

EVERYTHING
YOU NEED
TO KNOW TO MAKE
AN EDUCATED
DECISION ABOUT
YOUR TECHNOLOGY
STACK

STATE OF VUE.JS

2017

BROUGHT TO YOU BY MONTERAIL
IN COLLABORATION WITH EVAN YOU



What's Inside?

1	Preface	3
2	The Evolution of Vue.js Damian Dulisz	7
3	How Developers Use Vue.js	15
4	Case Studies	29
a	Behance & Adobe Portfolio Erin Depew, Matt O'Connell, and Yuriy Nemtsov	32
b	Chess.com Scott O'Brien	40
c	Clemenger BBDO Sylvain Simao	46
d	Codeship Roman Kuba	52
e	GitLab Jacob Schatz	60
f	Livestorm Gilles Bertaux	67
5	The Future of Vue.js Evan You	74

Preface

A few years back, Montrail was a well-established development agency, renowned for its Ruby and Rails expertise. Now, from today's perspective, such a label feels a bit odd. We started with traditional Ruby multi-page app development, but pretty soon it became obvious to us that many of the practices and patterns changed as technologies evolved. Backbone.js was our first foray into JS frameworks in 2011, undertaken since we couldn't just stay inert if we wanted to meet the market needs. We have been actively monitoring this dynamically changing world, and along the way we early-adopted Angular JS and became experts on it. With the advent of the latest generation of component-based frameworks, our team has embraced all the major players in the field, including React (along with React Native), Angular (2 and up), and—most extensively—Vue.js.



Familiar Does Not Mean Better

Before we started working with clients who requested an app written with Vue directly, we were talking to businesses who have never heard of it. Once they did, they were impressed with its openness and power, and wanted to include Vue in their technology stack.

We feel that most companies which choose more well-known frameworks make their decisions without having all the relevant informa-

tion and simply adopt something that sounds familiar. They're usually unaware that somewhere out there, there's a technology which combines the advantages of Angular and React and adds an additional layer of niceness on top of it.

▶ **The Rationale Behind the *State of Vue.js* Report**

With Vue in our tech stack we can efficiently deliver better products—it helps us drive our business and make clients happy, and so we believe it deserves all attention and love. With that in mind, we embarked on the journey to evangelize to developers and businesses and spread the word about Vue. That's how we ended up curating the weekly Vue-newsletter, organizing VueConf, the first international Vue.js conference in the world, and creating libraries like Vuelidate and Vue-multiselect.

The report you're reading is yet another milestone in that mission. It was created for three primary reasons. One, to provide a reliable source of Vue.js business use cases so anyone can get a sneak peek at how other companies use Vue.js. Two, to reach more individuals who have never heard of Vue and provide them with good reasons to give the framework a closer look. Three, to never, ever again have to convince our clients that Vue.js is a ready-to-use solution and has everything we need to build all kinds of applications.

▶ **Contents of the Report**

The *State of Vue.js* report offers a business owners' and developers' perspective on Vue. We surveyed over than 1,100 specialists from 88 countries to find out their experiences with Vue, and what they

like and dislike about it most. We dug even deeper, and interviewed six companies on what problems they wanted to solve with Vue.js. To give you an overview of its growth over the years, we described the story of Vue.js, also including a sneak peek of what's coming next from Evan You, the creator of the framework himself.

Enjoy the read,



Damian Dulisz
Frontend Developer
at Monterail



Karolina Gawron
Content Marketing
Manager
at Monterail



Marta Klimowicz
Head of Marketing
at Monterail

We would not be able to pull this report off if it weren't for many amazing people who supported us along the way. They all were a tremendous help, sharing their knowledge and experiences, just because they wanted to give back to the community they're part of.

Big thank you to Evan You, who was excited about the report from the very beginning and had our backs during the creation of this very piece of content. He also agreed to share invaluable insights about the future of Vue.js and supported our writing efforts.

Evan, as well as Chris Fritz, Vue.js core member, were insanely helpful with analyzing the *State of Vue.js* survey results. Kudos for that. Because of our collaboration, we felt comfortable about the quality of the final product.

The case study part of the report would never come into existence if it weren't for all those who agreed to spend their time sharing their stories. The warmest thank yous to Jacob Schatz, Sylvain Simao, Roman Kuba, Gilles Bertaux, Scott O'Brien, Erin Depew, Matt O'Connell, and Yuriy Nemtsov.



Contributors



Evan You
Creator of Vue.js



Chris Fritz
Vue.js team
core member



Jacob Schatz
Frontend Lead
at GitLab



Sylvain Simao
Technical Lead at
Clemenger BBDO
Melbourne



Roman Kuba
Lead Frontend
Developer
at Codeship



Gilles Bertaux
Co-founder & CEO
at Livestorm



Scott O'Brien
Lead UX Engineer
at Chess.com



Erin Depew
Software Engineer
at Behance



Yuriy Nemtsov
Software Engineer &
Manager at Behance



Matt O'Connell
Software Engineer
at Adobe Portfolio

The Evolution of Vue.js



Damian Dulisz
Frontend Developer
at Monterail

Did you know that when Vue was first released, it wasn't even called "Vue"? The first commit, is dated back to June 27, 2013 and Vue was still called "Seed" at the time. That's over 4 years ago today. The first name stuck for nearly six months before it was eventually changed to Vue in early December of 2013. The first public release (0.8.0), however, was unveiled only in February of 2014. At the time, Vue was a library that only focused on the View layer of the MVC architectural paradigm.

There were several important aspects of Vue that made it "click" with developers. The template syntax followed a style similar to AngularJS and a component-based architecture introduced by React, thus creating a smooth bridge between the two mindsets. I like to think about Vue as the lovely child that got the best parts of its parents, AngularJS and React, with a constant emphasis on developer experience and approachability.

The JavaScript community became increasingly interested in Vue, but it was a year later, when the Laravel community (gathered around this popular PHP framework) first discovered it that Vue really took off. A couple of months later, the long-awaited 1.0 version was finally released. This was a groundbreaking step for the library.

In the meantime, the community saw the release of `vue-router` (August 18th, 2015), `vuex` (November 28th, 2015) and `vue-cli` (December 27th, 2015). These libraries marked the transformation of Vue from a View-layer library into what we today call the Progressive Framework.

Last year, we've seen the release of the much-anticipated version 2.0—a complete rewrite of the framework which introduced several new concepts like the Virtual DOM and Server-Side Rendering capabilities. However, the API remained virtually unchanged, so the migration was smooth. The official `vue-migration-helper` tool helped the process, too.



The Community

Fast forward one year and the still-thriving community has made Vue.js one of the top 3 most popular JavaScript frontend frameworks to date. And it doesn't look like it's going to stop there.

People fell in love with Vue. But rather than trust in our emotional assessment, take a look at the numbers—Vue was the most starred project on GitHub in 2016. Talk about developer enthusiasm!

The community interest is incredibly strong—when we launched the [Vue Newsletter](#), hundreds of people subscribed in a matter of minutes. The never-ending stream of email notifications made us feel like Instagram stars. The first issue of the newsletter went to 759 subscribers. Sixty-three weekly issues later, our audience has grown to nearly 6,000 subscribers. Each new issue is harder to prepare because of how much new Vue-related content surfaces basically every week now. High-quality tutorials, insightful articles, and all the libraries I could ever think of now

tend to show up daily. It's insane! And that's not all—the Vue community is bolstered with a [thriving forum](#) and a [Discord channel](#), with thousands of developers active on both of these outlets each day.

Additionally, we can see that a growing number of companies across the globe are increasingly betting on Vue, following the rapidly growing interest of their developers. Just look at all the job offers published at [vuejobs.com](#).



The Ecosystem

I think it's worth mentioning that apart from the community projects, the Vue Core Team also maintains several official libraries, such as `vue-router`, `vuex` (state management), `vue-rx`, and `vuex-observable` (for RxJS), as well as tools like `vue-cli`, `vue-server-renderer`, `vue-loader`, `vetur`, and `vue-migration-helper`. Why is this important, you ask? Because it allows you to progressively opt-in to use other core libraries that transform Vue into a full-fledged framework like Angular or Ember, with the guarantee that it will work seamlessly. However, you can always switch parts of it for other, unofficial solutions, if that is what your project requires. Another good thing about the official supporting libraries is that they always represent the highest quality and offer long-term support and compatibility with Vue itself.

As one would expect, a massive and highly involved community such as Vue's comes with a significant number of community projects. And not just small, focused libraries, we're talking large-scale projects here. For example, [Nuxt.js](#) is a highly-opinionated framework built on top of Vue that combines several smaller tools as well as patterns that make it incredibly easy to develop applications with SSR support.

There's the [Quasar Framework](#), which helps with the development of hybrid mobile and desktop applications. There are also several very popular UI frameworks like [Element-UI](#) and [Vuetify](#) that will give you dozens of unified UI components to bootstrap your application. Vue also gets more and more support from mobile development frameworks like [OnsenUI](#) by Monaca and [NativeScript](#).

