Vue Report
Amsterdam 2022

SPECIAL EDITION FOR THE LARGEST VUE CONFERENCE IN THE WORLD

BROUGHT TO YOU BY MONTERAIL

What's inside

01.

Preface

What’s special about this edition of the Vue report? We’re sharing the idea and goals behind it to let you know what to expect down the road.

Since its first release in 2014, Vue.js has been increasing in popularity and users. It has been a long journey to become what it is now. From a small project, it turned into a mature framework used (and loved) by hundreds of thousands of developers all over the world.

What's the secret sauce to making Vue so universal?

This report shows how Vue can be used to solve interesting development challenges, how it changed since the last update, and what experts say about their experience with this framework.

The idea behind the report comes from our knowledge-sharing approach and willingness to support communities; values that have been with Monterail for years now. As an official Vue.js partner (and author of four Vue.js reports), we want to support and promote Vue.

We want to provide expert-based content for other experts. To inspire and give food for thought. For Vue lovers by Vue lovers - simply to enjoy.

Enjoy!

Joanna Staromiejska-Drwięga - *Editor and Writer*

CONTENT MARKETING TEAM LEAD AT MONTERAIL

Błażej Cepil - *Writer*

DIGITAL MARKETING TEAM LEAD AT MONTERAIL

Natalia Leśniak - *Designer*

WEB DEVELOPMENT & DESIGN LEAD AT MONTERAIL

Contributors

Evan You

Creator of Vue.js

/youyuxi

Anthony Fu

Core team member
of Vue, Nuxt and Vite

/antfu7

Lucie Haberer

Developer Experience
Engineer at Prismic

/li_hbr

Tomasz Kania-Orzeł

Head of Technology
at Monterail

/KaniaOrzel

Szymon Licau

Principal Engineer
at Monterail

/szymon_licau

Filip Rakowski

CTO & Co-founder
at Vue Storefront

/filrakowski

Carlos Rodrigues

VueJs Core Team
Member

/pikax_dev

Daniel Roe

Framework Architect
at Nuxt Labs

/danielcroe

Artur Rosa

Frontend Architect
at Monterail

/rosickeyy

Ramona Schwering

Software Developer
at Shopware

/leichteckig

Maya Shavin

Senior Software
Engineer at Microsoft

/MayaShavin

Vue Amsterdam

The Vuejs Amsterdam team is committed to fostering and creating learning and connection opportunities for the Front-end Developer community. One way of doing it is to organize events and conferences like Vuejs Amsterdam - the world's most special and largest Vue Conference.

Vuejs Amsterdam's organisers love for Vue, community building and education is also spread throughout it's intimate community events under the name Vuejs Roadtrip in various cities throughout Europe like Barcelona, Berlin and Paris and numerous Free Community Meetups. Other conferences include JSWORLD Conference, React Miami as well as 4 other major conferences. Our work for the Front-end Developer Community has seen over 6,000 people gain knowledge in the last 5 years.

Since Vuejs Amsterdam started in 2018, it has been a sold-out conference. Having Evan You, the creator of Vue, opening the event attracts thousands of people from all over the world each year.

VUE AMSTERDAM IN NUMBERS

20+ Vue.js Core Members, Library Authors & Experts

2k Attendees

50+ Audience Countries

2 Full Days of Talks

Why Vue?

We work with many developer communities and collaborate with many projects in the JavaScript ecosystem, however, the Vue community quickly differentiated from the rest because of how friendly, open and cohesive it has been through the years. When we first started Vuejs Amsterdam (back in 2018), we did it because we believed in the framework and wanted to see it succeed.

It's really remarkable how anyone with absolutely zero Vue.js experience can be quickly productive with it and understand our implementation after following some initial tutorials or reading the excellent documentation. On top of that, we use it in combination with Nuxt which greatly reduces the cognitive load when onboarding a new teammate since all architectural decisions have not only been very carefully considered by the Vue.js and Nuxt core teams, but they are consistent across all projects where we use Vue.js + Nuxt.

Luke Thomas

Founder of Vuejs Amsterdam

02.

What's new in Vue

The summary of Vue's exciting developments from the perspective of the Principal Engineer at Monterail.

Vue had quite a lot of exciting developments throughout last year. The latest big release of version 3.2 - „Quintessential Quintuplets” has continued to improve performance and add features for Single File Components. Many libraries embraced Vue 3 and added support for it, while others have compatible releases on the way.

Here are some of the exciting developments we could observe in Vue ecosystem:

We had the first stable release of Vite - a new kind of build tool for frontend development. Vite brings a combination of blazing-fast development experience and highly optimized production bundles. It's also framework agnostic and anyone can use it! Vite is the new recommended choice, while Vue CLI enters maintenance mode.

Following the naming convention, we have also seen the release of Vitest - a new unit-test framework powered by Vite, which brings its blazing fast speed to subsequent test runs, making it a fantastic choice for a TDD approach.

Pinia released its first stable version and it is now the recommended library for state management in Vue 3. Introducing a simpler API, proper TypeScript support, and utilizing hot module replacement, it's a huge step up in the development experience. While Vuex is still compatible with Vue 3, it is now in maintenance mode.

Volar was released as the new official development tooling for our IDEs. It's a huge step up in terms of full TypeScript support inside Single File Components and performance. It also provides cross-component props validation and type checking in our templates out of the box.

Recently Vue Version 3 became the new default! Potentially requiring some actions in projects using Vue version 2 as the latest version will now point to Vue 3.

With this, we have also seen the release of new Vue docs overhauling the design, UX and introducing new guides as well as a feature to toggle between Options API & Composition API. This version also introduced an interactive tutorial that walks you through all the major features of Vue. It’s never been a better time for newcomers to try and learn Vue.

With all the latest additions it looks like the Vue ecosystem has changed for the better along with the introduction of version 3. We see a new generation of tooling which brings improvements in all development aspects. Development experience has significantly improved thanks to Vite, Vitest and Volar.

Vue itself also continues to introduce many useful features and improve its performance. New, overhauled documentation is also a fantastic way of learning all the new and exciting things recently introduced. We see that Vue keeps on improving while also delivering the promise of a solid technology choice, especially for new projects going into the future.

Let’s see how Vue 3 and its evolution look from the perspective of its creator – Evan You.

Szymon Licau

Principal Engineer
at Monterail

03.

State of Vuenion 2022 by Evan You

Evan You

Creator of Vue.js

The creator of Vue.js summarises the breaking changes in Vue 3, shares the details behind the process of upgrading to the default version and envisages the future for Vue.

Q

What has changed in Vue 3 since the last update?

Evan: We've just shipped our brand new documentation in February 2022 which essentially marked the completion of the soft launch process. When Vue 3 was released it was really just the core, but Vue as a whole has grown into the full ecosystem over time. The framework consists of a library like a router, state management, build tooling, dev tools, extension, IDE support. All of these things took a lot of effort to bring up to date.

So now we have a new version of basically everything. We shipped brand new docs, replaced Vuex with Pinia, the latest recommendation for state management, our build tools are now powered by Vite. Our new IDE extension is Volar which provides much better TypeScript support and experience. We also shipped important DX improvements like <script setup>, and have more down the line.

Q

How has TypeScript influenced the development experience with Vue?

Evan: TypeScript is obviously on the rise so any modern framework has to be designed with it in mind. Even if you don't use TypeScript, Volar is able to leverage Vue typing and give you hints, removing this mental burden of remembering what type of variable it is. All the type inference and checks also work in the template. These simple mistakes happen when you don't have type checking. Using TypeScript also makes you confident when refactoring a large codebase because you can easily see changes - the tools will immediately show the errors for you to fix.

Especially in a team environment working on a large project, using TypeScript with Vue will greatly improve the robustness of the code that you write every day.

Q

It's been a year and a half since a stable version of Vue 3 made it the default. How did the process look?

Evan: For the initial release of Vue 3, we rewrote everything with TypeScript but it is only now complete as Volar became the official extension. The documentation was completely revamped. All of the sections were either revised or rewritten and they work with Options API and Composition API and allow you to toggle between them. So the learning experience and day-to-day experience got upgraded. Each piece was a project on its own which is why the whole process took so long to finally get everything in place. Now we can say that Vue 3 is ready.

Q

Is such a deep transition process necessary?

Evan: It's not always necessary but when we think of upgrading, there's always a tradeoff between building upon the existing codebase with incremental improvements (and building more technical debt along the way) or doing a big-bang rewrite.

Basing on Vue 2 codebase and making sure everything's 100% compatible, we would carry along the baggage of a lot of libraries that relied on internal behavior that's specific to Vue 2.

If we carried with it forever, it would seriously limit the amount of innovation we could implement in the long run.

So we felt like a big-bang rewrite is the right direction for us. The situation with Vue 3 was sacrificing some things in the short-term and breaking some libraries but it was the cost we were willing to pay in order to get rid of the past baggage. Now we have a clean state and it will pay dividends down the road.

Although React or Ember almost seamlessly introduce new versions, it causes a lot of maintenance overhead on their side. They have a good system of slowly adding new features, phasing out the old ones, which we can learn from. But at the end of the day, there's no absolutely correct answer in engineering. When it comes to upgrading, it's about the trade-offs that you are willing to make.

Q **Looking back, would you change anything?**

Evan: We probably could have done a better job but I'm not sure we would have achieved the same level of improvement. That's the tradeoff here. I would probably focus more on reducing the breakage of some small edge cases, small behavior mismatches that we only discovered after we started rolling out Vue 3. We would have tried to cover it a little bit better. But it's all in hindsight - you can't foresee some things until they happen.

Q **How will Vue 3 affect businesses and why should they make a shift?**

Evan: I would suggest evaluating the situation on a case-by-case basis. If you have a greenfield project, it's Vue 3 all the way. It doesn't make sense to stick to Vue 2 anymore with everything new in place. Nuxt 3 will go stable in a few months which is another reason for transition.

With an existing project in Vue 2 however, businesses still need to weigh the pros and cons of the upgrading.

If Vue 2 works and it works well, making a change might not be a cost-effective solution. It really depends on how much you want to leverage the new features. Vue 2 with Options API is still completely viable, and we plan to make it easier to use Composition API / <script setup> easier in Vue 2 with 2.7. But if you feel like switching to Composition API now will objectively improve productivity, don't hesitate to change.

Q **Vue is used for all types of applications and company sizes which shows it answers real-life cases. How did you work on making the framework so universal?**

Evan: The initial version of Vue was really designed to lower the barrier to entry. We made it easy to adopt, integrate into existing systems - we still stick to this approach and value it a lot. Over time, we slowly add pieces to support more advanced projects like Single Page Applications routing, state management, TypeScript support.

At first, we had more users with simple use cases and smaller applications. Along with more features and tooling, the pool of Vue users grew but initial users stayed and can still use it the way they like. So I guess that's our secret sauce - to cover the full spectrum.

It's been a long journey since 2014, with lots of challenges but every time we added something, we tried to sustain the core experience of using Vue.

Q **How do you feel five years after the first State of Vue? Back then, you defended that Vue is no longer a "one-man army project". What is Vue accused of in 2022?**

Evan: I don't really think people "accuse" us of anything anymore. If we are talking about complaints, there are still areas in TypeScript support that can be improved. We've done a pretty good job supporting template expressions. The only part we're still working on is scoped slots and generic components which are generally rare use cases, relevant to really advanced developers. We're trying to cover it but it won't be a deal-breaker for most users.

Q **Currently the world of frontend frameworks looks quite stable and differences are rather little. Do you think that there is still room for something which will change the way we think about front-end?**

Evan: There are a few new things worth mentioning like Svelte and Solid, especially the way they leverage the compile time optimization. I think frameworks moving more into a compiler is a general trend. What

I take away here is the "be a smart compiler" approach. Vue has already done a lot in this direction, but also still has a lot of further potential to be unlocked. Another aspect is the new wave of innovation, experimentation happening in full-stack frameworks about reducing the overhead of server-rendering and hydration. When we're in a full-stack environment, there are lots of questions about how to leverage both server and client to make the initial load more efficient, ship less out and do less hydration work. We're exploring it as next in line.

