UI scale in Laravel: from Blade to SPA

Blade, Alpine.js, Livewire, Inertia.js, and SPA: choosing the UI tool in a Laravel project by interaction density, SEO needs, and the team.

01 LARAVEL · UI

UI scale in Laravel: from Blade to SPA

Each step answers a different need. The choice is made not by tool quality but by interaction density, visibility requirements, and the team.

Muhammet Şafak · October 3, 2026



02 DATE OF THE SOURCES

This visual is based on posts from 2020-2021

> **DATE** The source posts were written in 2020 and 2021. The visual has no version-specific detail; the decision logic was carried over.

03 THE DIFFERENCE BETWEEN ALPINE AND VUE

The difference is not tool quality, it is scale.

When "an SPA for everything" becomes the starting point, the trade-offs disappear from view. A project's visibility requirement is also part of the architectural decision.

04 THE STEPS

Five scales from Blade to SPA

1 — Blade
The familiar path: the template is rendered on the server.

2 — Alpine.js
A few independent interactions in the template: dropdown, accordion, conditional form field.

3 — Livewire
A form-heavy panel; each interaction is one request to the server, and DOM diffs are applied.

4 — Inertia.js
The monolith stays, the UI is Vue or React; no API to write, a single client.

5 — SPA + API
A separate codebase; mobile or third-party clients can consume the API too.

05 DECISION MAP

Interaction density and search visibility

The vertical axis shows whether the page needs to be found by search engines; the horizontal axis shows how dense the client-side interaction is.

CONTENT BEHIND A LOGIN ↔ SEO REQUIRED

Blade + Alpine.js
Rendered on the server
- HTML arrives complete from the server
- Alpine is enough for a few independent interactions

SSR, SSG, or ISR
Next.js, Nuxt.js
- New content-heavy project: SSG or ISR
- The bot sees the full content on first load

Livewire
Admin panel
- Filtering, pagination, form steps
- A server round-trip is acceptable

Inertia.js or SPA
Behind a login
- If content is behind a login, SEO is not a real problem
- One client: Inertia; multiple clients: a separate API

LITTLE INTERACTION ↔ HEAVY INTERACTION

06 TWO PROJECTS, SAME YEAR

Livewire and Inertia.js

TOPIC	LIVEWIRE	INERTIA.JS
Who it is for	A backend-focused team; minimal JavaScript	A team with a frontend developer on board
UI	A PHP class and a Blade view	A Vue or React page component
Suitable project	A form-heavy admin panel	Complex client-side interaction
Limit	Every interaction is an HTTP request; latency shows at high frequency	Server and client are tightly coupled; the frontend cannot evolve independently

07 HOW INERTIA.JS WORKS

An SPA feel without an API

1 — Click a link
Inertia intercepts the page transition.

2 — Controller
Inertia::render() returns the page component and its data.

3 — JSON response
The component name and props arrive; shared data is added to every response.

4 — Update in place
No full page reload; the component updates in place.

08 SEARCH VISIBILITY IN AN SPA

Four options

OPTION	HOW IT WORKS	GAIN AND COST
SSR	HTML is generated on the server on every request	The bot sees the full content on first load; it consumes server resources and needs a Node.js server
SSG	All pages are generated at build time	Served from a CDN, very fast; build time grows, limited for dynamic content
ISR	Generated statically, refreshed in the background at set intervals	Balances SSG's build time against SSR's server load
Prerender	Pre-rendered HTML for bots, the SPA for users	Low conversion cost; an extra dependency, content may fall out of sync

09 THE RIGHT MATCH FOR THE JOB

Using Alpine.js and Livewire at their own scale

✓ Do

- Keep Alpine for small, independent interactions in the template
- Let Alpine open the modal instantly, and Livewire fetch the data in the background
- On form fields, use wire:model.lazy so you don't fire a request on every keystroke
- Move growing x-data logic into a separate component file

✗ Don't

- Don't load cross-page state and complex component communication onto Alpine
- Don't leave instant search, drag-and-drop, and animation to a server round-trip
- Don't leak business logic into the template
- Don't expect Livewire to handle complex client state

10 BEFORE YOU CHOOSE

Decision questions

- [ ] Are the pages reached without logging in and found through search engines?
- [ ] Is the interaction heavy, does the user expect millisecond precision?
- [ ] Is there someone on the team who can write Vue or React?
- [ ] Will a client other than the web UI consume the API?



11 SPA AND AUTHENTICATION

Cookie or token?

> **TIP** For an SPA on the same root domain (same-site), a cookie-based session is cleaner; the hassle of storing and refreshing tokens goes away.

> **WARNING** localStorage is open to XSS attacks; don't store the token there.

> **NOTE** The SPA and the API can be on different subdomains, but not on different root domains. Beyond that limit, you move to the token model.

> **PITFALL** If you use subdomains, the cookie domain must be set with a leading dot (.example.com); otherwise the cookie goes to only one subdomain.