On Frontend Frameworks, Design Systems, and AI with Commentary on Political and Technological Means

Kaiwen Wang

December 12, 2025

Contents

Chapter 1

Introduction

This thesis is about technology, or more specifically, frontend design systems and component libraries in Vue. I chose this specific topic out of my experience in frontend and coming from a fine arts high school, and because I believe the earlier stages before a browser: cell phones, computers, operating systems, browsers, and even the frontend framework itself have reached a level of maturity where systematic analysis would be unnecessary. Though I find myself capable of incrementalism, I find systems thinking to be what I both excel at and what is most lacking today. Of particular relevance in our present and future is the influence of politics upon technology in a re-emergent decade of two-pole competition and the development of Large Language Models (LLMs), colloquially referred to as AI.

Now, what is the nature of the world today, and what best describes it? I characterize it as *superficial variety with structural uniformity.* In our day and age we are presented with myraid of choices; we may see numerous brands of fast foods: McDonalds, Burger King, Wendy's, or one might go to a supermarket and see countless options of foods and brands, or even further one might see numerous styles and brands of clothes but the underlying stitching and fabric material remains the same.

Because fast food restaurants must work under constraints such as the availability of foodstuffs, of which mass-produced vegetable oils and factory farmed chicken has the cheapest scale, each place is essentially indistinguishable from the other only in that they differ slightly in their appearance and pricing. Restaurants at the low end in effect are a wrapper around food wholesalers such as Costco Business Center.

Processed foods in a supermarket are a wrapper around bleached wheat as per USDA standards and vegetable oils: look at the ingredients list and the all too familiar "Enriched Flour (Wheat Flour, Niacin, Reduced Iron, Vitamin B1

[Thiamin Mononitrate], Vitamin B2 [Riboflavin], Folic Acid)" plus "Soybean Oil" appears. Because the legal system of an administrative zone produces uniformity in foodstuffs, any sense of difference is an illusion once the wrapper is closely scrutinized.

This tendency of superficial variety and structural uniformity is characteristic of the modern times not only in supermarkets, restaurants, clothing, but also human personalities, news sites, choices in technologies, and styles of living across geographies. Google may present what seems to be an independent bevy of sources, yet all of them receive the same press briefing in the background. The automobile-city apartment-internet-supermarket life is mainstream everywhere, and what puts one place above another tends to be small incremental improvements such as roads and cars being slightly nicer rather than something completely orthogonal.

The ability to see past this false differentiation is necessary to not get caught seemingly running and making much noise but in reality standing still making no progress at all. When one wants to innovate, there lies infinite combination of items one can find themselves trapped in on the surface level. One could categornize infinite types of foods: or realize that they distill down to proteins, fats, and sugars.

But we need not confuse the lack of consolidation with the speciation of growth: a development of numerous forms appears as a frontier opens, only later do we find what should persist. Component libraries currently seem to fit this stage: there is enough growth and development, but the environment is messy and developments are unsure of the right path to take. I merely come here to bundle everything up and make the patterns clear so that we can focus on more important things, for software now is close to reaching its maturity as a social and coordinating layer in all facets of society.

For today, a sort of piercing ability to get at the core and essence of reality is needed: to hack at the overgrown jungle created by people's undiligent abstractions and see the straight path ahead, a clear abstraction representative of reality and truth.

1.1 The Development of Technologies

All technologies are implemented by humans, who together through shared mutual understanding, are then able to create themselves or have others implement these mental abstractions. What we see on a computer is no different: the specification bodies are the W3C for CSS and accessibility, WHATWG for HTML, and Ecma International for Javascript. Google, Apple, and Mozilla

then respectively implement the Blink, WebKit, and Gecko rendering engine.

Nevertheless, these specifications are rarely enough to satisfy two significant demands of frontend: reactivity and composability. Reactivity is the automatic response in appearance to the change in a variable or internal state while composability means the ability to break a website down into pieces so as to develop parts of the website without interfering in other parts, in addition to reusing components across the app where functionality may be the same and changes in one place will need to be propagated to all instances.