Q **Let's look broader than Vue, what major challenges are waiting for the frontend world or even web development?**

Evan: When we look at the web development world, there are various apps there - simple landing pages, fully interactive apps, backend dashboards which are very complicated so you don't even expect it to load fast. There's also this performance-sensitive case like e-commerce. In general, the challenge is to balance developers' efficiency and the end performance. In most of these areas, we've reached a pretty good place, except for content-intensive e-commerce.

We'll need more vertical solutions that could handle both the backend and frontend. That's where you can start doing interesting things. We need to stop thinking from a pure client-side perspective or a pure backend.

We already see some new things here like Remix or Qwik. A lot of these require controlling from a compiler to the server to the client. And all of them have to play together to give you a fully integrated solution that's optimized end-to-end. You still write relatively easy code and get optimal performance at the end. This is going to be the biggest challenge - which solution can best integrate all three parts together? I think this is a direction where we can work closely with the Nuxt team and other members in the Vue ecosystem to explore.

Q

Let's look 5 years into the future. How do you see Vue?

Evan: I tend to think of Vue as an ever-evolving platform. We probably won't do a "Vue 2 to 3" type of upgrade in the next five years because Vue 3 is a solid enough foundation to build upon for quite a while. We'll continue experimenting with the compilation strategy as the advantage of Vue is a really flexible reactivity system.

We have a compiled single-file component and it can be compiled into different output. You can change the output and the source code stays the same. We hope for users' experience to be stable throughout this time. From the development experience perspective, we want it to stay stable, but the compiled approach allows us to potentially ship big improvements under the hood. That's our goal.

04.

Vue Today

What's Vue's current position on the frontend landscape? How fast is Vue growing? The analysis of dozen of trusted sources and key takeaways.

Let's take a look at the numbers. What has changed since 2021 and what's the current position of Vue on the framework landscape? How fast is Vue growing?

We've reviewed over a dozen trusted sources and compared the results to previous years as well as compared Vue to other popular JavaScript frameworks. This data allowed us to establish an overview and draw conclusions. (Kudos to Tanguy Krotoff who prepared this data comparison that allowed us to peek into some historical records).

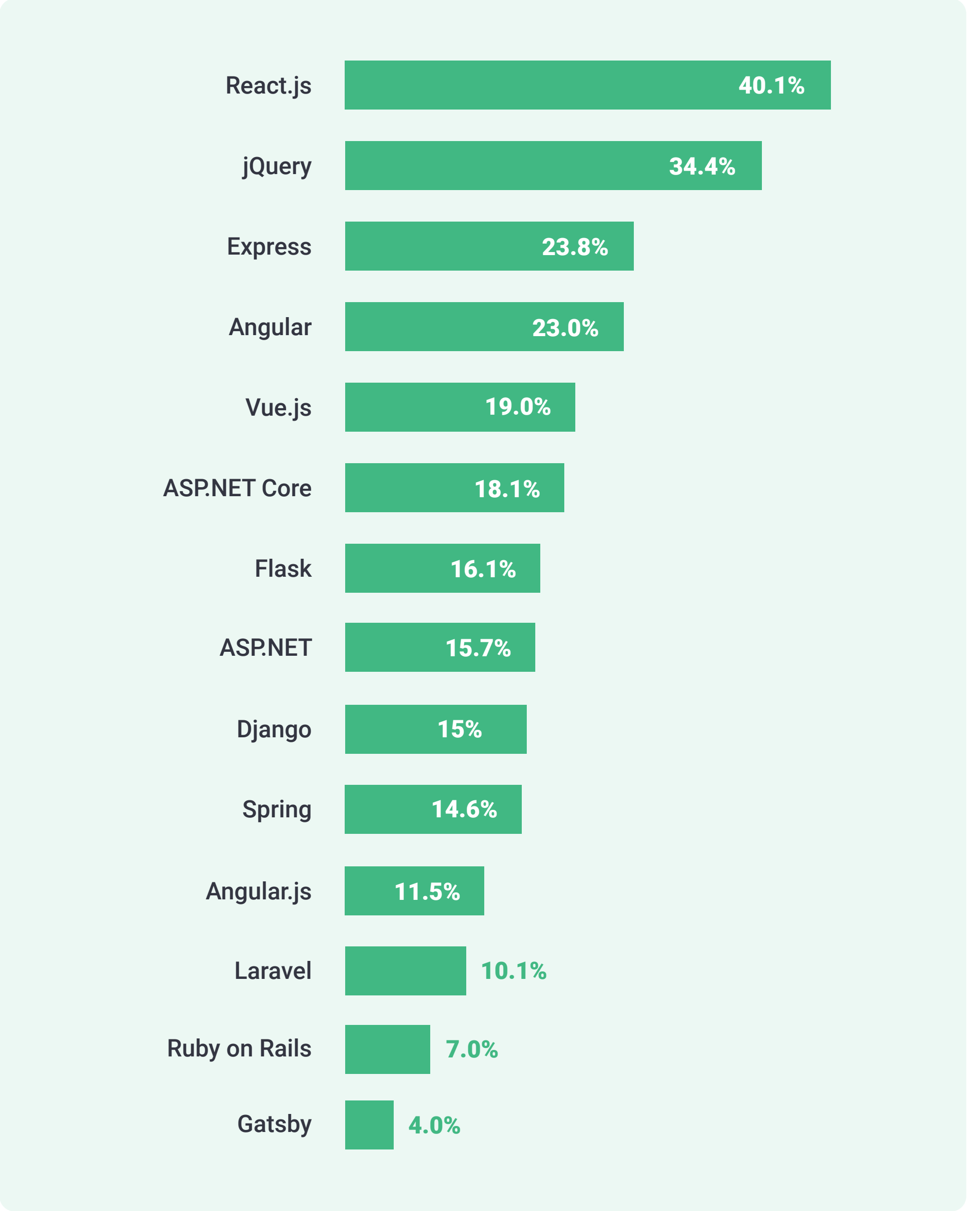
Although the numbers are not consistent across sources because of different survey samples, we can say with certainty: Vue is steadily growing in popularity. It does provide some unique (compared to React/Angular) possibilities like incremental implementation into an existing project, or ease of creating and maintaining small apps (not to mention beginner-friendliness). You can build long-lasting products with it, since it's only increasing in popularity and use where in some cases the two main competitors have seen a slight downside curve.

Vue manages to do all this without the corporate backing that Angular (Google) and React (Facebook) have. Some of the sources we used also don't consider the use of Vue in China where it's quite popular. Taking all this into consideration allows us to conclude that Vue is alive and kicking strong.

Let's dive into the details.

Statista

In August 2021, Statista asked 67,593 developers about their framework of choice and these are the results. Vue moved up two places since 2020 and came in 5th with 19% of respondents opting for it.



Most used web frameworks among developers worldwide, as of August 2021 (source: www.statista.com)

HackerRank

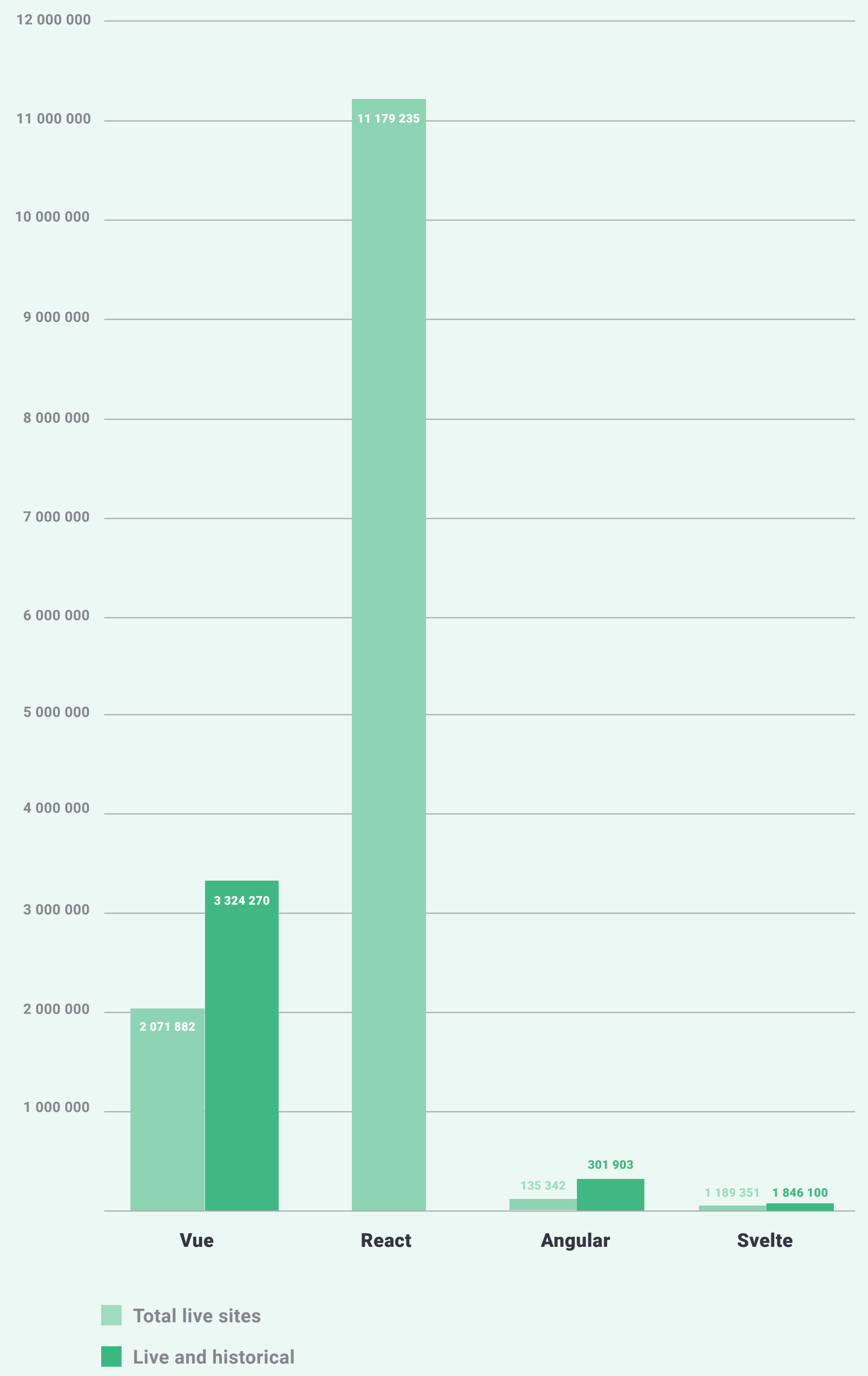
In 2020 (the latest data we have is still from 2020), HackerRank reached out to 116,648 developers to ask about the same thing as Statista did, but they also compared the results with previous years. Vue came in 8th but it showed a steady growth of interest among the HackerRank-related developers.