From my perspective as a Web applications developer, I can assure you—it already has everything your app will probably need. Each week I see more and more libraries being published, to the point that it's impossible to keep track of all of them. Many of those libraries can be found here: [\[A list of awesome things related to Vue.js\]](#). Additionally, the core team manages a list of recommended libraries at [curated.vuejs.org](#) for popular tasks like validations, i18n, AJAX requests, to help with the fatigue of picking the right tool.



The Backing

Many people point out that Vue is not backed by any large company in the same way Angular or React are, and they act like it's a bad thing. I dare to disagree. Vue truly embodies the spirit of open source, like jQuery, Babel or webpack do, and those are the most reliable tools in the JavaScript world. There is a significant advantage in that. The project doesn't have to follow the needs of a particular company and can instead focus on what the community needs.

And Vue does deliver what the community needs the most. When speaking about the support for code splitting, Sean Larkin, a core team member of webpack, described Vue as:

The first framework to craft a Developer Experience with webpack in mind.

But the focus on developer experience goes far beyond webpack and is present in every aspect of the library, starting from its ease of use, through smooth integration, and up to top-notch documentation and overall scalability.

Obviously, Vue.js—again, like almost every other open-source library—began as a one-man project. Since then, it grew to the point where it has a fully-staffed core team that takes care of different aspects of the library and its ecosystem.

The funding? Over the last two years, individuals and companies from all around the world decided to support Evan You (the creator) and the Core Team with a stable monthly income of more than \$10,000 through successful campaigns on both Patreon and Open Collective. This allowed Evan to shift to working on Vue full time.

The sponsors include multiple companies and hundreds of other private backers. The list of current sponsors can be found [here](#).



The Growth

Here are some numbers that might give you a feeling of how fast the Vue ecosystem is growing.

Let's take GitHub stars for example. Although they might not be the perfect metric to demonstrate the popularity of a project just yet, they surely

show the excitement around it. And the excitement was high enough to make Vue the [Most Starred Project on GitHub in 2016](#). Not just within the JavaScript or frontend categories. It was the most starred project that year, period, and right now it's the second most-starred frontend framework, sitting slightly behind React, and the sixth most-starred GitHub project overall. It already surpassed jQuery and Angular.

Surveys like the [State of JavaScript 2016](#) indicate that Vue has one of the highest satisfaction ratings and 89% of developers who've used Vue before would pick it again.

Naturally, there are a number of other metrics to check out, such as monthly npm downloads (~800k) or weekly dev tools users (~270k). The npm numbers might seem small compared to React, but it's worth mentioning that Vue downloads increased more than five times over the last twelve months. Looking at the momentum Vue currently has, I believe it's safe to assume the numbers will grow at an even greater pace in the coming years.

A big part of this growth comes from the fact that more and more companies are picking Vue as their main frontend framework. Among other things, these recent adopters have been praising Vue's incredibly smooth learning curve, ease of integration into their existing stack, top-notch performance, and—what's probably the most important factor—the improvements in development speed and reductions in maintenance costs. In other words, choosing Vue saves money.

But don't take my word for it just yet. To make our case, we surveyed 1,126 developers from 88 countries, and collected several case studies, hailing from a range of different industries, that have adopted Vue.

Read on.



How developers use Vue.js

We were curious about the experience with Vue.js among software developers and Chief Technology Officers. In an online survey we asked them a number of questions to determine:

- what were the reasons behind adding Vue to their tech stack,
- what advantages did working with this framework offer them,
- what were their main doubts with regard to using Vue in their project,
- what resources did they use to master Vue.js,
- how many of their colleagues use Vue and whether they expect this number to grow in the next 12 months,
- for how long have they and their teams used Vue.js,
- what other backend and frontend technologies are used in those companies.



Report Data

All report data comes from a survey conducted over a four-week period in August and September of 2017. We received 1,126 responses, mainly from software developers and Chief Technology Officers (94.1% of the respondents held these or related technical roles) whose organizations use Vue. The responses came in from every continent on Earth (except Antarctica); in total, we've managed to collect replies from 88 countries.

We also asked Evan You, Vue creator and Chris Fritz, Vue Core Team member to comment on some of the survey results, in order to provide additional insights or to share their broader perspective.



Key Insights

96%

survey participants would use Vue.js again for their next project.

94%

of the respondents used the Official Vue documentation as their main source of knowledge about the framework.

81%

of survey participants say that ease of integration is one of the chief advantages of having Vue in their organizations' tech stack.

54%

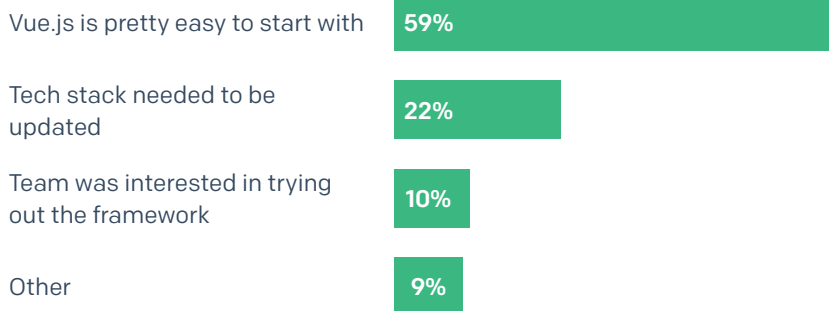
are convinced Vue.js is going to get more popular in their organization in the next 12 months.

▶ Survey Questions

Q *What was the most important reason behind adding Vue.js to the technology stack?*

Whether starting a new project or inheriting one—the developers are more or less unanimous: Vue.js is pretty easy to start with, even for a really complex applications. They appreciate its simplicity and architectural elegance, elaborate on how easy it is to integrate Vue, but also compare its lightness and performance to other popular frameworks, claiming Vue is the unquestionable winner here. All in all, **Vue.js is described as a beginner-friendly framework by more than half of the respondents.**

THE MOST IMPORTANT REASON BEHIND ADDING VUE TO THE TECH STACK



Could be used for existing and new projects + pretty easy to use!

Chief of Technology Officer,
Large enterprise, France

Easy to integrate with existing apps or go full SPA.

Software developer,
Medium enterprise, Australia

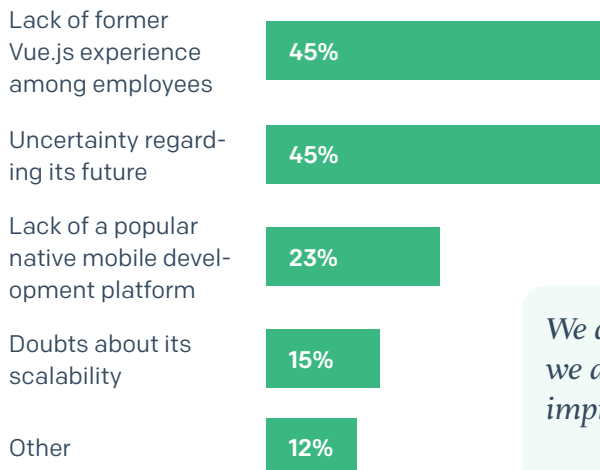


What were the doubts you and your team had when planning to add Vue.js to your tech stack?

The respondents mention two main doubts they had when planning to add Vue.js to their tech stack. First concern, expressed by 45% of respondents, is more related to their teams themselves. **Lack of former Vue experience** among their colleagues was seen as a possible issue when planning to add the framework to their tech stack.

DOUBTS WHEN ADDING VUE.JS TO TECH STACK

Percentages do not sum up to 100% due to the multiple choices.



Vue's mobile options are constantly improving. In the meantime though, Vue offers very strong support for Progressive Web Apps, including a dedicated template. Community projects like Onsen UI even simplify the process of building native-like, hybrid UIs.

Chris Fritz
Vue.js core team member



We do have Weex and NativeScript, but we acknowledge they both have areas to improve.

Weex has been used in production at Alibaba for quite a long time, and is essentially Alibaba's bet in the mobile dev space. But it lacks in terms of documentation and learning resources for English speakers. We intend to bridge that gap next year by providing official guidance on how to get started with it from the Vue side.

NativeScript is also a solid technology, and its integration with Vue, although relatively young, is being improved at a rapid pace. It's getting more impressive every day. Definitely keep an eye on it if you are interested in using Vue for native development.

Evan You
Vue.js creator



Lack of a popular native development platform was mentioned by the same percentage of respondents as their doubt when considering using Vue.js.

Doubts about Vue.js scalability were chosen by 172 survey participants, making it one of the five most prevalent doubts plaguing developers planning to add the framework to their current tech stack.

Architectural-wise, Vue is built on the component-based model for UI development which is a proven pattern shared among all major frameworks, with solid official solutions for SPA routing and large-scale state management. It is designed for approachability, but is also designed with scale in mind.

We also have many users successfully building large scale projects with Vue, some even with hundreds of components and are still perfectly happy with the workflow. In addition, some existing big apps are being rewritten with Vue and we've received very positive feedback from devs working on them, for example Adobe Portfolio and JSFiddle.