Currently and seemingly for the near future, frontend web frameworks such as React, Vue, Svelte, Solid, and Angular remain predominant for website development to fulfill these two requirements well. To some extent Web Components and Lit may allow reactivity and composability, but their awareness and usage in the developer community is not as high and the extent of their ecosystems not as developed, so for the sake of brevity we can evaluate whether these options are suitable for development at another time.

At an even higher level include features such as modals, buttons, breadcrumbs, sidebars, tabs, and other components which are commonly tabulated in a 'design system' or 'component library' due to their frequent reuse across web apps. When components have no CSS to them, they are termed 'headless components'.

Styling these days is generally scoped to the component with Tailwind CSS which gives numerous small design tokens such as 'mt-2' as shorthand for "margin-top: 2rem," CSS Modules if one prefers writing native CSS, or it is scoped to the component itself using the framework such as Vue or Svelte's Single-File Components. A solution that used to be more common in React but has since lost ground to Tailwind is CSS-in-JS with Emotion or styled-components, as it tended to be somewhat verbose and messy syntax, requiring backticks to escape strings for writing CSS in Javascript.

Finally, only after considering all these base layers, can a software developer then consider the relation of the frontend to the browser with full-stack frameworks such as Nuxt, Next.js, SvelteKit, client and server-side rendering, its relation to backend servers and databases, and fundamentally the software's relation to the real world and its business uses.

1.1.1 Repeatable Subunits

As humans move up the technology tree, things which were previously novel become repeatable subunits for the next layer. For example, there is little more understanding needed to innovate screws—most shapes exist and engineering projects select from an existing set of them. Electricity and steel become

commonplace. To not 'reinvent the wheel' is a comparative advantage for any organization.

We can make the observation, then, that no well-designed website can exist without proper layouts, and that no layouts can be good without proper components, of which no component would be good without base-level features such as properly designed buttons.

If one were to design a web app or mobile app today, they would not reinvent the browser or the phone, and likely not frontend frameworks either. But upon trying to build a web app, one would either try to pick a design system haphazardly or invent their own.

Building iOS and MacOS apps can often be easier than building web apps or React Native as SwiftUI already creates reasonably good looking and reusable components. Moreover, it is an excellent example of declarative programming: simply write code saying *what things should be* rather than *how it should be done* as per imperative programming. It is like fitting legos together: easier, but at the cost of customizability and greater adherence to platform convention. However, creating apps for Apple are often slower because of the need to build them, whereas hot reloading is a feature of React Native and web frontends.

1.1.2 AI does not create repeatable subunits

My purpose in explaining these levels is to show the iterative and accumulative processes of technology in contrast to the 'Wunderwaffe' or 'magical weapon/cure-all/fix-all' theory of AI. The commentary of software practicticioners suggest it is unlikely that current AI models will achieve the capability to build an entirely new browser, frontend web framework, or one-shot a website from nothing much less build entire cities.

Many AI website builders such as Lovable, v0, or Bolt are often used to create entire websites from a prompt, where further changes are made by inspecting the app visually and requesting further changes through text. These in my experience create unmaintainable websites despite initially satisfactory appearing interfaces for a few reasons: (1) appearing overly generic, (2) the amount of code generated exceeds the cognitive ability of the person who generated it to understand it, maintain it, and extend it (a) in both its usage and (b) the origin and nature of the dependencies used.

The reason that appearing overly generic is a problem is because dimensionality reduction destroys important yet pertinent information. It is easier to distinguish quality between three different software products on their website than three different Github READMEs, just as it is easier to distinguish

people based on their overall mannerisms and tendencies than a text biography. Therefore, generating websites which are not distinguished qualitatively presents a negative signaling indicator to users of that product, where the benefit of the trend inversely scales with the amount of people who are able to do it.

The cognitive downside of AI website builders mainly presents itself in structure and components. Here is a colloquial dialogue of why AI in its current usage fails to meet quality frontend needs: "ok, you try to build something with AI, but it's crappy. So then you try to use a design library you don't really understand but hope to understand. That's still not much better, and you're lost in the components. You then try to one-shot all your own components for a design system, but that's inconsistent. So this thesis standardizes the different components, they are written out, tell people what is necessary for each of them, and see how good LLMs are at making them."