	2020	2019	2018
AngularJS	1	1	1
React	2	2	3
Spring	3	3	2
Django	▲ 4	6	6
ExpressJS	▼ 5	4	4
ASP	▼ 6	5	5
.NETCore	7	7	7
Vue.js	▲ 8	9	10
Ruby on Rails	▼ 9	8	8
JSF	10	10	9

(source: www.info.hackerrank.com)

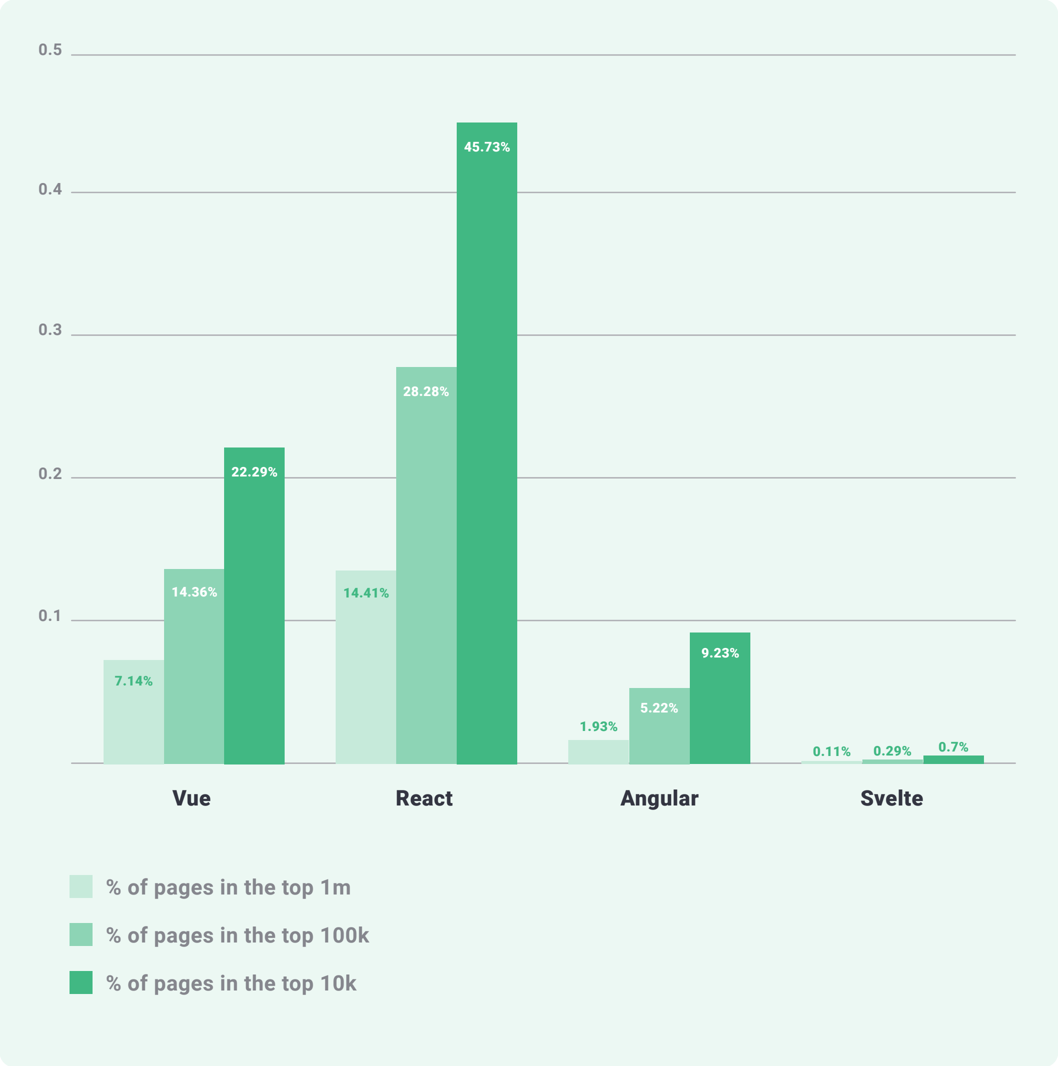
BuiltWith

Looking at the current (May 2022) Internet landscape, we clearly see that Vue, although not being the most widespread, is steadily growing. Since January 2021, Vue.js has doubled the number of live and historical websites and more than doubled the number of live sites exceeding 2 million already.



Number of websites built with (source: www.builtwith.com)

When it comes to a share in top websites, Vue built far more of those in the top 1m than Angular. And while it built fewer big-hitters in the top 10k when compared to React, the share increased from 4,99% (in 2021) to 7,14% making it a pretty significant contribution.

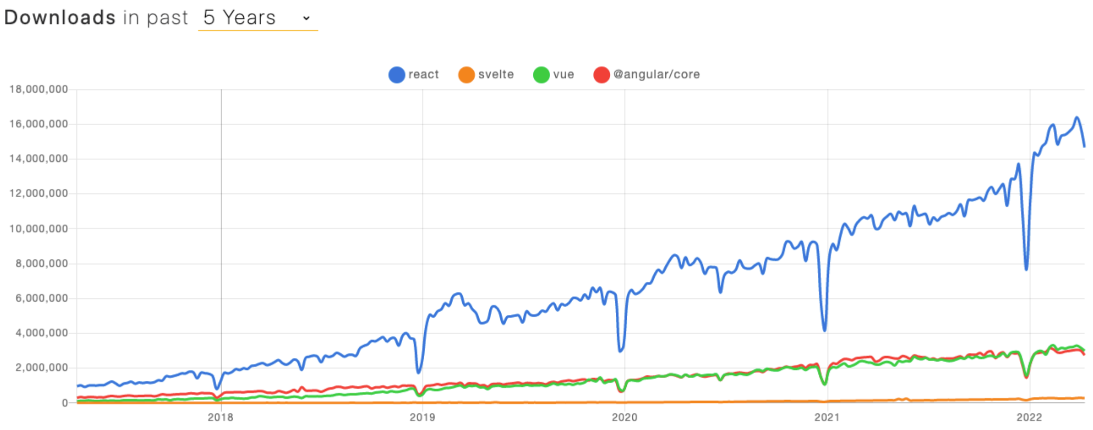


Share in top sites (source: www.builtwith.com)

	Total live	Live and historical	Top 1 M	Top 100 K	Top 10 K
Vue	2 071 882	3 324 270	7.14% 71 363	14.36% 14 357	22.29% 2 229
React	10 945 991	no data	14.41% 144 101	28.28% 28 283	45.73% 4 573
Angular	135 342	301 903	1.93% 19 348	5.22% 5 220	9.23% 92
Svelte	20 922	29 636	0.11% 1,065	0.29% 291	0.7% 70

NPMtrends

NPMtrends says that the use of Vue has been growing steadily in the past five years, maybe less so than React's but it's going hand in hand with Angular. Svelte is still crawling with total downloads of npm packages with little over 300k.

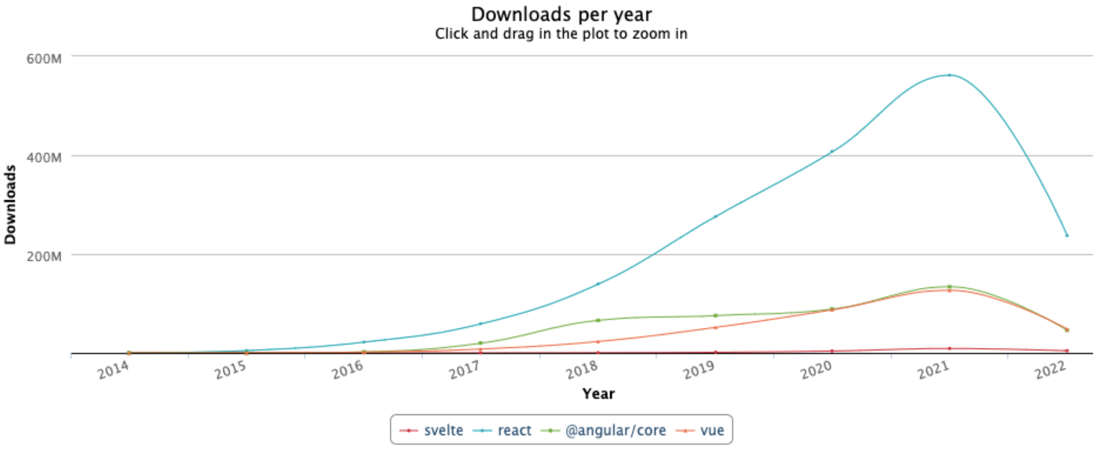


Downloads in past 5 Years ⌄

(source: www.npmtrends.com)

NPM-stat

NPM-stat shows the number of packages downloaded per year and the results confirm the growth trends from previous sources. However, take into consideration that the graph below concerns only the first four months of 2022 so we need to wait a little longer to get the full picture.



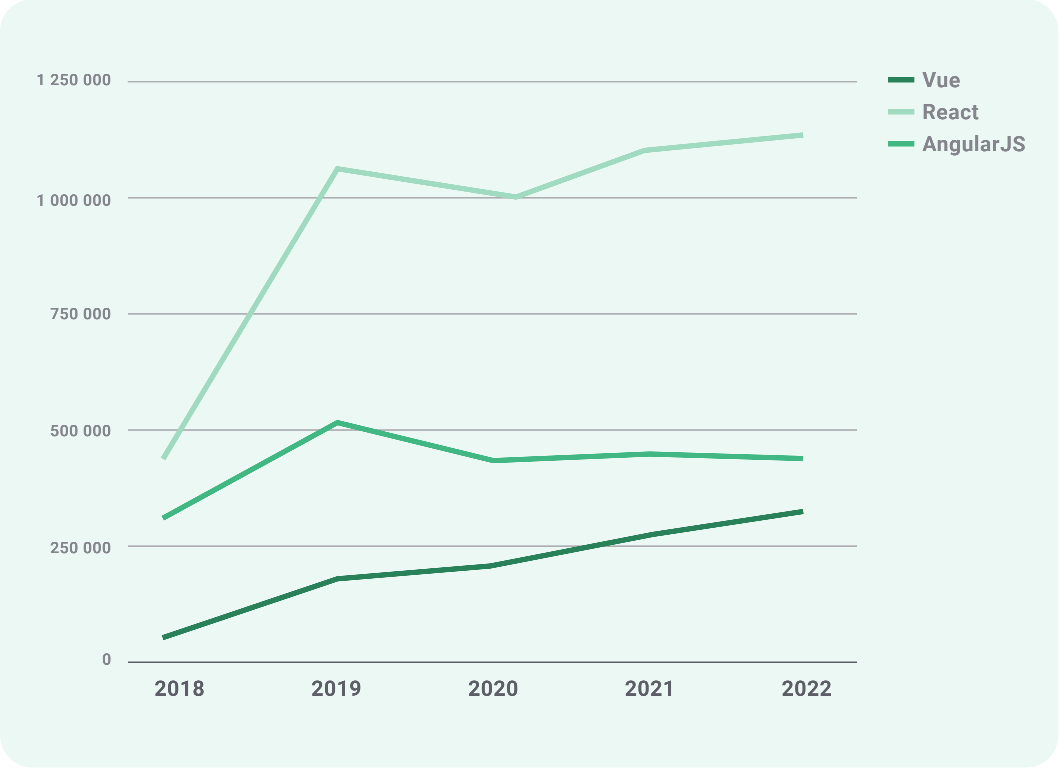
Total number of downloads between *2014-12-12* and *2022-04-22*:

package	downloads
react	1,707,850,174
@angular/core	431,698,170
vue	346,506,715
svelte	18,648,508

(source: www.npm-stat.com)

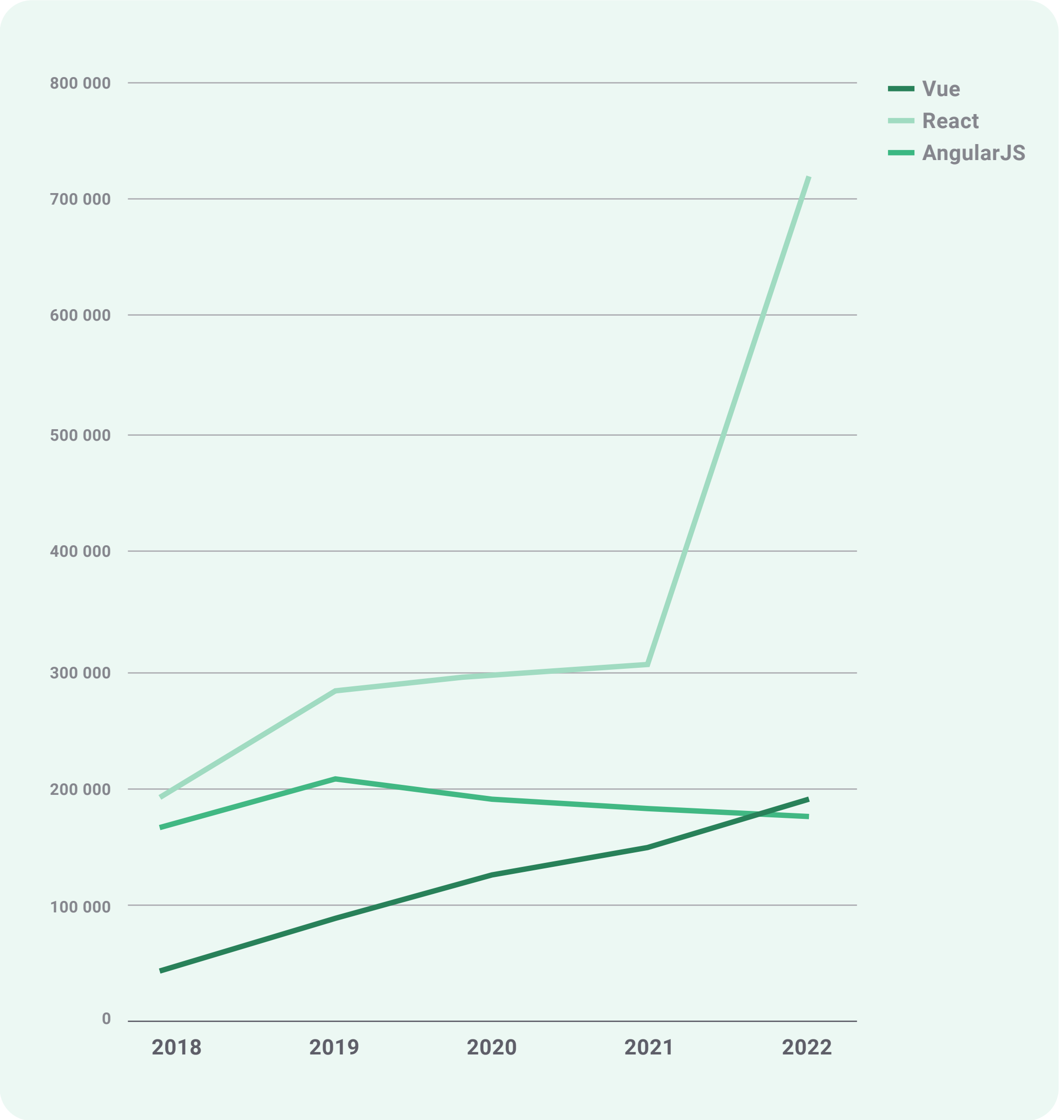
SimilarTech: market share & web usage statistics