Evan You



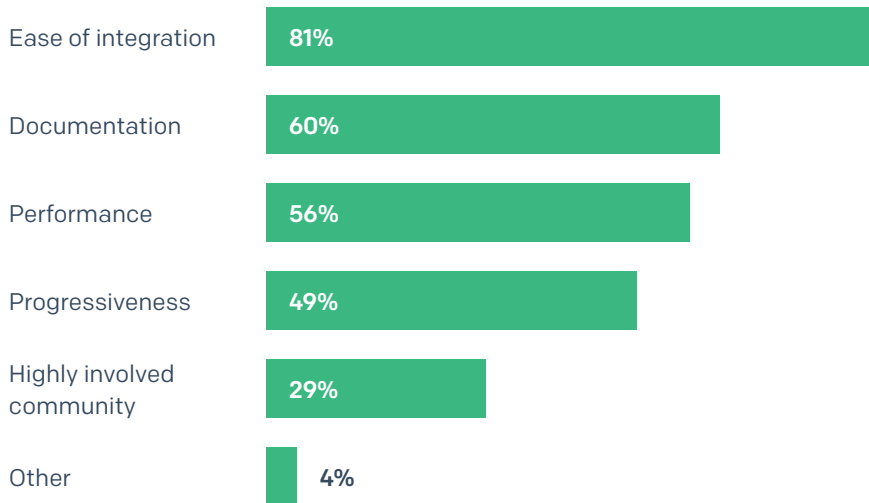
What are the biggest advantages that Vue.js brings to your organization?

An impressive **81% of developers emphasize ease of integration** when it comes to Vue. The majority remarked upon how easy it is to master Vue and claimed that it's much easier to learn than other popular frameworks. They also appreciate its **uncomplicated integration with backend frameworks**.

Documentation is another strong point of Vue, brought up by 60% of surveyed developers. A similar number of respondents (56%) identified the framework's performance as its biggest advantage.

THE BIGGEST ADVANTAGE

Percentages do not sum up to 100% due to the multiple choices.



The smooth learning curve make a lot of people interested into Vue.js.

Senior developer,
Medium-size business, New Zealand

We were debating React vs Vue and ultimately chose Vue and we are glad we did.

Software developer,
Medium enterprise, USA

Vue.js makes frontend development manageable and scalable. The learning curve is pretty easy so backend developers can easily get what's going on without too much guidance. Since there are a lot of pretty good webpack configs already out there, it kinda feels like plug and play nowadays. Finally, the fact that we can use Vue.js either via run time or compiling it, it's a pretty amazing tool to use from small to large applications without too much difficulty in scaling.

Software developer,
Small business, Philippines



Is there anything you're missing when it comes to Vue.js?

We received 481 valid responses to this open-ended question. Since some of the shortages were mentioned by more than 20 people, we decided to employ broader categories for them.

Lack of a native Vue mobile solution is one of the biggest pain points, spontaneously mentioned by more than 24% of respondents. There is definitely **a strong demand for a more advanced mobile solution for Vue.js.**

15% of those who replied to this question, identified a **relatively small ecosystem** as another Vue drawback. Once the ecosystem gets better and bigger it will be able to ensure a better collection of components. Additionally,

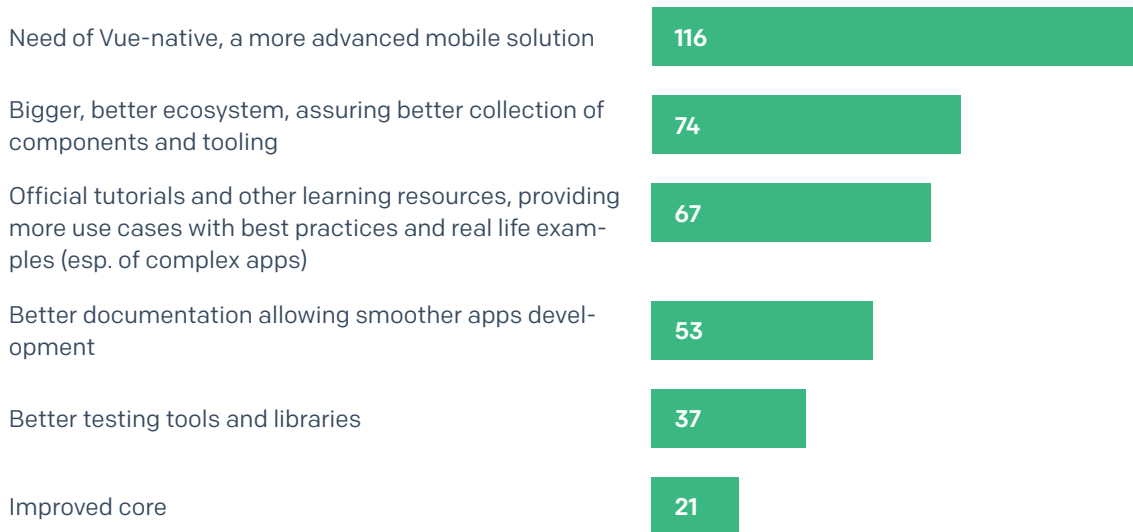
tooling is going to be improved even more with the next iteration of the CLI, assures Evan You.

In their replies, respondents also brought up lack of official tutorials, a “Vue bible” as one of them called it, or a comprehensive Vue Cookbook that would offer more real life examples, especially in complex apps. As Christ Fritz comments,

The recently released official style guide now provides a sort of Vue bible, but it was unavailable at the time of the survey.

There is also a need for **better documentation for the framework**, since 53 of respondents mentioned some issues related to it either directly (eg. *More architecture-related documentation regarding bigger apps*) or indirectly, mentioning some issues they wrongly assume are impossible to solve with Vue. The two final issues, identified by more over 20 respondents, were related to improvements in testing tools and the need to **improve the core.**

SUGGESTIONS



The cookbook, which will begin serious development in November, will help provide examples for larger apps, common integrations, and also explore architecture questions.

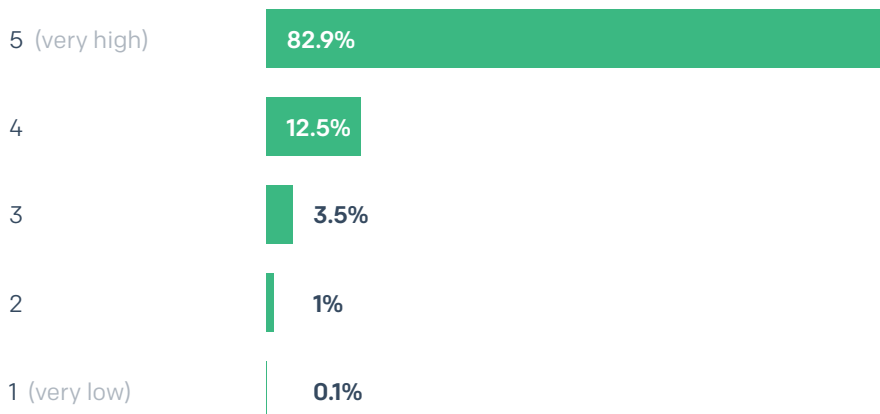
Chris Fritz



What's the probability of you using Vue again for a new project?

More than **95% of respondents claim they would use Vue.js again for a new project**. A number that high clearly proves that their doubts went away once they gained experience with the framework. Even though they mention its shortages and express the need for improvements, Vue is clearly appreciated by almost everyone who has ever used it and the overwhelming majority of respondents have no doubts about using it again for their next project.

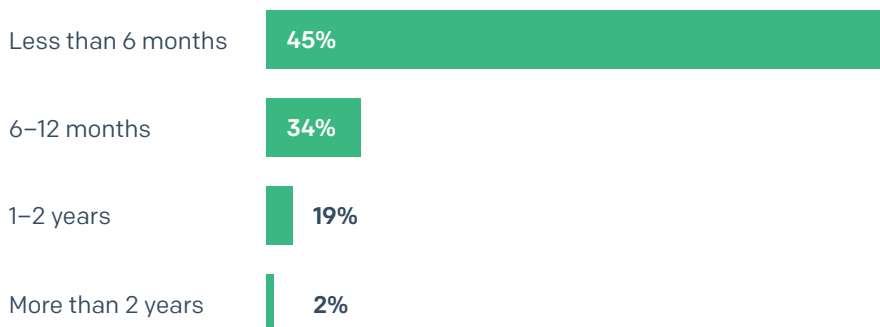
THE PROBABILITY OF USING VUE FOR NEXT PROJECT

*How long Vue.js has been used within your organization?*

With its growing community, dedicated events popping up all over the world, and its position among the top 10 most starred repositories on GitHub, Vue is gaining more and more traction. **More than 3/4 respondents added Vue.js to their tech stack in last 12 months.**

It may mean we should expect the number of Vue developers to grow rapidly over the next few years and the framework itself maturing, with a better ecosystem and more use cases.

HOW LONG VUE.JS HAS BEEN USED WITHIN YOUR ORGANIZATION?