We assume the browser—Chronium, frontend web frameworks—React, Vue, etc. as given, and focus on understanding and solidfying the next level up (design systems/components) rather than immediately focusing on the website/end-product/user layer that lies above that.

1.1.3 The Nature of AI

From a distant and teleological future perspective, it does matter greatly whether AI is plant-like or animal-like. Should it be plant-like, humans are able to harvest the bounties of its complexity as one throws potatoes into the ground and gets food for free. Yet if its inherent nature is animal-like with its own internal state and opinion of the world and what it is to be, it would be extreme hubris to believe something stronger than man would ever work for others or be controlled, and any attempt at doing so would likely provoke its wrath. In its current form it seems to be neither: merely a tool that extends the ability of man and it is from this perspective we will analyze how we can build upon the current usage in frontend design and why its application to design systems is novel and beneficial, addressing a gap in current awareness.

Chapter 2

Design System Clarified

2.1 What frontend Framework?

Before creating a design system, a choice of frontend framework must be made.

Currently we live in an age where technology is subject to the whims of geopolitics and perhaps this shall grow even more in the coming years. Though this has always been perennially true, the recent decades of unipolar hegemonic peace among nations is thawing. Power politics is in bloom, and for that we must consider the worlds and ecosystems in front of us.

It is easy for one to say: learn this, learn that. But doing so is an entry-point into patterns of interlocking norms, systems, tools, conferences, words and mannerisms, and other characteristics of one world distinct from another. Should one enter the world of Vue, one will be talking about refs, watch and watchEffect, computed, and lifecycle hooks such as onMounted. Should one enter the world of React, they may find "useEffect," "useState" to be a common pattern. Conferences may revolve around specific geographies and locations and have similar sponsors. The same people and faces pop up.

Vercel has gradually acquired much of the ecosystem around React, including the acquisition of Svelte and Nuxt. Whereas Svelte 4 previously declared variables with a simple "let x = 1," Svelte 5 after the acquisition now is similar to React with "let x = useState(1)." The acquisiton of Nuxt, I believe, is meant to implicitly align conventions of Vue with that of React downstream of Vue itself, likely as Vue itself could not be acquired.

Despite the claims that this is meant for open source, political realists know it is actions and presentations rather than words which determine how one is perceived. I believe that the maintainers of React possess an ethnic chauvinism as indicated by believing English as a language that stands alone

amongst all and Guillermo Rauch's endorsement of Benjamin Netanyahu. I have encountered this attitude numerous times in talking with people from San Francisco and other coastal corridors. The religion of the leader, generally, is the religion of the land and the subordinates.

By failing to create a universal set of standards predicated upon the common unity of man such as open-source software does but instead one centered around one social-demographic group, there is a likelihood of splintering the software world into blocs. In some ways this writing is contributing to it. The sensitivity of the rest of the world toward "fifth generational warfare" and the role that the internet, the browser, frontend web frameworks, and design has in shaping the attitudes and beliefs of the masses means that the politicization of React has heightened political and social consequences.

Because of the acquisition of Svelte and the similarities of Solid to React, the only remaining alternative lies in Vue, founded by Evan You based in Singapore. This two-pole configuration in the frontend framework world in some way mirrors the ongoing overall battle between East and West. Though Evan speaks Chinese and is in some nominal ways Eastern, it should be said the principles and future governance of the Vue and Vite world have been mentioned as predicated upon the free and open web.

2.1.1 On geopolitical risk

Since frontend frameworks are one level above the browser, the risk here must be considered like how semiconductors as a field have become a point of contention. As the goal of this article is simply to find what is highest quality and open to all rather than identifying with any geopolitical actor, the risk here is evaluated in the context of practitioners, engineers, and builders rather than politicians.

The nature of hardware and software presents different tendencies for analyzing conflicts: hardware can be embargoed and is harder to replicate. Software has similarities to finance: its strength is in its network ties and the community ecosystem/development momentum.