A look at SimilarTech brings some interesting findings. The number of websites and unique domains built with Vue continues to grow from the very beginning. Angular faces slowdown or dips while the number of React-based domains has more than doubled. Historical context here clearly shows a growing trend both for Vue and React.



SimilarTech: number of websites (source: www.similar-tech.com)

	2022/04/06	2021/01/07	2020/09/22	2019/12/12	2018/12/16
Vue	298 670	220 538	195 214	157 831	54 881
React	1 279 596	1 126 095	1 005 214	1 069 073	420 066
AngularJS	368 054	384 515	378 038	517 701	325 339

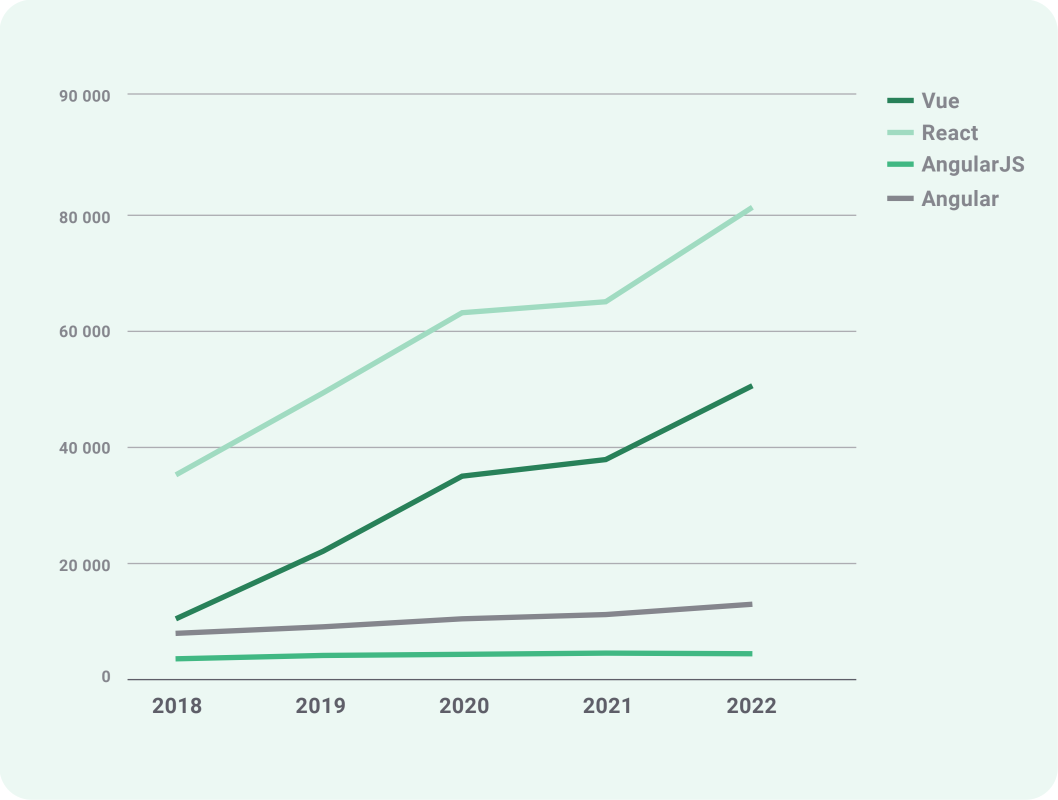


SimilarTech: unique domains (source: www.similar-tech.com)

	2022/04/06	2021/01/07	2020/09/22	2019/12/12	2018/12/16
Vue	194 403	140 184	121 748	91 509	40 033
React	709 461	313 355	296 347	287 997	196 048
AngularJS	182 889	188 734	189 590	203 614	171 570

GitHub dependents

The number of apps that will not run without Vue continues to grow showing a 46,52% increase since 2020. What's more interesting — Vue's usage on GitHub has not slowed down, whereas Angular 2+ did and in the case of AngularJS the number is even lower than the year before.

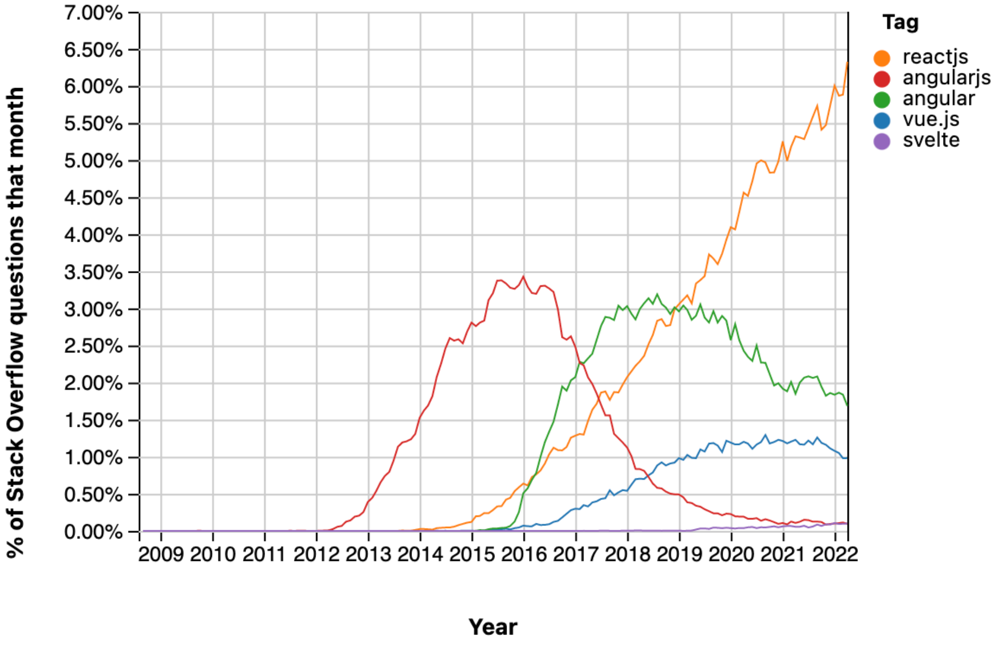


Number of dependents (source: www.npmjs.com)

	2022/04/06	2021/01/07	2020/09/20	2019/12/12	2018/12/16
Vue	53 010	36 179	32 101	21 575	9 792
React	84 262	66 033	61 870	48 718	32 331
AngularJS	4 091	4 125	4 112	3 959	3 693
Angular 2+	12 236	11 246	10 873	9 610	7 555

Stack Overflow questions

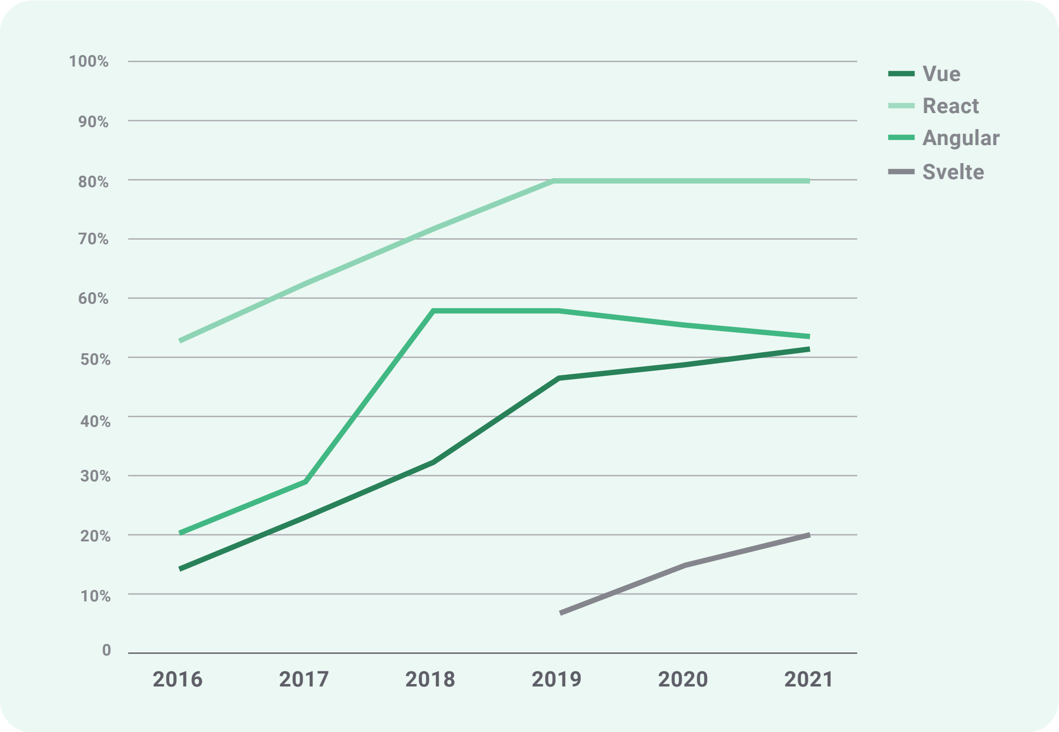
The questions asked on Stack Overflow increase in numbers for Vue and React. More so for React, but this could be explained by the fact that Vue is more beginner-friendly and is lauded for its comprehensive documentation. Add increasing popularity (for both Vue and React, drawing in more beginners) to this rationale and we get a quite reasonable explanation. There has been noticeably fewer questions about Angular and AngularJS since around 2017 and a minimal share of questions asked about Svelte itself.