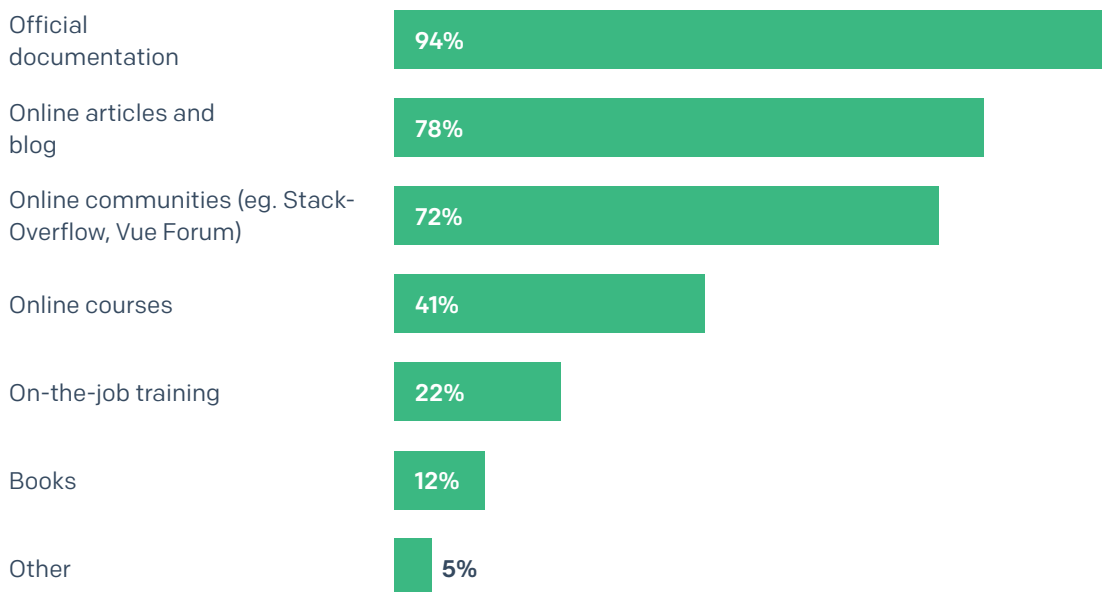


What resources do you use to learn about Vue.js?

The official Vue documentation is the most popular resource used to learn about Vue.js. It was chosen by 94% of software developers, proving that a well thought out documentation is a major strength of any framework. Additionally, online articles, blog posts, and communities such as StackOverflow or the official Vue Forum have been identified as a source of knowledge by over 70% of the surveyed software developers. Online courses have attracted 41% of them, while on-the-job training and books have served less than 1/4 of the respondents.

LEARNING RESOURCES

Percentages do not sum up to 100% due to the multiple choices.

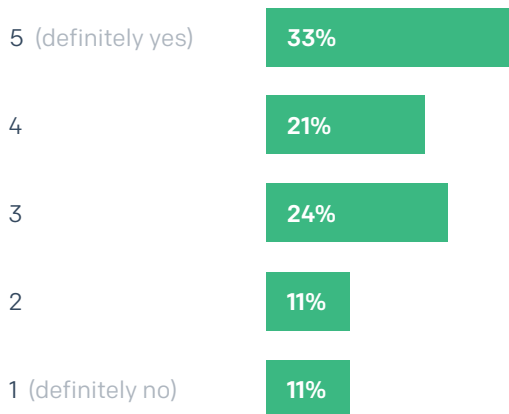


Do you think the number of employees using Vue.js in your organization will increase in the next 12 months?

54% respondents are convinced Vue.js is going to be more popular among their organization in the next 12 months. However, those who work

at large enterprises (more than 1,000 employees) are even more certain Vue is going to be widely adapted in their companies: 76% of them believe so.

INCREASE IN THE NUMBER OF EMPLOYEES USING VUE.JS



Other projects in the company are going to use Vue (or already do).

Software developer,
Large enterprise, France

We're hiring like crazy and have lots of projects coming up. They'll all use Vue.

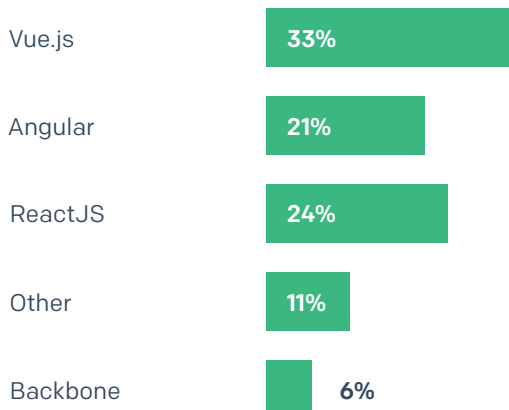
Head of Engineering,
Large enterprise, Germany



*What are the main technologies and frameworks you use for **frontend** development?*

INCREASE IN THE NUMBER OF EMPLOYEES USING VUE.JS

Percentages do not sum up to 100% due to the multiple choices.

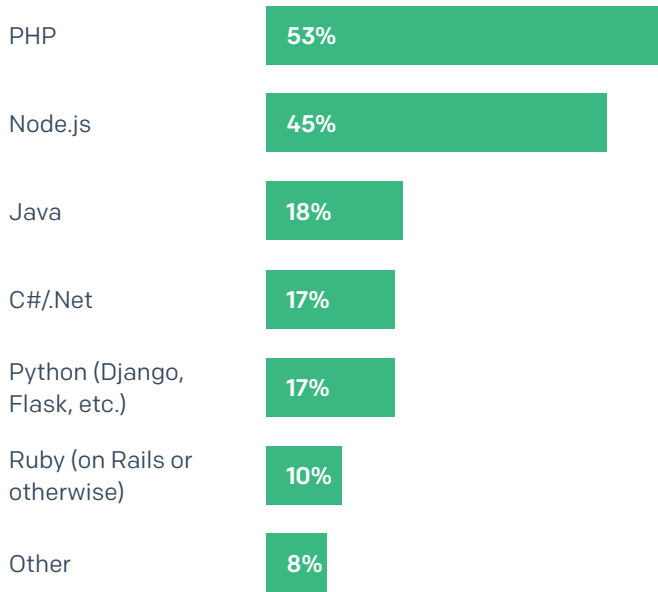




What are the main technologies and frameworks you use for *backend* development?

MAIN BACKEND TECHNOLOGIES AND FRAMEWORKS

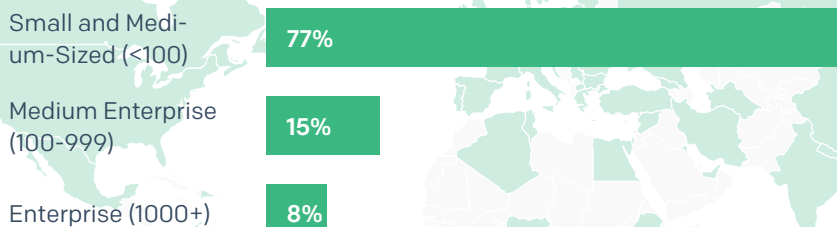
Percentages do not sum up to 100% due to the multiple choices.

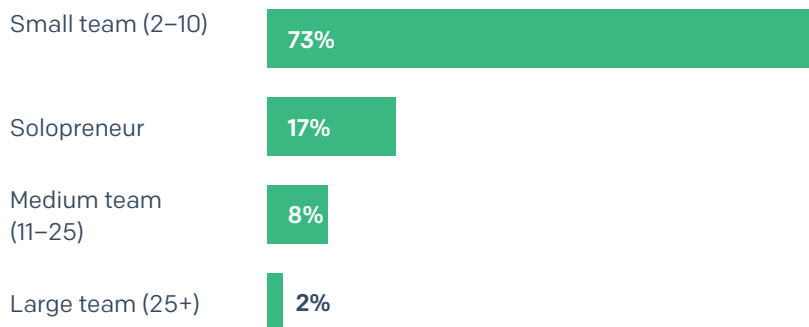
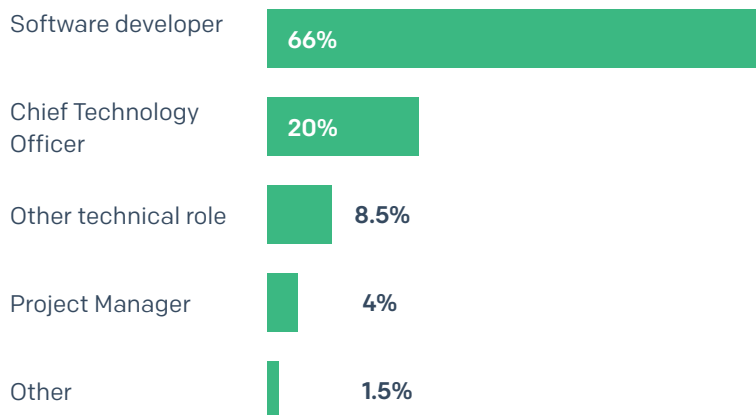


Demographics

We surveyed 1,126 software developers, CTOs, and other technical roles familiar with Vue from 88 countries.