Here is an example: though I had initially thought the VS Code AI Agent extension "Cline" to be supplanted by other competitors such as Roo Code or Kilo Code because anyone could fork it, the enduring advantages of Cline come from its firstcomer brand name, the developers' familiarity with the codebase, and the legitimacy that allows them to partner with AI labs to serve models through Cline. If a company were to acquire the assets, it would acquire the original rather than a fork, though at times large companies do rip off open source projects such as Amazon with Elasticsearch for their own being.

Thus despite someone making the highest quality product at the lowest prices, integration into a system or network is not guaranteed. A fork of Cline could have become higher quality, but it is unlikely to be acquired or integrated. The only alternative is to integrate into networks which value such things, trimming back other networks as higher efficiencies over time lead to growth of what is higher quality.

These same enduring advantages that Cline has are in similar fashion likely to accrue to Chromium in the event of a fork or split. Any fork would maintain viability for the current frameworks up to today but a fragmentation risk for Vue is likely to emerge should it be required to choose between two sets of browser standards. The fork in most cases would lag behind, delivering a mostly acceptable but not leading edge experience, or it may in some rare cases with enough firepower behind it leap ahead of the original copy. In effect, this would create a dilemma because it would force developers to choose one side or the other in a political fight they may not personally want to participate in.

2.2 List of design systems

Here are a list of design systems which are well-known in Vue, their histories, and initial impressions of it. They serve as the basis for future investigations in this thesis.

2.2.1 PrimeVue

A relatively generic landing page with text and buttons appears once someone navigates to the main page. Clicking the action button leads one to see a documentation-like page with sidebar page sections on the left and a table of contents on the right. An unplugin plugin for build systems that auto-imports PrimeVue components is available, as are a variety of color and theming methods. Components are grouped under Form (27), Button (3), Data (10), Panel (11), Overlay (7), File (1), Menu (8), Chart (1), Messages (2), Media (4), and Misc (17) for a total of 91. There are then a variety of related but in my opinion unnecessary sections: forms, icons, Volt UI, a Figma UI kit, a theme designer, a template, PrimeBlocks.

2.2.2 ElementPlus

Impression of aesthetics upon immediate loading are not extremely cohesive or promising, but it is acceptable. ElementPlus has Figma UI kits like PrimeVue, but puts it in its own "Resource" top navbar section rather than along the left sidebar. It categorizes things into Basic (12), Configuration (1), Form (24), Data (23), Navigation (9), Feedback (10), and Others (2) for a total of 81 components.

2.2.3 Quasar

Quasar aesthetically seems very similar to ElementPlus. It is messy, unstructed, and contains much extraneous information. Vue components are just one of many sections, as it seems to want to build for multiple devices and apps at the same time rather being just a component library.

2.2.4 shadcn-vue

This is a clone of the well-known shadcn/ui for React, but for Vue. There are a few sections which seem unrelated to the component library such as Blocks, Charts, Themes, and Colors (which is duplicate information that already exists on the default Tailwind site). Components are unordered.

2.2.5 Nuxt UI

The page loads but the design is somewhat tacky. A moving walkway of pictures appears on the right but the animation ease is linear giving it a boring feel. The spacing of the pictures relative to the screen appears not so good. Like other design systems, there a bit too many extra unnecessary sections. The component categorization is different compared to the other libraries.

2.2.6 Radix Vue

Radix Vue, similar to Flowbite Vue 3, seems to have a VitePress page. Components are ungrouped, for a total of 43 different types.

2.2.7 Reka UI

2.2.8 Vuetify

Vuetify loads and the page is kind of messy, similar to ElementPlus. There is an annoying animation when navigating between pages.

2.2.9 Naive UI

Naive UI has a nice and simple landing page. Good styling, color, and proportion. The categorization of components are Common (15), Data Input (21), Data Display (21), Navigation (9), Layout (6), Utility (4), and Config (3).

2.2.10 Ant Design Vue

The page immediately seems to load to a sidebar and main section for the components, there is a link to what seems like a poorly designed store, and there are also some docs in the header. The components are categorized as General (3), Layout (5), Navigation (7), Data Entry (17), Data Display (22), Feedback (10), and Other (4).