(source: www.insights.stackoverflow.com)

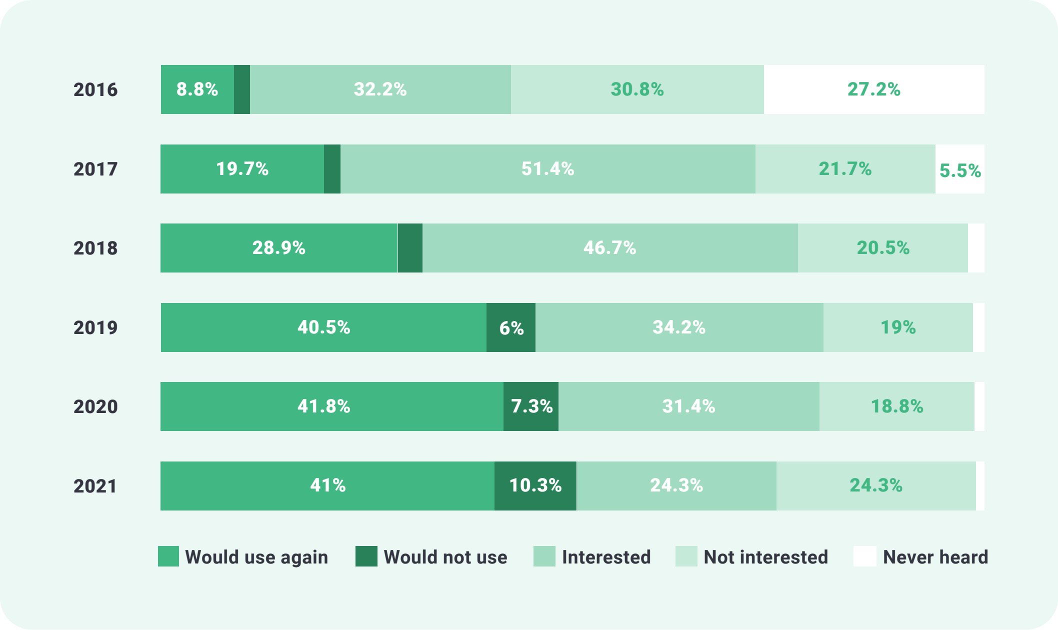
State of JS

As authors of State of JS 2021 examined, there's a growing upward trend in usage of Vue.js by developers that reached 51%, the highest result since the beginning of tracking.



Usage of front-end frameworks (source: www.2021.stateofjs.com)

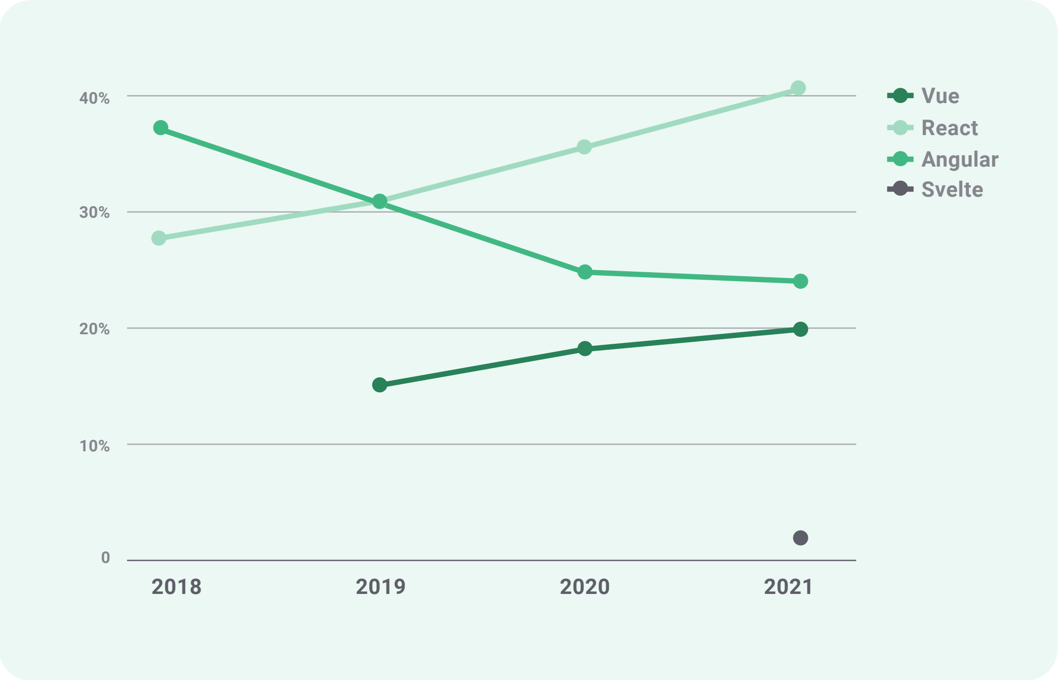
Although Vue.js is easy to learn (more than half of the SoV 2021 respondents describe it this way), there are more React developers on the market who are already experienced in this framework and would use it again.



Experience over time - Vue.js (source: www.2021.stateofjs.com)

Stack Overflow surveys

Stack Overflow surveys its users each year, asking about their most and least favorite frameworks. While the available survey questions are not consistent over the years, the results confirm the upward popularity trend for Vue and Svelte (and downward for Angular).



Stack Overflow survey: framework popularity (source: www.insights.stackoverflow.com)

2021 (+80,000 developers)

	Popularity	Loved	Dreaded	Wanted
Vue	19.0%	64.4%	35.6%	16.7%
React	40.1%	69.3%	30.7%	25.1%
AngularJS	11.5%	23.2%	76.8%	5.8%
Angular	23.0%	55.8%	44.2%	8.47%
Svelte	2.8%	71.5%	28.5%	6.6%

2020 (65,000 developers)

	Popularity	Loved	Dreaded	Wanted
Vue	17.3%	66.0%	34.0%	16.4%
React	35.9%	68.9%	31.1%	22.4%
AngularJS	16.1%	24.1%	75.9%	7.7%
Angular	25.1%	54.0%	46.0%	10.6%

2019 (+90,000 developers)

	Popularity	Loved	Dreaded	Wanted
Vue	15.2%	73.6%	26.4%	16.1%
React	31.3%	74.5%	25.5%	21.5%
AngularJS/ Angular	30.7%	57.6%	42.4%	12.2%

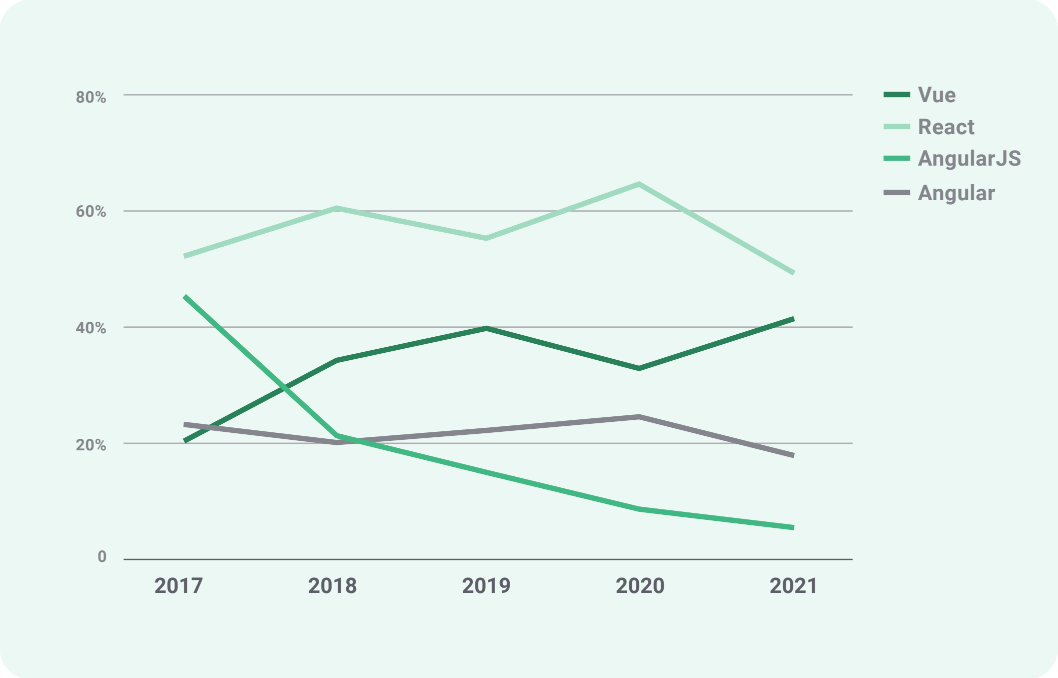
2018 (+100,000 developers)

	Popularity	Loved	Dreaded	Wanted
Vue	-	-	-	-
React	27.8%	69.4%	30.6%	21.3%
Angular	36.9%	54.6%	45.4%	14.3%

JetBrains survey

In 2021, JetBrains released a “The State of Developer Ecosystem 2021” report based on the answers of 31,743 developers from 183 countries or regions.

The share of Vue.js regular users grew from 32% in 2020 to 43% in 2021, while that of Angular users decreased from 23% in 2020 to 18% in 2021. Similarly in the case of React which is regularly used by less than 50% of developers, compared to 64% in 2020.

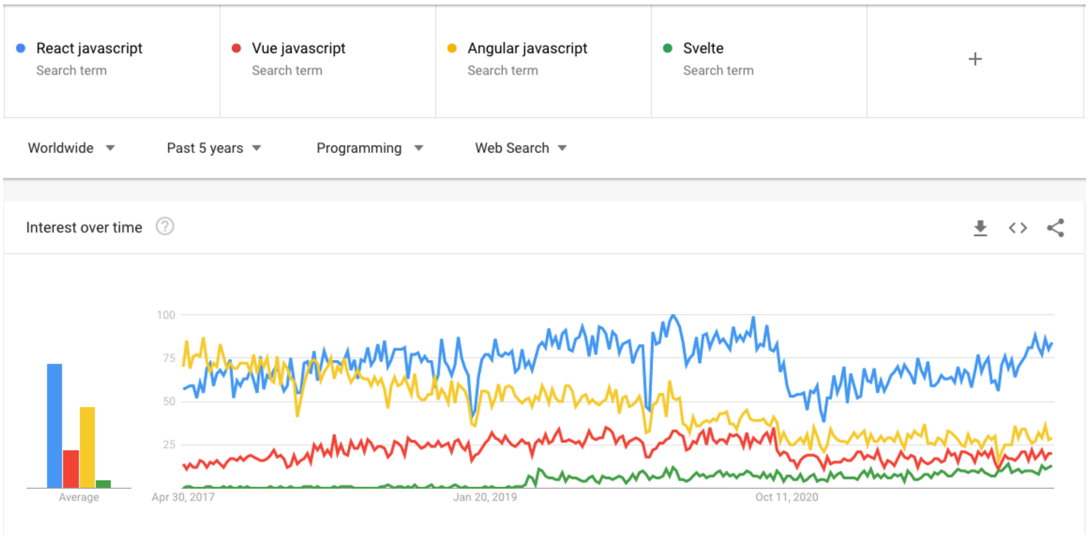


JetBrains survey: framework popularity (regular use) (source: www.jetbrains.com)

	2021	2020	2019	2018	2017
Developers	**31 743**	**19 696**	**7 000**	**6 000**	**5 000**
Vue	43%	32%	39%	33%	20%
React	49%	64%	54%	60%	49%
AngularJS	9%	11%	14%	21%	44%
Angular	18%	24%	23%	20%	22%

Google Trends

Google Trends gives us quite a broad spectrum of interests but for gauging popularity it's one of the most trusted sources out there. There was a drop of interest in all frameworks around October 2020 but since then Vue has been trending again, slowly getting back on track.