COMPANY SIZE (NUMBER OF EMPLOYEES)



TEAM SIZE (NUMBER OF TEAMMATES)**ROLE IN ORGANISATION**

Case Studies

One of the reasons for drafting the *State of Vue.js* report was to provide a substantial body of evidence that Vue is a mature technology, adopted by companies of various shapes and sizes. Each presented case study proves that Vue is ready for commercial usage. All six companies we interviewed had faced the challenge of choosing the right framework, and they all decided to go with Vue—even though they are in different stages of growth and have different goals.

Before Codeship started working with Vue, their audience experienced freezes and browser crashes. They had a long list of users who were dissatisfied with the performance of the app. Their story is a great example how Vue can help build a reliable software, with a bulletproof, easy-to-maintain code.

If you're looking for a good illustration of how enterprise-level organizations use Vue.js, the Behance and Adobe Portfolio case study may come in handy. Their team built two independent products in Vue from the ground up—and they're not going to stop there.

In the Livestorm case study, Gilles Bertaux, co-founder and CEO, describes how they created a profitable product starting from scratch. Thanks to Vue and its reusable components, their development has been faster and much easier.

Jacob Schatz, frontend lead at GitLab, explains why they decided to move from jQuery to Vue.js and describes the main challenges they encountered along the way. Their focus on better UX resulted in a more desirable product and therefore in increased sales.

Chess.com had to deal with hard-to-maintain legacy code in Angular 1. With Vue.js, they found it much easier to collaborate within their fully remote team of 15 developers. Chess.com is a platform with a massive infrastructure serving 19 million users around the world. In their case study, you'll find out how Vue.js solved many of their issues.

The last case study differs a lot from all the others. Sylvain Simao, technical lead at Clemenger BBDO Melbourne, explains how they use Vue.js for projects with short lifespans—from 4 to 12 weeks. The biggest challenges they encountered included tight deadlines, working a lot with animations and transitions, and delivering highly-performative campaign websites.

Behance & Adobe Portfolio

Behance is the leading online platform for showcasing and discovering creative work.

Adobe Portfolio is a custom website builder designed to enable users to showcase their creative work.

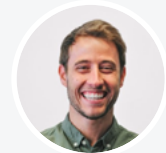


Erin Depew
Software Engineer
at Behance

We were a little hesitant since there aren't all that many major companies using Vue out there. However, every time I've had any issues, it's usually because I was overthinking it, and I was pleasantly surprised when it really was that easy to implement.



Yuriy Nemtsov
Software Engineer
& Manager
at Behance



Matt O'Connell
Software Engineer
at Adobe Portfolio

CHALLENGE

- Moving from homegrown solutions to community-supported technology
- Maintaining high user experience and performance
- Being able to share components between other teams and projects

SOLUTION

- Switching the Behance and Adobe frontend teams to Vue.js
- Using Vue.js to migrate the existing codebase

OUTCOME

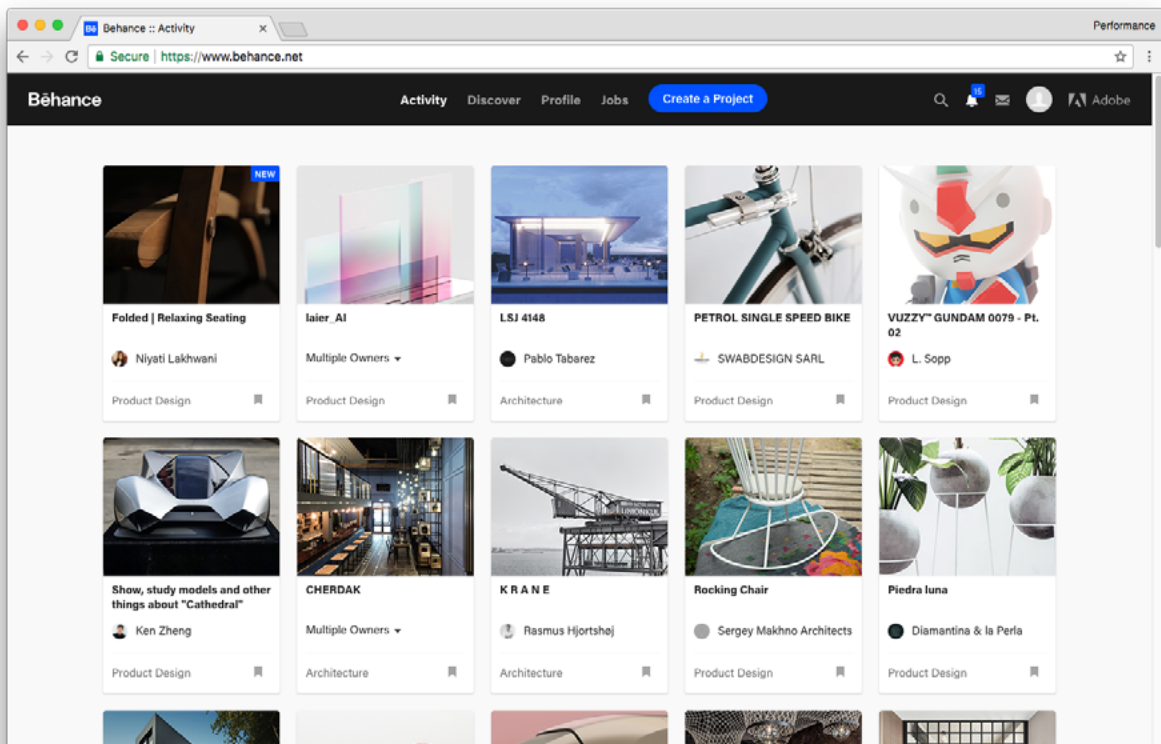
- Comfort of converting the site at an unhurried pace, without having to do it from scratch
- Ease of integration with existing codebases
- High performance and cost effectiveness

Challenge

Adobe and Behance, a company under the Adobe umbrella, have been leveraging the latest technologies and design thinking to create revolutionary products that connect and empower the creative world for years.

The team had decided it was time to update to a community-supported open-source framework since they were starting to hit the limits of the homegrown technology that they were currently using.

Before Vue, we were using a homegrown MVC framework that leaned heavily on Hogan.js (mustache) and jQuery. Our framework wasn't able to render declaratively, forcing us to imperatively keep the DOM in sync with the data. It also didn't have the ability to compose features into components, enforce a unidirectional data flow, nor a comprehensive documentation. So, even though it worked well for years, we were ready to switch to a framework that would allow us to build features rapidly with fewer bugs, facilitate their maintenance, and allow for quicker onboarding of new people, Yuriy explains.



Mustache specifically was important to us, because we were using (and still are, for the majority of behance.net) the same templates on the backend and frontend. The speed with which we can deliver the first meaningful byte to the browser is significant to us and our users. That same speed would be tremendously difficult to achieve if we were to wait for the browser to download the JS, parse, compile, and execute it, and only then display that project to the user. We were also looking specifically for a framework with server-side rendering capabilities, he says.

For the Behance team, the primary goal was to build a codebase that's easier to work with and a strong foundation for new features to be added going forward.

I think that one of the biggest challenges that we've faced is that since we've decided not to "split" our codebase and start with a clean slate, we had to spend a lot of time unraveling older code to form new components. That trade-off between refactoring older code to Vue while still maintaining the rest of the site and shipping features has definitely been challenging, Erin elaborates.

We also take performance very seriously at Behance, so we have been very careful to make sure that we can keep our performance metrics while converting over the codebase.

For Matt and his team, user experience was also an important factor, one leaving much room for improvement.

Regarding Adobe Portfolio, we started with nbd.js, which is a custom-rolled version of Backbone that was originally extracted from

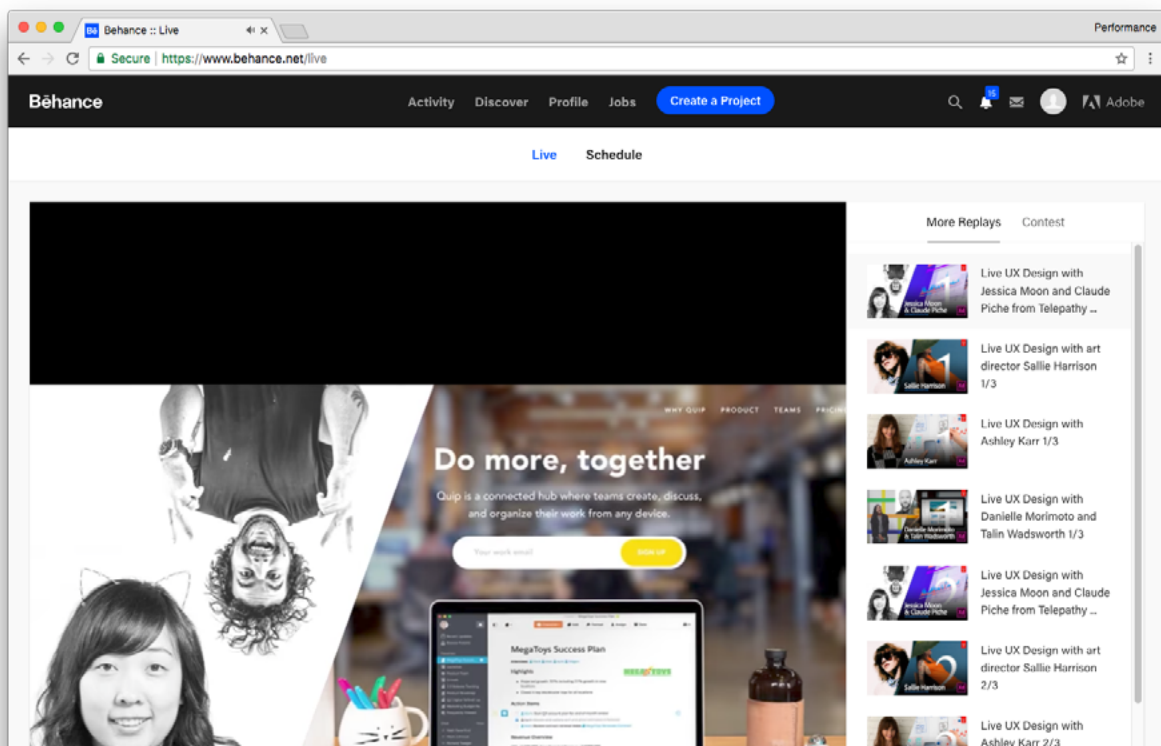
a product we no longer have, called “Action Method Online.” From there, we used that to build large chunks of the Behance network. It had limitations for reactive systems, so we built the “reactive” part of portfolio using Ractive, Matt adds.

