




DIFERENCIAS ENTRE JAVASCRIPT VANILLA Y FRAMEWORKS



Solorzano Rodas, Julio Andres
[COMPANY NAME] [Company address]

Tecnología	Año de creación & quién mantiene	Tipo (framework / librería / meta-framework)	Ventajas	Desventajas	Casos de uso recomendados	¿Qué se puede hacer más rápido que con vanilla?
JavaScript (vanilla)	Lenguaje fundado a mediados de los años 90; mantenido por el estándar ECMA International (ECMAScript)	Lenguaje base / sin framework	- No dependes de librerías externas - Todo el control es “puro” - Ligero (no overhead de framework)	- Mucho código repetitivo para interfaces complejas - Difícil escalar para UIs grandes - Falta de estructura predeterminada	Pequeñas páginas web, scripts simples, prototipos rápidos	Básico: manipular DOM sencillo, sin abstracciones. Pero para sistemas de componente o estados complejos se hace más lento.
React	Creado por Meta Platforms (antes Facebook). Versión pública open source ~2013. Wikipedia+1	Librería de UI (aunque muchos la llaman “framework”)	- Arquitectura basada en componentes reutilizables - Virtual DOM mejora las actualizaciones del UI - Ecosistema enorme, comunidad muy activa DEV Community	- Necesitas integrar muchas librerías adicionales para routing, estado, etc - Curva de aprendizaje para “hooks”, contexto, ecosistema - Puede producir bundles pesados	Aplicaciones de usuario ricas, SPAs, interfaces que cambian mucho	Crear UIs reactivos con estado y representación eficiente sin tener que gestionar manualmente cada manipulación del DOM (como con vanilla).
Vue.js	Creado por Evan You. Primer commit en 2013,	Framework progresivo de UI	- Sintaxis más amigable, templating	- Ecosistema menos vasto que React	Proyectos medianos, equipos que	Crear componentes reutilizables y manejar la reactividad (estado -> UI)

Tecnología	Año de creación & quién mantiene	Tipo (framework / librería / meta-framework)	Ventajas	Desventajas	Casos de uso recomendados	¿Qué se puede hacer más rápido que con vanilla?
	lanzado públicamente ~2014. Wikipedia+1		declarativo - Curva de aprendizaje menor que otros grandes - Buena integración incremental (puedes adoptarlo parcialmente) DEV Community	- Algunas empresas grandes lo usan menos que React en ciertos ámbitos DEV Community	quieren montar algo rápido sin tanto “overhead”	sin tener que “pintar” manualmente cada cambio.
Angular	Originalmente AngularJS (2010) pero la versión “Angular” moderna (2+) se lanzó en 2016 por Google. livebook.manning.com+1	Framework completo (SPA)	- Solución “baterías incluidas”: routing, estado, inyección de dependencias, etc - Tipado fuerte (TypeScript) - Buena para grandes equipos, grandes aplicaciones empresariales fleischer.design	- Curva de aprendizaje empinada - Puede sentirse “pesado” para proyectos pequeños - Menos “flexible” que librerías más ligeras	Aplicaciones corporativas grandes, mantenimiento a largo plazo, múltiples módulos, varios equipos	Montar una estructura de aplicación con módulos, servicios, rutas, roles, etc, de forma más rápida que arrancar desde cero con vanilla y definir tú mismo toda la arquitectura.

Tecnología	Año de creación & quién mantiene	Tipo (framework / librería / meta-framework)	Ventajas	Desventajas	Casos de uso recomendados	¿Qué se puede hacer más rápido que con vanilla?
Svelte	Creado por Rich Harris. Primera versión ~2016. Wikipedia+1	Framework / compilador de UI	- Compila a JavaScript puro en build-time (menos “runtime overhead”) - Bundle más pequeño, mejor rendimiento codesphere.com - Muy buena experiencia de desarrollo con sintaxis limpia	- Ecosistema más pequeño que React/Vue - Menos presionado en el mercado laboral que los grandes - Algunas integraciones pueden requerir más verificación	Sitios donde el rendimiento es crítico, PWA, equipos pequeños que quieren velocidad de desarrollo	Desarrollar componentes/reactividad con menos código, menor peso de librería, tiempos de inicio más rápidos comparado con vanilla/manual porque muchas tareas se eliminan o automatizan.

Comentarios adicionales

- Con **vanilla JS**, todo se hace “a mano”: manipulación del DOM, actualización del estado, sincronización entre datos y UI. Esto es viable para cosas pequeñas, pero cuando la interfaz crece se complica mucho.
- Con los frameworks o librerías, se obtiene abstracción: manejar el estado, UI, reactividad, componentes, rutas, etc. Esto permite **desarrollar más rápido**, con menos “boilerplate” repetitivo.
- Por ejemplo, usar Vue o React te permite decir “cuando el estado cambia, la UI se actualiza automáticamente” en lugar de poner escuchas manuales en eventos, buscar elementos del DOM, actualizarlos... todo eso con vanilla.
- Con Angular obtienes una estructura empresarial lista para usar: módulos, servicios, inyección de dependencias, etc, que con vanilla tendrías que construir tú.
- Con Svelte, el hecho de que compile a código más puro significa que la sobrecarga de librería es menor, lo cual se traduce en mejor rendimiento y bundle más ligero.

JavaScript **vanilla** sigue siendo mejor opción cuando el proyecto es **pequeño o simple**, como una landing page, un script puntual o una funcionalidad aislada. También conviene usarlo cuando se busca **máximo rendimiento y mínimo peso**, sin dependencias externas. Es ideal para **aprender los fundamentos** del lenguaje o cuando no se justifica la complejidad adicional de un framework.