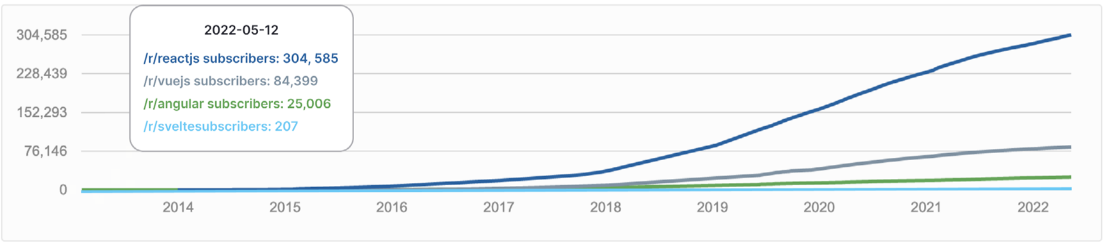
● React javascript — Search term
● Vue javascript — Search term
● Angular javascript — Search term
● Svelte — Search term

Worldwide ▾ Past 5 years ▾ Programming ▾ Web Search ▾

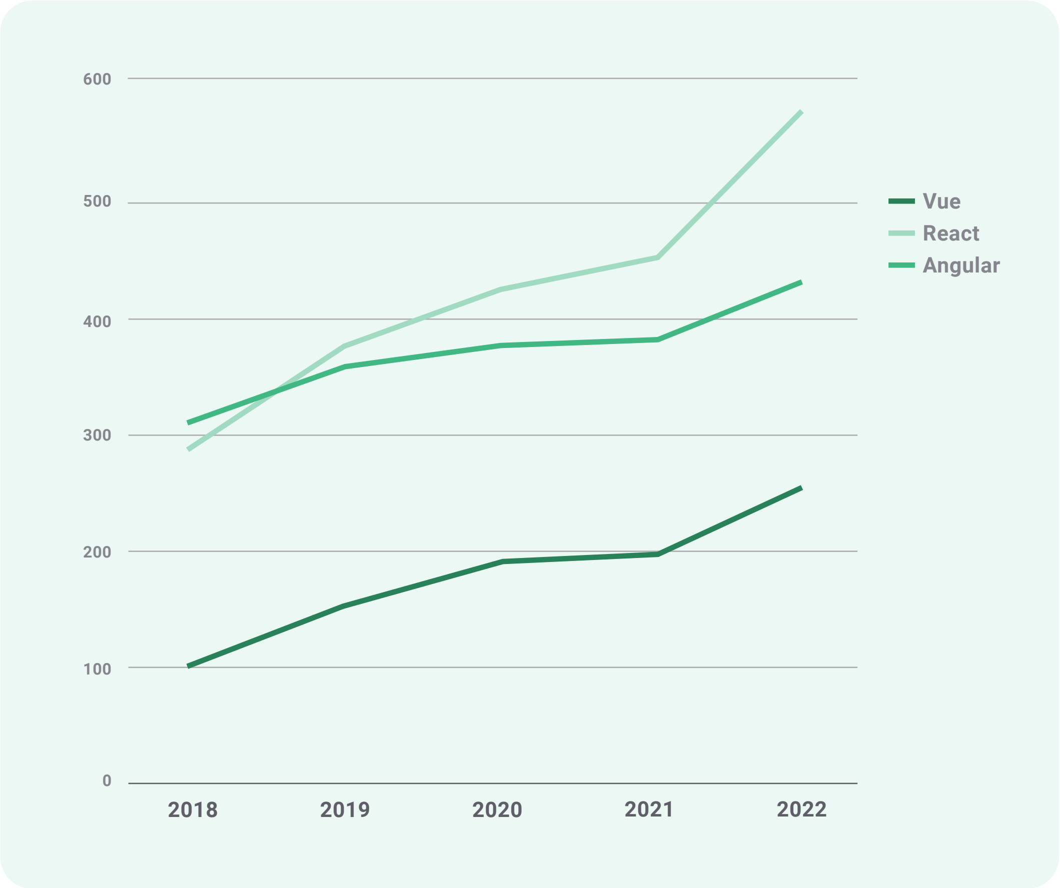
(source: www.trends.google.com)

Social media

And one last place we can look at to see how Vue is doing is social media. Namely Reddit and Twitter — virtually two of the most authoritative social sources out there. Looking at the number of followers over the years it's clear that Vue continues to get traction.



Reddit followers (source: www.frontpagemetrics.com)



Twitter followers (source: www.twitter.com)

	2022/04/06	2021/01/07	2020/09/22	2019/12/12	2018/12/16
Vue	247.8 K	196.8 K	183.8 K	149.3 K	102 K
React	572 K	451.1 K	421.8 K	355.5 K	278 K
Angular	417.7 K	376.8 K	370 K	345.9 K	304 K

05.

Review of Vue

The same five questions, yet pretty different answers. Vue experts talk about their most exciting Vue-based projects, favorite aspects of the framework, and most burning problems.

Five essential questions answered by five experts will help you gauge how Vue is used, can be used and what's in store for the future.

The questions we asked were:

1. **Why do you use Vue and what do you like most about it?**
2. **What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?**
3. **What is the most burning problem you'd see Vue tackling next?**
4. **Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind/start the process?**
5. **How do you see the future of Vue?**

Carlos Rodrigues

VueJs Core Team Member

Q

Why do you use Vue and what do you like most about it?

Carlos: I use Vue because I love how the reactivity does most of the heavy lifting and things just work. The way where the Vue template is basically html with some extra bits, is awesome because you can use the best tool for the job (using html and css), without needing to go to JSX (which is not HTML).

What I like the most is the Single File Component.

Q

What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?

Carlos: Building a medical app is probably the most exciting project because of some technical challenges but also because of the goal. Building such software will improve the life of clinicians and make a huge change to outdated infrastructure. I recommend Vue for all sorts of projects, it is quite flexible - you can build simple brochure apps with it or go for highly complex applications.

Q

What is the most burning problem you'd see Vue tackling next?

Carlos: DX, currently DX on Vue could be better. I believe Vue 3 improves quite a lot and new tools aim to provide a better DX for developers, either by TypeScript or by extensions like Volar or VueDX.

Q

Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind/start the process?

Carlos: Yes, I have adopted Vue 3 to all my current projects. Because I work mostly with complex applications, Vue 3 was a no-brainer. The Composition API and reactivity are much welcomed. Although you can still do it in Vue 2, the integration as a plugin is not as good as the native support in Vue 3.

Q

How do you see the future of Vue?

Carlos: I see Vue becoming more prominent in the Web space. I remember when I started learning Vue, there weren't many job offers but today it is a different landscape. I'm quite happy to see it and expect the efforts that the Vue Team is doing will improve the framework and bring even more people to the ecosystem.

Lucie Haberer

Developer Experience Engineer at Prismic

Q

Why do you use Vue and what do you like most about it?

Lucie: Funny enough, when I started to learn JavaScript frameworks in 2017, I started with React. A bit after, I picked up Vue on the side to be able to compare the two frameworks and come up with my own opinion on them.

As I was learning Vue, things clicked for me! As a student I struggled to understand JSX, the Vue syntax felt just more natural to me, being closer to what I already knew: HTML, CSS, and JavaScript.

The proximity of Vue with web basics is something I'm still really fond of. To me, it's one of the framework's greatest strengths, and the main reason I keep recommending Vue to teams who want to pick up a JavaScript framework for their projects.

Q

What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?

Lucie: In my opinion, Vue fits any kind of project. You need some interactivity on a single page? Drop in the runtime library. Want more? Between Vite and the incoming Nuxt 3, the choice is yours to build the website or web app of your dreams.

More recently, the new Nuxt 3 link component has been an exciting project I had the opportunity to work on and a great way to contribute to the framework.

Regarding Vue projects I worked on, I'm still really fond of the experience I had at Société Générale (French multinational bank). I managed and organized the migration of Java monoliths to Vue-based Jamstack apps there.

Q **What is the most burning problem you'd see Vue tackling next?**

Lucie: During the last year or so, the Vue Core team and the community did an outstanding job at making the Vue ecosystem ready for Vue 3. With Vue 3 now being the default, I think it's time for us to look ahead and explore what the future of web development will look like.

Q **Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind/start the process?**

Lucie: Yes, and no, so I'll answer both!

First of all, part of my job at Prismic is about developing the Vue integrations developers use to integrate with our tool. In that regard, I'm maintaining both a Vue 2 and a Vue 3 plugin. From that experience, I can tell that I enjoy the new composition API and strong TypeScript integration that come with Vue 3.

When it comes to projects, I am and would definitely build new web apps and SPAs with Vue 3. However, when it comes to static sites, Vue 3 still lacks strong support for them: Server-Side Rendering is still experimental with Vite and Nuxt 3 is not fully there yet at the time of answering.

Q **How do you see the future of Vue?**

Lucie: I'm really excited about the future of Vue. Vue 3 set strong foundations that will enable developers for quite some time to continue exploring the future of web development.

I'm also hyped about the future of the ecosystem living on top of Vue. We've seen game-changing tools being created around the framework, starting with Vite, and more recently Vitest, without mentioning Pinia, Slidev, Histoire, and more!

Filip Rakowski

CTO & Co-founder
at Vue Storefront

Q

Why do you use Vue and what do you like most about it?

Filip: I started using Vue long ago when AngularJS was still a thing. I saw an article about Vue on Medium and immediately fell in love with its syntax and simplicity. As I knew AngularJS, the reactivity system was familiar to me and the rest of the concepts I grabbed almost immediately.

At that time I was an inexperienced developer. The fact that Vue was fully Open Source without any big corporation behind it was a thing that made me excited about being part of this community which led to many amazing opportunities and friendships.

What I loved about Vue the most was its simplicity and the fact that it was enforcing some level of code readability despite being very easy to learn.

Q

What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?

Filip: I don't think anyone would imagine me writing anything else than Vue Storefront! It's an Open Source framework for building eCommerce Storefronts. Right now it requires heavy time and money investments to migrate from monolithic platforms to headless and the frontend is a significant part of this investment. Our goal is to make headless available for big and small companies all around

It has proven itself to be extremely easy to adopt which was also a major factor influencing the success of Vue Storefront.

the world through our Open Source technology. We chose Vue at that time because we knew it would be easy to learn by backend developers and juniors.

A lot of time has passed since then and Vue matured a lot. I can confidently say that Vue is perfectly suitable for any project that requires a frontend framework, including enterprise ones but it still shines the most when it's being used by inexperienced developers or ones who transition from backend technologies.

Q **What is the most burning problem you'd see Vue tackling next?**

Filip: I wouldn't call that a problem, rather a challenge. We see that the frontend frameworks world is going through a massive change. Paradigms are changing. People are starting to notice that modern frontend frameworks produce vast amounts of JavaScript that slows down the websites so obviously they are looking for ways of overcoming these issues. Svelte is reducing the runtime JS to a minimum through a compilation step, many frameworks adopt island architecture to better utilize server-side capabilities of the framework and use JavaScript only when it's absolutely necessary. I am curious how Vue will adapt to the changing requirements of the frontend market and if its concepts will remain viable. I keep my fingers crossed for it!

Q **Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind/start the process?**

Filip: I had a chance to play a bit with Vue 3 but since Vue Storefront is using Nuxt we are still waiting for Nuxt 3 to be stable to fully embrace it.

Q **How do you see the future of Vue?**

Filip: While I can't wait to see Vue exploring different concepts of reducing the JavaScript size and its execution time, the thing that I am excited about the most is Nuxt 3. I think the JavaScript frameworks themselves are getting closer to each other in terms of syntax and features but the meta frameworks on top of them like Nuxt and Next are the main differentiators in the framework ecosystem. I'd love to see Nuxt implementing new features that „automagically" do the boring stuff for us and under the hood optimize our code to the most optimal form. This is what they've been known for, and looking at the current state of Nuxt 3 it seems like it's gonna be even easier to build excellent web applications with Vue!

Maya Shavin

Senior Software Engineer at Microsoft

Q

Why do you use Vue and what do you like most about it?

Maya: Vue is very light weighted, so easy to start, and still keeps the core concept of HTML, JS, and CSS as the independent languages respectively. On one side it gives you the power of a modern frontend framework for building reactive UI, on the other, it doesn't require a sharp learning curve to start. I also like the fact that Vue focuses on the UI and the UI only. This clear goal gives a clear roadmap for Vue to improve, and I really appreciate it as a developer. Among all the features, my favorite one is how the ecosystems around Vue are built, all the tools are very connected to the core of Vue, and the strong community. This is unique.

Q

What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?

Maya: I think every project I did with Vue is exciting! The most exciting is the latest project where I start building a component library using Vue and XState as the state engine for controlling the component. Another exciting one is StorefrontUI, an e-commerce component library I was part of. It was to give the developers a library dedicated to building scalable and performant e-commerce with Vue under the hood.

I would recommend trying out Vue for small and medium-size projects first, such as e-commerce projects, or any project you want to start up quickly and scalable.

Q **What is the most burning problem you'd see Vue tackling next?**

Maya: Scalability for large projects, definitely.

Q **Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind?**

Maya: I started adopting Vue 3 in some new projects, composition API is great, but OptionAPI is still more organized in a way. Vue 3 is still new and contains a lot of important breaking changes that prevent us as library authors from migrating our projects to support Vue 3. And because of the lack of support for Vue libraries, it's hard to start using Vue 3 completely.