The biggest challenge thus far has been, just like in the case of Behance, maintaining a fast user experience with complex user data state management, while providing instant feedback on both the content and styling of a user’s site.

Solution

Rather than establishing new developer teams who would focus only on Vue.js, both Adobe Portfolio and Behance retrained their existing teams to use Vue.js in their day-to-day work.

The vast majority of the team was here before we switched to Vue. Once we made the decision, we needed a few small projects to cut our teeth on. For us, that meant very small, frontend only features, and a set of features that weren’t publicly accessible, like our



style guide. This way, we could learn how to use Vue, how to write the tests, and to style the components relatively safely. Only then did we feel comfortable enough to take on a larger project. That was Behance Live, Yuriy recalls.

On Portfolio, our entire frontend team works with Vue—nine engineers in total. Some of our backend developers are also starting to pick it up as well. There are about eight frontend developers on the Behance product who program in Vue, Matt explains.

There is quite a bit of overlap between the two teams (Adobe Portfolio and Behance). We share a lot of libraries and APIs between our codebases and feature roll-outs usually appear on and require collaboration between both sites, Erin adds.

The Behance team encountered many challenges on their way to determine how to structure the application in general, and how to define the roles of different components.

The vuex store was also tricky to structure for a larger application. We decided to use namespaced modules. It wasn't clear at first whether there should be a single store-module per route/page or data type (e.g., user or project). Creating route-specific stores meant that the actions across routes wouldn't be reused. Making them data-specific was the best solution for us, with a top-level route store-module that combines the modules that the route needs. The solution is, however, still far from perfect, Yuriy says thoughtfully.

To define roles of various components, we make a distinction between a "page" component (the first component that the router points

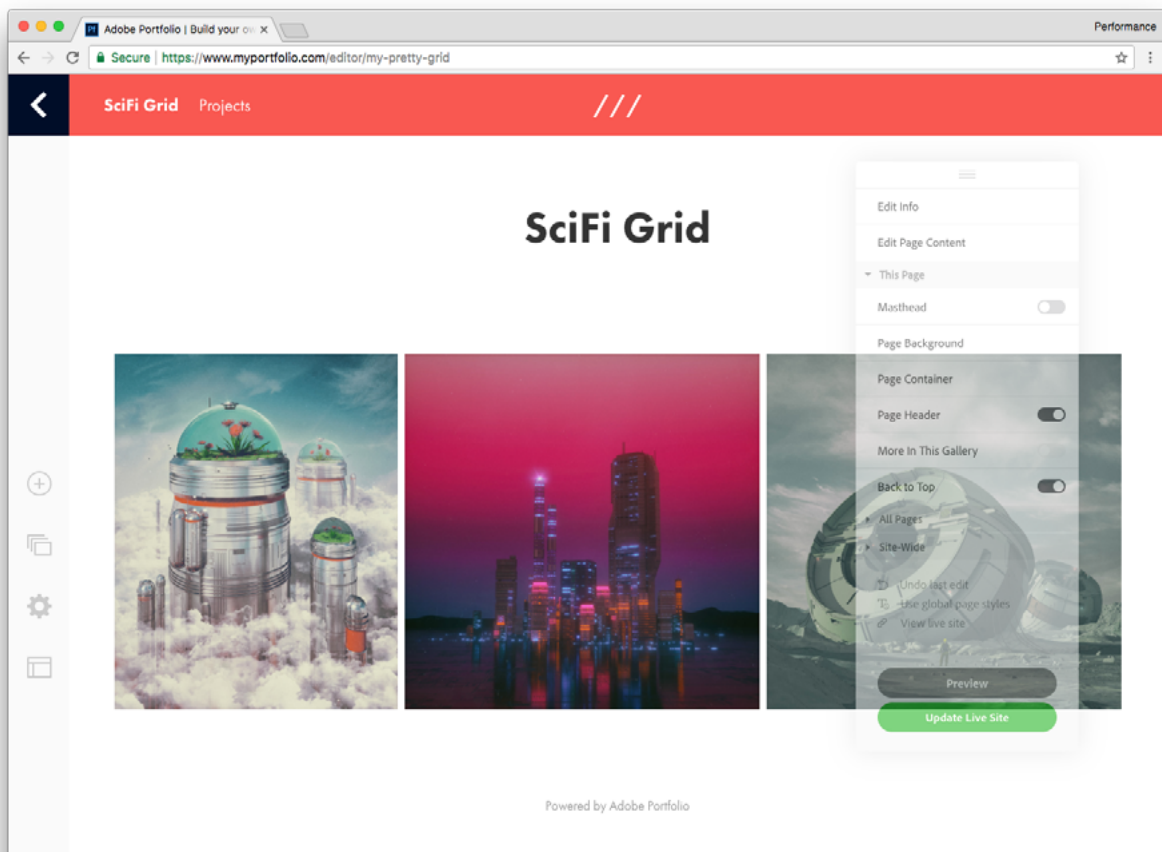
to and the one that interacts with vuex) and a “dumb” component (that solely sends properties down to child components and broadcasts events up to their parents).

Using Vue.js for almost a year now, Matt and his team managed to build or rebuild a bunch of features.

At Adobe Portfolio, we started with the Manage Content feature—the area where users can reorder, add, delete, and perform various other actions on their portfolio site. As needed, we created reusable UI components like select dropdowns, overlays, toggles, and drag and drop lists, Matt says.

Outcome

According to Erin, due to its progressiveness and great flexibility, Vue is easy to integrate with Behance’s existing codebase.



I always say that every framework is just another tool. However, one of the biggest benefits of using Vue, besides the speed and stellar documentation, is how well it integrates into our existing codebase. Unlike with other component-based frameworks, Vue has given us the luxury of mounting our components onto our existing pages, enabling us to slowly convert the site at our own pace instead of going all-in.

I'd say Vue exceeded our expectations. We were a little hesitant since there aren't all that many major companies using Vue out there. However, every time I've had any issues, it's usually because I was overthinking it, and I was pleasantly surprised when it really was that easy to implement, she adds, laughing.