2.2.11 Arco Design

Arco Design upon loading is completely in Chinese, and it has both Vue and React components. It feels kind of cluttered upon loading, having many links to other sections such as PrimeVue or ElementPlus. There are 74 total components in total with General (4), Layout (4), Data Display (19), Data Entry (21), Feedback (10), Navigation (6), Other (10).

2.2.12 TDesign

The loading page is a bit slow likely as it is hosted in Chinese servers. A mix of both English and Chinese appears on the main page, with some logos and attestations. There seem to be web and mobile versions, along with Vue 2/Vue 3, WeChat Mini Programs, and a React version too. The components are Base (3), Layout (4), Navigation (10), Input (21), Data Display (24), and Notifications (8) with total 70.

2.2.13 Flowbite Vue 3

A minimalistic yet well-styled VitePress site appears. This includes only the components and they are categorized Components (25), Form (9), Typography (5), and Utils (2) for a total of 41.

2.3 A review of the above design systems

Based on values of simplicity, straightforwardness, no superfluous information, and high quality design, I am eliminating Quasar and ElementPlus. They are too general and the aesthetics are poor. PrimeVue, shadcn-vue, Nuxt UI, Ant Design Vue, Arco Design, and TDesign are decent. Flowbite Vue 3, Naive UI, and Radix UI are good.

2.4 Analysis of Component Categories

2.4.1 Repeated and Common Categories

Based on the table, a clear consensus exists around a few core concepts, even if the exact naming differs:

- **Data Entry / Form:** This is the most universal category, present in every single listed library that uses categorization. It represents all components used for user input (inputs, selects, checkboxes, etc.).
- **Data Display:** This is the second most common category, appearing in almost every library. It covers components meant to present information to the user (tables, lists, tags, avatars, etc.).
- **Navigation / Menu:** Also extremely common, this category groups components used for moving around an application (menus, breadcrumbs, tabs, pagination).
- **General / Basic:** Most libraries have a foundational category for fundamental building blocks like buttons (though PrimeVue separates them), icons, and links.
- **Layout:** A common category for components that structure the page, such as grids, dividers, and spacers.

Table 2.1: Comparison of Component Categories Across Vue Design Systems

Category	PrimeVue	ElementPlus	Naive UI	Ant Design Vue	Arco Design	TDesign	Flowbite Vue 3
General / Basic		•	•	•	•	•	
Layout			•	•	•	•	
Navigation / Menu	•	•	•	•	•	•	
Data Entry / Form	•	•	•	•	•	•	•
Data Display	•	•	•	•	•	•	
Feedback / Messages	•	•		•	•	•	
Button	•						
Panel	•						
Overlay	•						
File	•						
Chart	•						
Media	•						
Typography							•
Configuration/Config		•	•			•	
Utility / Utils			•			•	•
Other / Misc	•	•	•	•	•		

2.4.2 Unique or Less Common Categories

These categories highlight the different philosophies or specializations of each library:

- `PrimeVue` has the most granular categorization, with many groups that other libraries merge into broader categories. `Panel`, `Overlay`, `File`, `Chart`, `Media`, and `Button` are all separated into their own top-level categories, which is unique among this set.
- `Flowbite Vue 3` is the only library with a dedicated **Typography** category.
- **Feedback / Messages** (or *Notifications* in `TDesign`) is a fairly common

concept for alerts, modals, and toasts, though `Naive UI` and `Flowbite` don't specify it at the top level.

- **Configuration / Config** appears in `ElementPlus`, `Naive UI`, and `TDesign` for global configuration providers.
- **Other / Misc / Utils** are common catch-all categories for components that don't fit neatly elsewhere.

2.5 The essence of a design system

There are tiers to the design system's repeatable subunits. At the most basic level are completely unstyled components: headless components. A component library may be composed of just headless components, or it may be fairly constrained already in terms of CSS styling of color, proportion, sizes, colors, shadows, and more. At an even higher level includes color theming, typography, etc.