Q **How do you see the future of Vue?**

Maya: Vue continues to be a lightweight and innovative frontend framework, especially with all the performant tools such as Pinia, Vite, Vitest around it. However, the problem of Vue 3 support can slow down Vue adoption for new developers, and prevent companies from choosing Vue 3 to work with. The future is interesting to see.

Ramona Schwering

Software Developer
at Shopware

Q

Why do you use Vue and what do you like most about it?

The first point is my love for open source. Vue being driven by an open-source community means a lot to me, not only because I'm working on open source projects most of the time. I started to learn Vue as I began to work with Shopware 6 and have continued to do so ever since. I quickly grew to love Vue's simplicity: Building and structuring components in the way Vue provides feels clean and easy. On top of that, I was thrilled to see it's easy to learn, and the learning curve was shallow. The way a framework is onboarding new developers is a high priority, so Vue won me over quite fast.

Q

What's the most exciting project you've done with Vue? For what type of projects would you recommend trying out Vue?

The most exciting project with Vue is Shopware 6. It's an open-source ecommerce platform using Vue in its Administration part. Even though it's still on Vue 2, it's exciting to see how we can build a large-scale application with Vue.

> **If I think about which projects to try out Vue for, it's a beautiful idea to use it on any single-page application.**

Q

What is the most burning problem you'd see Vue tackling next?

It is more general, but as projects grow more complex, I'm always keen to see how Vue can become even easier to use (as it already is) and how the developer experience can become even better. That way, I'm excited to see any simplification of usage and new features for the Vue DevTools.

Q **Have you adopted Vue 3 yet and how did it look? What was the reason? OR What keeps you from implementing Vue 3? What would make you change your mind?**

Not yet, as Shopware is an open-source project, it would cause breaks. This way, I'm still dealing with Vue 2 most of the time. However, I hope we'll be able to change to Vue 3 in one of the upcoming major versions, so I don't need to be convinced anymore. I played around with Vue 3 in private side projects and for test automation and got an excellent first impression.

Q **How do you see the future of Vue?**

One word: bright. Especially with this beautiful community and their diverse perspective and views. I'm excited to learn to get better in Vue and build even more clean, performant and fantastic applications to help other people and myself as soon as possible.

I'm confident that Vue will stay so well-to-use and focused on the needs of us, developers.

06.

Experts Corner

A deeper dive into the Vue Amsterdam presentation topics and a guest interview with the Vue Core Team member - Anthony Fu.

Everything you need to know about Web Performance

Filip Rakowski
CTO & Co-founder
at Vue Storefront

Building efficient web applications doesn't have to be difficult. Knowing what affects it and following a few good practices, we can make each application work quickly. In this article, you will learn how to measure performance, what affects it, and what are the most effective ways to optimize the performance of web applications.

I hear a lot of people saying that web performance is hard. Honestly, I don't think that's true. It could feel complex and intimidating at first glance because there is a lot of domain-specific naming, metrics, etc. but to build a fast website you don't need to know them. You need only need a basic understanding of what influences the speed of your website the most and make sure you have it under control. That's enough!

What influences your app performance?

Let's start with identifying all the aspects that influence it. I find this mental model most useful when thinking about web performance.

There are essentially three "steps" that sum up the overall loading performance of your app:

- **Server-side execution** - first the HTML document has to be generated on the server. In some cases, this step is costing us nothing because it's already generated (static sites).

- **Network** - the generated HTML document has to travel through wires and routers to arrive in the user's browser.
- **Client-side execution** - the document needs to be parsed, and dependencies (CSS, JavaScript) have to be downloaded and executed. Once it's all done our page is fully loaded.

Optimizing server-side execution

If you're building SPA there is a high chance you're also adopting SSR. In that case, the same code will run both on the server and the client. The best code is the one that never has to run so you should first consider SSG. If it's not an option and you're sticking to SSR make heavy use of full-page caching and distribute cached content through CDN.

Some pages will have to be generated on the server during runtime and just cannot be cached. In those, make sure to fetch only fast, essential data on the server and make less important, and slower API calls on the client-side. This way you will significantly improve your Time to First Byte.

Optimizing Network

Optimizing the networking part boils down to four main rules:

- Ship the smallest possible assets. The bigger they are, the longer it will take to download them.
- Avoid chaining network requests (making one request depending on another) and try to download them in parallel.
- Avoid using multiple external domains in the critical path. Establishing a connection with all of them will take more time than downloading everything from one source.
- Cache static assets (HTML, CSS JS) through a Service Worker.

If you take care of that there is a much smaller chance you will run into performance bottlenecks on the network part.

Optimizing client execution

This is where we, frontend developers, have the biggest power and where we also make a lot of mistakes. From my experience, 90% of frontend performance bottlenecks are caused by two factors:

- **Unoptimized images**

To make sure images aren't the bottleneck simply adjust their size to the screen and use next-gen formats like webp. You can automatically resize and optimize your images using `<nuxt-img>` and/or Cloudinary. Also, load your below-the-fold images lazily. You can use native `<img loading="lazy" />` for that.

- **Unoptimized JavaScript**

The thing that is usually leading to the biggest number of performance bottlenecks is JavaScript. In SPAs it's very easy to lose control over your JS bundle size. Here's what you can do to prevent it from growing into a Brontosaurus:

- If you're using SSR/SSG it means that many of your components are already rendered on the server and they don't need interactivity on the frontend. You can drastically increase the speed of your hydration by hydrating only the components that need to be interactive and only when they need to become ones. You can use Astro.build or vue-lazy-hdyration plugin if you're using Nuxt to control the hydration process and exclude the components that don't need it.
- Split your app into multiple lazy-loaded chunks (start with routes!). Every sidebar, modal or expensive widget can be loaded lazily on interaction.
- Your website could seem fast when you're building it but once the marketing team will put all the analytics there I guarantee it will slow down. You can use web workers to run the non-critical code asynchro-

nously. I strongly recommend Partytown - it's integrated with all major frameworks from the Vue ecosystem.

Measuring performance

If you can't measure - you can't say if there was any improvement. Measuring your performance constantly is as important as optimizing it regularly.

If you want to quickly check how your website is performing try Page Speed Insights. It will run a Lighthouse audit on your website using the closest Google Data Center.

You should also incorporate performance checks into your CI/CD pipeline. Use Lighthouse CI to run a synthetic Lighthouse test on each PR and bundlesize package to raise alerts if your bundle size exceeds a certain threshold.

Believe it or not but that's all you need to know to have your performance under control!

Developer Experience with Nuxt 3

Daniel Roe

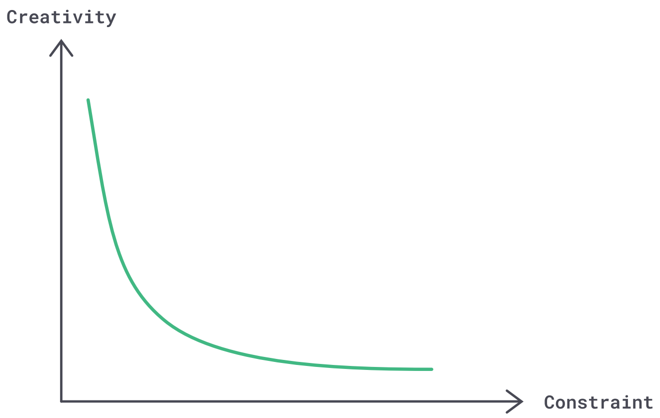
Framework Architect
at Nuxt Labs

Optimising developer experience with Nuxt 3 - a tour through the ways the new version of Nuxt will save time and make your life easier. Here I'll explore zero-config options, dive into Nuxt 3's improvements around data fetching and along the way tour a host of other features that will give you superpowers.

Great developer experience (DX) matters. It matters for our own feeling of well-being, competence, and productivity. And it even matters to the businesses who employ developer teams - by improving efficiency, increasing agility and retaining talented team members.

There are many factors that go into making an optimal environment for creators. Of course, many of these factors are independent of the tools we use - such as a safe environment, the trust of those we work with, and the opportunity to make a real difference in the world.

But some of what enables us to flourish comes from the tools we use. I think we feel most productive when our tools *optimize* for creativity and *decrease* constraint.



Here are three ways that we're trying to make Nuxt 3 an amazing experience for developers.

Zero config

Nuxt is a framework for building web applications that doesn't require any configuration at all. Create a page in the `pages/` directory and we automatically enable vue-router integration, including automatic bundle splitting.

That may be familiar to users coming from Nuxt 2. But we've also added a host of new zero-config options. For example:

- Files in `plugins/` get automatically scanned and run when your app is initialized on server/client - if needed, you can specify which with a .client or .server suffix.
- Any files in `components/` become Vue components that can be used anywhere in your app - but don't worry, by default they are still only imported into the pages that use them, meaning your build will still be optimized for production.
- Any files in `composables/` are scanned for exports, and these are then auto-imported wherever you use them in your app.
- Any files in `middleware/` are automatically registered, and anything with a `.global` suffix runs automatically on every route change.

- There's a new syntax for dynamic routes in `pages/` enabling more complex patterns for part-dynamic routes, such as `pages/@[username].vue`.
- Any files in `server/api/` become Nitro server routes. You can specify the HTTP method they accept by adding `.get`, `.patch`, etc. before the file extension.

… and any files in `server/middleware/` are automatically registered to run on every server request to your built app.

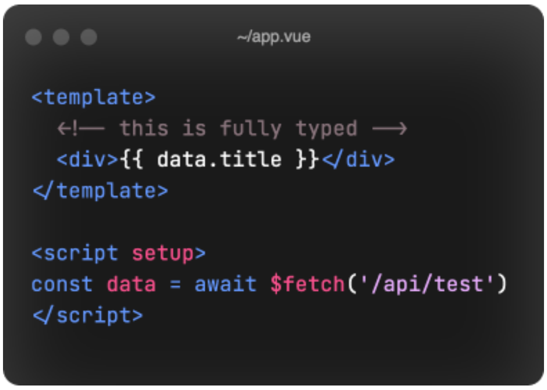
No boilerplate

As much as possible, we've tried to get rid of the code that doesn't matter.

Instead of needing to import utilities like ref, defineComponent, defineNuxtPlugin, and so on, Nuxt automatically imports them when and where you use them.

We've also been able to make some significant DX improvements around data fetching. For example, in your server routes, you can directly return data rather than stringifying it and setting the content-type headers manually. And using the new `$fetch()` helper powered by ohmyfetch, you can directly access content returned from your fetch calls without needing to parse the response yourself.

The new Nuxt CLI also makes it easy to add new API routes, plugins, components, composables, middleware, layouts, and pages - meaning that the out-of-the-box Nuxt starter is just `app.vue`, an empty `nuxt.config.ts`, a stub `tsconfig.json` that extends the one Nuxt will generate, a `package.json` and a `README.md`.



```
<template>
  <!-- this is fully typed -->
  <div>{{ data.title }}</div>
</template>

<script setup>
const data = await $fetch('/api/test')
</script>
```

Fully typed

The types you get in your project are generated automatically at runtime based on the specific settings of your project, and exposed to your editor so ‘what you see is what you get.’

For example, auto-importable components used in your app are made available to your IDE. If you’re using Volar, you’ll even get type support for the props of these components.

… and any auto-importable composables you’re using throughout your app will also be declared globally so you get full type support without needing to import them individually.

The types for any global injections you make in your plugins are also inferred and injected globally into the `useNuxtApp` helper function.

If you have a `pages/` directory, then you’ll even get as-you-type help when choosing layout or middleware for your page using the new `definePageMeta` compiler macro.

Plus, your runtime configuration is typed, meaning you can see exactly what keys are available when you call `useRuntimeConfig()` to access dynamic data that can be set after your app is built.

You’ll even get type-hinting for the return type of any calls you make to the Nitro endpoints (in `server/api/`) you create within your app. That means you get full type checking support for something like this:

And when using a Nuxt module, you no longer need to add the types manually to your tsconfig - it’s all done automatically for you.

We’ve put a lot of work into taking Nuxt’s developer experience to the next level, leveraging tools like TypeScript to make the experience effortless.

If you'd like to try it out, head to https://v3.nuxtjs.org. And why not join the Nuxt community on https://discord.nuxtjs.org and on https://github.com/nuxt/framework? I'd love to hear how you get on. Plus, Nuxt is a community project, so if you see any areas where we can improve, I'd love to hear from you!