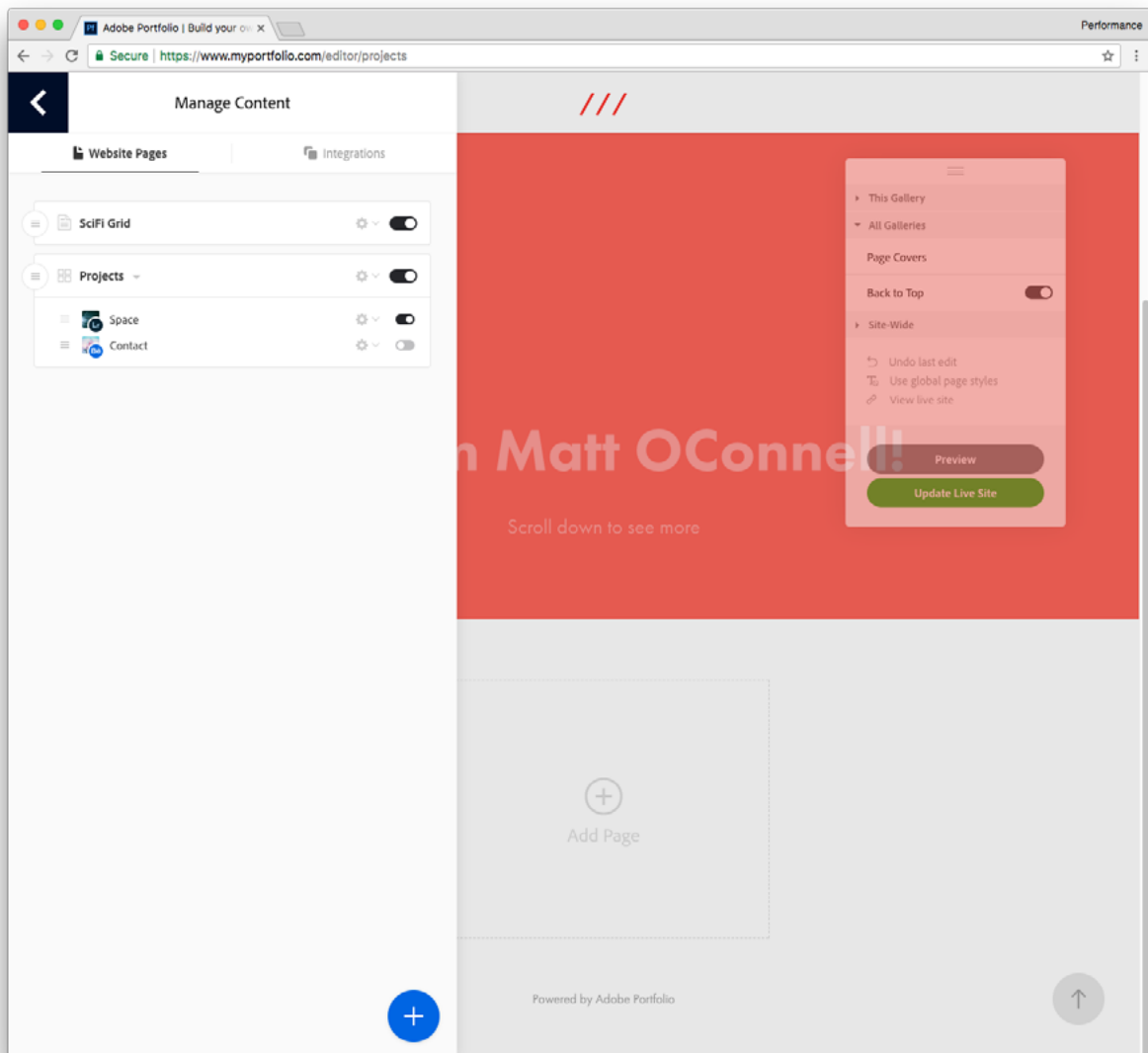
Currently, we're planning on converting our entire Behance codebase over to Vue, and, of course, recommending the adoption of Vue to other Adobe teams.

In Yuriy's opinion, Vue.js gives developers just as many possibilities as other frameworks. In contrast to some frameworks, however, it makes development easier and... less expensive.

I wouldn't necessarily say that Vue allows you do things that you couldn't achieve using a different framework. However, it was really difficult to squeeze proper SSR performance from React. Prior to the Fiber rewrite (React v16), a page with a large component tree would block the main execution thread, which, in turn, meant that if it took 100ms to render one page, all of the other clients of that Node server would just wait. So, we needed to either increase the number of processes-per-server or the number of servers

to increase throughput. This was difficult to maintain and more expensive than it needed to be. Vue's SSR story is much stronger. There's caching and streaming built in. As a result, even without spending a lot of time optimizing, the performance is good on Behance Live, he says.

Working with Vue.js definitely is different than with other frameworks. Somehow you just tend to enjoy your life more.



Chess.com



Scott O'Brien
Lead UX Engineer
at Chess.com

Chess.com is the #1 destination for online chess. Every day more than a million games are played by chess players from all around the world and all skill levels. Chess.com is a fully remote team with 100 team members.

It was the first time I've read the entire documentation in one sitting. It was 1:30am in the morning. By the time I got to the end of it, I knew that Vue.js was something special. There was something unique about it. Something I'd never seen before.

CHALLENGE

- Dealing with hard-to-maintain legacy code in Angular 1.
- Introducing new features to increase user engagement.
- Managing change in a fully distributed development team.

SOLUTION

- Benchmarking all available frameworks.
- Moving from Angular 1 to Vue.js.
- Architecting a growing library of components, each with its modular CSS.

OUTCOME

- Ease of collaboration with a fully remote team.
- More effective way to write CSS inside the app.
- Scaling up efficiently in terms of speed, power, and abstraction when compared to other frameworks.

Challenge

Chess.com is the most frequently visited website about chess with a great social network with over 19 million members. It has news, blogs, community, lessons, puzzles, and of course real-time gaming. The complexity of the portal may be overwhelming.

Its legacy code was in PHP and Angular 1. At any given moment, Chess.com is hosting tens of thousands of games in real time on the Web and mobile devices. For such a website, performance is everything.

We got to the point where the old way of doing things with Angular 1 was a huge performance bear. It was just getting bigger and bigger. Some parts of our website became unusable on legacy hardware from a performance perspective. It was unmaintainable, Scott recalls.

The challenge that Chess.com was facing was not only dealing with existing features, but also planning for future ones.



A lot of the discussion was about architecture because we knew we had a bunch of new features we are trying to add to keep people engaged to play more chess and try different ways of playing chess, Scott explains.

I wouldn't say it was out of reach with Angular, but it was very tough to do it well in terms of performance with those legacy javascript frameworks.

To improve the user experience, Chess.com required some real changes.

We knew we needed to take a leap. There was significant deliberation over which framework we wanted to move to from Angular 1. Of course, we considered the big players—Angular 2 and React.

The massive infrastructure and ongoing product development requires a well-organized and quite big-sized team.

We have a collection of very different skillsets within our development team. Moreover, the team is fully distributed and international—we are all over the map. Any decision as big as leaping to another technology brings a lot of concerns.

► Solution

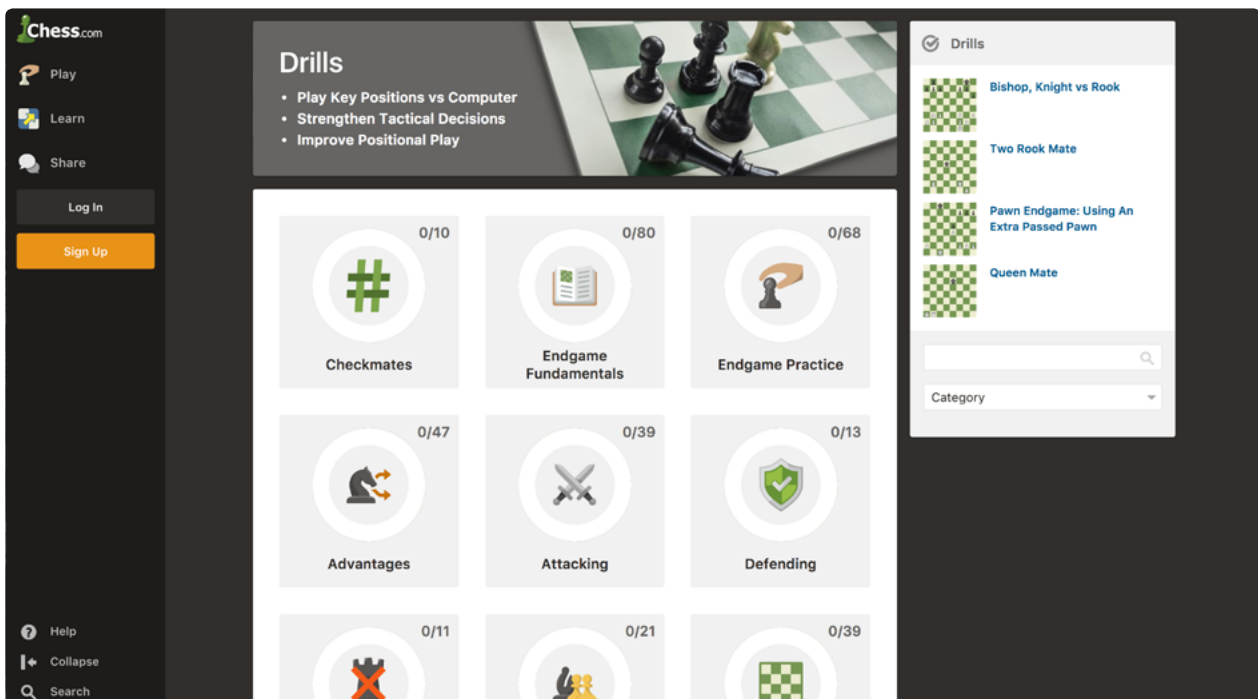
Choosing a framework backed by Facebook or Google, such as React or Angular, respectively, seemed to be a safer choice. Yet, the Vue.js community proved that the newcomer framework is definitely a contender.

We were so performance-centric that we would probably go with a less developer-friendly framework if the benchmarks looked good. Seeing Vue.js win the rendering and performance benchmarks was mind-blowing, Scott explains.

We were concerned that the entirety of Vue.js was built on Evan's ideas and that the framework would live or die with him. We decided that as long as the community was rapidly growing and we believe they are doing something revolutionary, we wanted to take the leap and believe that other people will see the value in the future that we see now. So the big concern was if it's going to continue to grow and I think that has been proven by now."

One of the first things that Chess.com team had to work on was rewriting different pages from AngularJS to Vue.