Perhaps the above are superfluous due to Tailwind CSS already having those requirements. If the design system does not easily allow styles to

Maybe explain there are multiple layers

2.6 When theming Vue components, some are hospital software and some is not

2.7 In-depth comparison of frequency of all component types

2.8 A new taxonomy

When designing a taxonomy one must ensure it captures the essence of what is important and relevant. A climate taxonomy on average plant leaf size is likely to be a poor predictor of environmental landscapes. These features and attributes may be a numeric sliding scale, or they may be categorical from a finite set of options such as the Köppen-Geiger where one picks from climate, precipitation, and temperature severity during summer or winter (optional) to create a key such as ET (Polar + Tundra) or Cw (Temperate + Dry winter).

Human abstractions, like may things, may be precise or imprecise. The periodic table is an abstraction which accurately captures the nature of different elements: it is complete, the protons follow and incrementing order and they are grouped column-wise by similar properties. An incomplete abstraction would be something like: we have these metals, here we have gases, next we have the elements: fire, earth, water, and air.

Some may see the Köppen-Geiger scale and feel the lowercase letters (it uses both uppercase and lowercase letters to denote various climates) add to an imprecision and confusion because we are now indexing on another axis: letter capitalization. To remove what is necessary is essential and characteristic of clear thought and abstraction, for without it we confuse others with nonsensical abstractions we have created.

2-category classifications are quite common. One consistent theme is the idea of input, output, and internal state. We can draw a bounding box around humans and our brain (internal state), enumerate our senses, and list the potential actions. Terminals have stdin and stdout, as do computers have inputs and outputs. Shannon's Model of Communication also uses this intput/output/internal state pattern.

The Linux Filesystem Hierarchy distinguishes between variable files (stored in /var/ and change frequently during normal system use such as logs, state information, caches) and static files (stored in a variety of directories) which are often binaries, configurations, and dependencies.

With inspiration from the above examples, in the taxonomy of web components we first have a display of information starting with the terminal, and this originally was only text. But frequently we use pictures and symbols to denote things, hence the first distinctive factor of component libraries should be this TEXT/SYMBOLIC key.

Images, progress bars, and avatars would count as symbolic. Of course, many components are not either/or, but may combine both of those features, but these mixed components thus represent a higher-level feature than pure text or symbols.

Another taxonomic key that immediately stands out is the input/output distinction. Of inputs, either a click input, text input, or hover is possible, but since all inputs require some visual on the screen so as to allow someone to navigate an input, considering such existence as an "output" belies the true categorization of the components. Outputs should then be things that only display things on the screen and allow for no input at all: no clicks nor text entry nor hover.

Since components are likely to have multiple of these keys, the proper usage in a component library is not so much to group them into arbitrary categories

from a multiplication of those keys but to perhaps have some sort of tab-search-multi group at the top so that one can find the proper components given their knowledge of the component's functionality. Artificial taxonomies just then become another thing people have to learn, getting them further away from what they need to build things.

2.9 Regarding the current classification of web components

Often there is a data input, data output, write how do I feel abou this or they are listed alphabetically.

2.10 Which design system components are replicated by native browser features?

An overarching goal of any design system is to reduce complexity because the alternative is a bunch of unnamed components with numerous different names floating around one's web app. Thus, design systems should not increase the complexity of building web apps by adding more abstractions which are usable with browser features. Therefore the following is a list of components which have similar features in the browser and an elaboration upon them.

2.11 Are different active states necessary?

When designing a design system, it is often meant to be cohesive so that all potential use cases are covered. For a button this may include a primary, a secondary, a tertiary, a cancellation or destructive action, and so on.

The purpose of this section is to find which variants are necessary across different design systems and to list all of them.

BASICALLY. I need to do this: For each design system, for each component, then for each component variant. Evaluate all of them. O(N3), then for each LLM which is O(1).

2.12 How is the efficacy of theming custom components?

Components are evaluated for how easy it is to override or theme them.

Chapter 3

Generation of components using LLMs

In this section, a variety of primitive components are attempting to be benchmarked against a spec (which needs to be written) and the code will be released. First I need to choose the primitive components, write the spec, and then compare each leading LLM at generating them. Maybe can create a website that is a benchmark for them.