The story of improving developers' experience with Vue

Anthony Fu
Core team member
of Vue, Nuxt and Vite

INTERVIEW

After the Vue 3 release, we have seen significant changes in the whole Vue ecosystem. A new generation of tooling brings improvements in all development aspects - developer experience has significantly improved thanks to Vite, Vitest and Volar. What's the story behind these frameworks and tools? And how to make the transition to the latest version of Vue easier?

Q

After the Vue 3 release, we have seen a lot of development in the whole Vue ecosystem - what has changed and how does it influence developers' experience?

Anthony: Vue's major change comes down totransitioning from Options API to Composition API. It's about the capability of Composition API to support TypeScript better and make the app logic possible to easily decouple from components. This way you have a better organization of your code which improves the developer's experience and project management - it makes code healthier.

What's more, perforance in Vue 3 improved by around 30% compared to Vue 2. This is the baseline that every app could get. Together with the introduction of Composition API, we introduced the new Script Setup RFC. It enables you to have a cleaner look of your Single File Components so you can focus on the logic instead of writing the scaffolding code in every single component that you have. It reduces time and effort which was used to write the parts you basically copied and pasted like defineComponent. The script setup syntax reduces that a lot.

Also, there are minor features like suspense and async component support. All of those features are something that makes people migrating to Vue 3 feel the real upgrade. These are the things mostly for the Vue itself but it's not all - there's also Vite.

Q **Speaking of Vite, what kind of improvements does it bring over existing alternatives?**

Anthony: Vite is framework-agnostic and was originally created for Vue but currently serves more frameworks like React for example. Vite leverages the power of ESM inside the browser. You don't need to bundle your apps but can ship everything - modules inside your source code directly, with some transpilation to remove the TypeScript notations so that the browser understands the code.

Together with HMR (hot module replacement), it enables a really fast development. I mean really fast - start-up times can decrease tenfold in comparison to other tools, such as webpack.

Let me give you an example, in Vue 2 and Vue CLI you needed to wait one or two seconds for your changes to appear in the app. It doesn't seem like a lot but in practice, your changes might break the app and you would see it only after some time. Sometimes it can take a lot of time to trace back changes and fix errors. In Vue 3, you can try different combinations during coding and see the effect immediately so you can quickly figure out what's the best solution. It reduces the feedback loop of your changes and it's something that makes a huge difference in developers' experience.

Also, having an ecosystem on top of Vite is a huge advantage. In Nuxt 3 we support both Vite and webpack as bundlers and you can switch between them but we made Vite the default bundler so you can get the benefits of the powerfulness that Vite provides. These are fundamental changes to the Vue ecosystem.

Q **How about Vitest? What's the difference compared to other testing frameworks?**

Anthony: Vitest is a testing framework and its API is pretty similar to Jest which is the most popular testing framework. It was created to solve the issues we experienced in Vite.

Some of Vite users wanted to have a better way of testing their app and had trouble configuring Jest with Vite. It became a barrier for people to migrate to Vite.

> **We tried to find some tools to recommend but we didn't really find any good options so we decided to utilize Vite itself.**

The most important feature of Vitest to me is that it offers everything on demand, unlike previous bundlers like a webpack. If you visit a page, it will only request the modules you use inside of this page and when you request a page, Vite starts to transpile. It only does the work that's necessary while in webpack, you need to bundle your whole app and wait for your app to start.

By using Vite and Vitest you can use the same configuration and the same plugins. Thanks to that you can have a more consistent environment between the tests and your Vite app.

Q **With the speed of Vite and this live feedback, do you think we might see even more cool integration into things like our IDEs?**

Anthony: We already have VS code extensions and also we have Vitest UI. If you enable flag, it will open up an app server which is also powered by Vite. You can view your test files visually, you can click and see what's going on instead of a huge console log.

Currently I'm experimenting with ideas like running your tests inside your source file, similarly to Rust. Rust has this feature that you can declare a testing block inside your source file. The test file will be later moved away when you compile it into production view. But when you're on the task, it can share the context of your source files. You can have an internal state that you can test without the need of exporting it outside of your module.

These are the options we want to offer and the approach we support so I would expect more future integrations.

Q **How do you feel about the adoption rate of Vite and Vitest in the Vue community and beyond?**

Anthony: I'm not familiar with the numbers but the adoption is still relatively small in comparison to previous bundlers like webpack since the migration takes time. However, the satisfaction of the community using tools and leaving positive feedback makes me really happy. People are willing to try it and once they do it, they stick to it. The Vite team feels really proud of these tools.

Q **What do you find the most important aspect of Vue 3 for businesses and when would you recommend transition?**

Anthony: Whether to make a change or not, you need to weigh your pros and cons. Vue 3 is written from scratch, the codebase is completely different and we made some breaking changes. Nuxt 3 also has a brand new codebase. At the time Vue 3 was released, most of the existing UI frameworks are not really compatible between Vue 2 and 3. If you want to make them compatible with Vue 3, there will be quite some work to do.

First, the ecosystem needs time to migrate to Vue 3. If you have a large app, it might not be an easy task and it takes time and carries some risk of breaking your app during migration. Some dependencies might not work and you may need to look for alternatives.

The biggest change between Vue 2 and Vue 3 is IE11. We used a new reactivity system based on Proxy features inside the Javascript which weren't previously available in IE. That's the tradeoff we made here in Vue 3. If your app really targets users still using older IE, you might not want to migrate to Vue3 at this moment. On the other hand, Vue 2 is still being supported for 1-2 years down the line.

If you don't want to upgrade now, it's totally fine. However, for the new app, I would totally recommend going with Vue 3 from the beginning.

Q **What is the most burning problem you see Vue Core Team tackling next? What are your plans for the future?**

Anthony: I think we've made significant changes during the last one and half years and solved lots of burning problems. We're already made the transition from Vue 2 to 3 a lot easier. Thanks to introducing compatible mode you can use Vue 3 as a core with some Vue2-compatible APIs.

This compatible mode has a lot of flags that you can toggle and for each breaking change, you can migrate your codebase step by step.

We have a Vue 2 plugin called @vue/composition-api which I maintain. So you don't need to migrate to Vue3, you can still use the Composition API and get the benefits from it which I think is awesome. What's more, I made a plugin for Vite and webpack for the new Script setup syntax inside a Vue 2 app. For Nuxt we implemented Nuxt Bridge which brings

Nuxt 3 features to Nuxt 2.

We also had quite a few libraries from the community, for example, VueUse, that can run in both Vue 2 and 3 seamlessly. Our roadmap for Vue core is almost done. While we also have some interesting experiments in progress like reactivity transform, which you can find in our RFC repo.

07.

Technical Case Studies

How can Vue solve technical challenges? Five real-life examples with lessons learned and solutions based on Vue projects delivered at Monterail.

1 3d and 2d object management and data synchronization

Challenge:

Building a web engine that will be able to manage 3D and 2D scene full of objects in real-time and synchronize the data.

Solution:

Vue and its reactivity engine are a perfect fit here.

We've used the framework's declarative approach to build an interior designing scene inside the application. Additionally, we've leveraged Vue's reactivity and virtual DOM to manage the DOM elements.

The 3D engine had its own internal state and our goal was to connect it with the central application state. Since a constant full-state synchronization would be costly, we can trick Vue.js to optimize it for us. It internally uses a mechanism called Virtual DOM to keep the DOM up to date with the component state by minimizing the number of DOM changes. We used it to generate synchronization calls to the 3D engine, according to a relevant part of the state, by building DOM representations of 3D entities and optimizing its changes with the Virtual DOM.

Scan code and read more about building 3D app with Vue

2 Creating a page builder

Challenge:

Creating a multi-tenant tool for building web pages visually.

Solution:

Vue, together with core libraries such as vue-router or Vuex, is a very flexible framework. It allows you to completely change the website structure, routing, and dynamically add and remove store modules at the runtime. We took advantage of these opportunities when creating a visual website builder.

In the application, the users had the option to define the routing of the new website. For each page of a new website, they could add elements from a predefined list, remove them or change the position of individual parts of the page using drag and drop. Each component was also editable — users could change its colors and content. The website created with this tool was dynamically created in the runtime, in the user's browser, based on the document describing the website structure which was loaded asynchronously.

In order to improve the SEO and UX of the output website, the whole thing was also rendered on the server-side (SSR). We used the "Dynamic Components" and "Async Components" features to render components on individual subpages of the application. Those mechanisms built into Vue, while very powerful, are very easy to use and do not add the extra complexity of the application. Each component was loaded only when it was needed (lazy loading), which was necessary due to the large library of components that could be used.

3 Showing huge amounts of data

Challenge:

Showing huge amounts of data in the form of a tree or 2D grid.

Solution:

The application is directed at system engineers, so it needs to show huge amounts of data in a user-friendly way.

We've leveraged DOM virtualization to render only the data that is currently visible on the screen. When the user scrolls the page, a special component responsible for the list virtualization is listening for the `scroll` event. It is checking which elements should be visible on the screen and displays only these items. Thanks to the use of scoped slots, this behavior is invisible not only to the user, but also to programmers – the component cleverly hides this implementation.

For lists (that displayed 10-20 items on the screen), this solution worked perfectly but turned out to be too slow for the 2D grid view. Such a grid in our application could have a size larger than 10k x 10k and the screen often fit more than 30x30. It meant that we needed to render more than 900 elements on the screen. This is not a big problem for Vue, which is very performant, to render this number of elements once.

However, we found out that it is not optimal to use Vue to frequently perform swaps for hundreds of items while scrolling.

Vue compares the keys of individual elements with each other, which already takes a long time with such a number of elements, but the rendering itself is also slowed down due to the overhead of reactivity.

We programmed the same process as for the list virtualization component: listening for the `scroll` event and checking which items should be visible on the screen. However, we decided to render elements imperatively. In the end, we've set up Vue to render only an empty UI element of an appropriate size. Thanks to the fact that we know the order in which elements will change while scrolling the page, we were able to optimize the process of comparing keys.

We wanted to keep using the transparent scoped slots API, so we used it but rendered it manually in the custom render function triggered by the `scroll` event. The rendered grid cells were placed and removed in the element rendered by Vue with native JavaScript methods.

4 Creating a SEO-friendly hybrid application

Challenge:

Building an application with great SEO that's also available as a mobile application with minimal code duplication.

Solution:

We have created two separate build bundles - one for our web application with SSR, the other for a hybrid mobile application utilizing Ionic and Capacitor, all bundled using Vite from a single codebase.

We introduced SSR relatively easily following Vite's SSR guide to optimize for SEO on all public pages of the app in our web build. Ionic was introduced to provide the best native feeling we can get for the mobile applications out of the box and also add some native-like UI components for the mobile-only part of our mobile interfaces.

Our goal was to re-use as much code as we can and as such Vue's composition API was a great help here. We leveraged composition API in order to reuse code in terms of platform-specific component variants and used Vue's async components as a way to provide those for a given platform if necessary.

All of our platform-agnostic components were implemented in a responsive fashion and doing so allowed us to share the majority of the code between the two with minimal exceptions.

As a result, we created a web application with SSR along with hybrid iOS and Android mobile applications built from one codebase.

5 Rapid UI redesign

Challenge:

When a project was in its final stages, we had to perform a rapid and comprehensive redesign of the entire application interface. Styles were global and not scoped at that time.

Solution:

We pivoted the fronted design strategy and started using Vue with small, reusable components, that allowed for style encapsulation.

Using Vue's slots and scoped slots, we quickly created a collection of simple reusable components from the new design system. We gradually replaced almost all usages of global classes and styles with our styled components. Thanks to the use of components and scoped styles, the unchanged pages still looked as before. We were able to focus on each component separately and we weren't worried about unintended changes and naming conflicts related to global style.

Monterail is a full-service software development company from Poland, the official Vue.js partner and author of four State of Vue.js reports. Our experts, using battle-tested technologies and frameworks, have delivered more than 30 Vue-based projects. At Monterail, our love for Vue is more than just utilitarian - it goes beyond Vue services and projects.

Scan the QR code

and find out more

www.monterail.com

monterail

AMSTERDAM, JUNE 2022