Eine Sammlung aus vordefiniertem CSS, UI-Komponenten (Buttons, Navbars, Formulare …) und optionalem JavaScript (z. B. Modal, Dropdown, Tooltip), mit der man responsive, mobile-first Websites schnell bauen kann.

Warum nennt man Bootstrap „Framework"? Weil es ein **Gerüst + Regeln** liefert:

- ein **Grid-System** für Layouts (Container → Row → Columns, basiert auf Flexbox, responsive)
- **vorgefertigte Komponenten** mit konsistentem Styling, Forms, Buttons, Cards, Navbar
- viele **Utility-Klassen** (Abstände, Display, Flex, Text, Farben), um ohne viel eigenes CSS zu layouten

Wichtig: Bootstrap ist **nicht** so ein „App-Framework" wie Angular. Es löst primär **Layout/Styling/UI** (und etwas JS für UI-Interaktionen), aber nicht „App-Architektur/State-Management".

Vorteile:

- Sehr schnell „schöne, responsive" Seiten/Prototypen
- Wenig eigenes CSS nötig (Utilities + Komponenten)
- Gut für klassische Websites, Admin-UIs, Schulprojekte, kleinere Webapps

Nachteile:

- Viele Seiten sehen „bootstrap-mäßig“ aus, wenn man nicht bewusst customized
- Für sehr individuelles Design kann man gegen das vorgegebene CSS ankämpfen

JavaScript-Bibliothek

Viele der erwähnten Frameworks nutzen weitere Bibliotheken zur Unterstützung.

jQuery
jQuery hat sich zu DER JavaScript-Standardbibliothek entwickelt. Es stellt einfache Funktionen bereit, mit denen sich DOM-Elemente ansprechen und verändern lassen. Es verfügt über mehrere Zusatzpakete wie jQuery UI (User Interface Elemente) und jQuery Mobile (JavaScript-Funktionen für mobile Websites).

www.jquery.com/download