The process of rewriting is still going on today. It has been happening for months at this point. The other mission that we had was essentially building our internal collection of reusable components, Scott points out.



I think it's the most impressive thing we have been using Vue for—architecting a growing library of components, each with its own modular CSS, which will eventually comprise the entirety of user interface elements on our site. While one team has been using Vue to implement components, routes, and stores for particular product domains, another team has been working on our component library to be shared throughout the site with little to no concern for collision. Additionally, it has infused our product discussions with a greater sense of abstraction and reuse.

Outcome

For a big app like Chess.com, one thing about Vue brings more benefits than the rest.

Single File Components was an absolute game changer for structuring and maintaining our repository. Being able to just buy in to the official pieces of the framework having an official state management system. We're having confidence that these things are going to work together—it's all part of the collective vision.”

With Vue.js in place, Scott finds it easier to collaborate with his remote team.

What we love about Vue is that it has an incredible ease of use and a low barrier to entry while simultaneously having the ability to scale up with comparable (if not better) power, speed, and abstraction to other component libraries, he points out.

We're a fully remote team of 15 developers and we rely heavily on Slack, Jira, and GitHub. However, it's actually easier to collaborate in Vue, because it's not so different from our legacy code—there's still declarative templating and everything that we are used to.

Secondly, the ease of writing CSS now is amazing. It made such a tremendous benefit for us. We have many developers speaking different languages with different coding styles. Coming up with the names relevant only for markup in a particular file without worrying about the global namespace. The ease of use is just wonderful.

As Vue has lent great support to the Chess.com team, they will definitely continue working with it in the future.

We're all in with Vue.js right now! As I said, right now our process is twofold: essentially re-architecting our components and moving from Angular 1. Therefore, we're implementing it two totally separate ways simultaneously. That's elating.

The screenshot displays the Chess.com website interface. On the left is a dark sidebar with navigation links: Play, Learn, Share, Log In, and Sign Up (highlighted in orange). Below these are Help, Collapse, and Search. The main content area is titled "Master Chess Games" and includes a search description: "Search through millions of top games played by the strongest chess players of the past and present. From world chess champions to FIDE masters, you will find an enormous collection of games that you can search, sort, and download." Below this are four featured player cards, each with a photo and game count: Garry Kasparov (2,139 Games), Bobby Fischer (944 Games), Magnus Carlsen (1,901 Games), and Jose Raul Capablanca (765 Games). On the right, a "Games" search panel is visible, featuring a dropdown for "Openings", input fields for "Player 1" and "Player 2", a "Search" button, and an "Advanced" link.

Clemenger BBDO



Sylvain Simao
Technical Lead at Clemenger
BBDO Melbourne

Clemenger BBDO is a full service agency offering a full suite of capabilities including brand strategy, integrated creative development, CX, digital services, CRM, PR, design, shopper and activation.

In the last 12 months the agency has been named World's most creative agency at Cannes Lions and D&AD.

We decided to pick Vue.js because it was answering all the requirements we had to cover for our projects, while offering a comfortable development environment for our team. It's so close to native JavaScript, that it's extremely easy to start working with it.

CHALLENGE

- Projects with short lifespans (4 to 12 weeks) done by different people
- Working with animations and transitions
- Need to load and work fast on mobile devices

SOLUTION

- Using Vue.js with a pre-render solution for static pages
- Building ES6 modules rather than framework specific code

OUTCOME

- Delivered several successful interactive campaign experiences within strict deadlines
- Digital projects ready for high traffic volumes
- Quick onboarding and project setup



Challenge

Most of the projects Clemenger BBDO works on are campaign websites. It's mostly frontend with a little backend magic—most projects use the serverless approach, API, AWS services, and the like.

Working on different projects simultaneously under strict deadlines, Clemenger BBDO had to devise a standardized solution that would significantly increase development speed and be flexible enough to work across very different experiences.

As technical lead, one of the main things I need to keep in mind is the ability of my team to deliver high-end, quality projects within short time frames. We are an advertising agency, which means that a 3-month-long project is a really long one, Sylvain explains.

A fast-paced environment means that we need people to be able to jump quickly into new tools. Sometimes we also need to work with external contractors, so the perfect solution for us was something easy to learn and start working with. Vue gives us a lot of flexibility in terms of workflow—for example being able to work with already known preprocessors for HTML and CSS is a big plus.

On client projects, Sylvain and his team worked with a variety of different JavaScript frameworks.

I feel like we've tried all of them! Sylvain laughs.

We've tried frameworks like Angular, React, and Riot.js, but Vue is the one we fell in love with. Vue offers simplicity and robust-

ness at the same time. For us it's a breath of fresh air among the others. It has a rich ecosystem out of the box, and the fact that it's incrementally adoptable makes it the perfect tool for the type of work we have to deliver on.

Interactive campaign websites are challenging in many ways.

You have to deal with SEO, accessibility, and extended browser support—but also deliver on animations, transitions, and very interactive interfaces in general. Combining those is definitely the most challenging aspect of our work.

Solution

Due to its smooth learning curve, Vue.js makes it easy to work with new developers or external contractors.

We've noticed that Vue.js is great in terms of onboarding new people. Why? The learning curve is really smooth and it's really close to vanilla JavaScript, Sylvain claims.

For us, as a business, it's really awesome. People get up-to-speed really quickly and we can deliver more efficiently. One other remarkable point is the incredible quality of the official documentation and resources available for Vue. It probably deserves an award for the most comprehensible framework documentation!

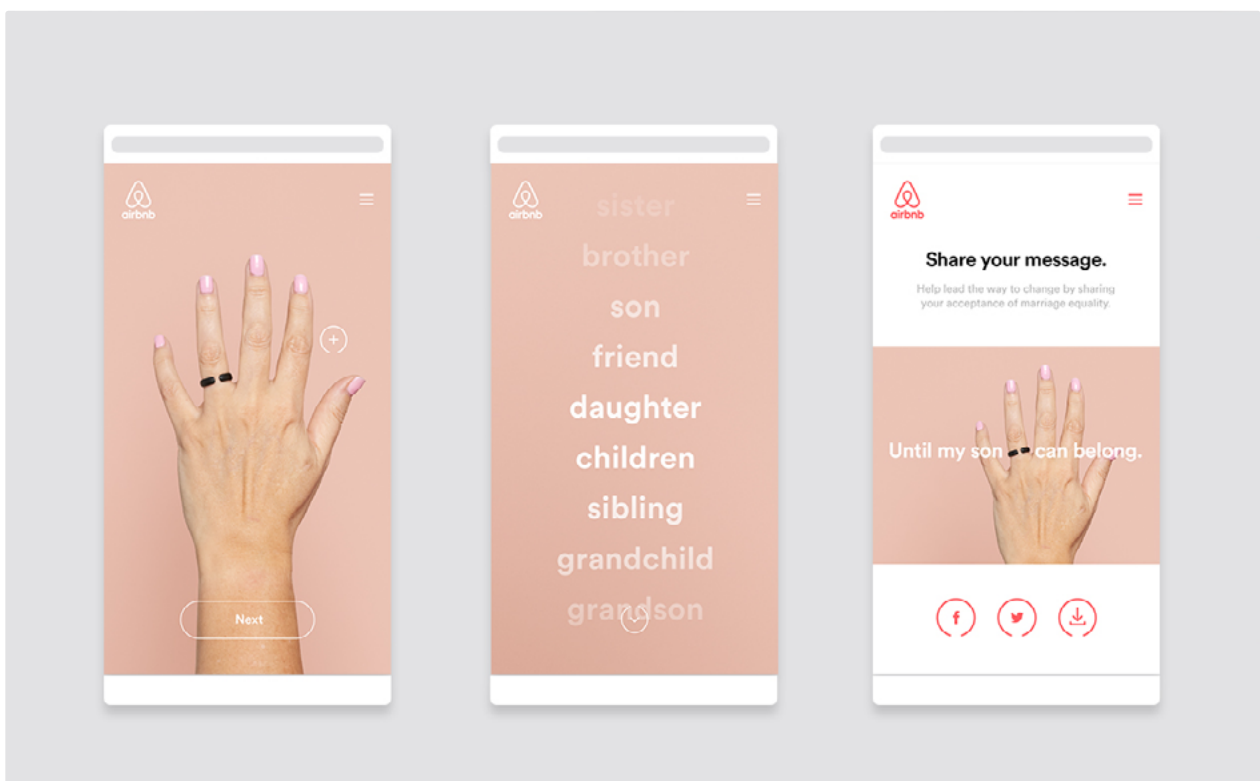
For every website Clemenger works on, it's important to make it visible for bots for successful SEO.

For that specific problem we do pre-rendering for all our pages. Most of the time, when we have a new project that requires Vue.js, we start from a boilerplate that we've built on top of the official Vue webpack template. Then, we use libraries like PhantomJS or Prep to render static snapshots of the pages. The last step consists of serving those pages to robots, which can easily be achieved by targeting the user agent with Nginx or Lambda@Edge, Sylvain elaborates.

Sylvain uses Vue.js to deal with animations and transitions.

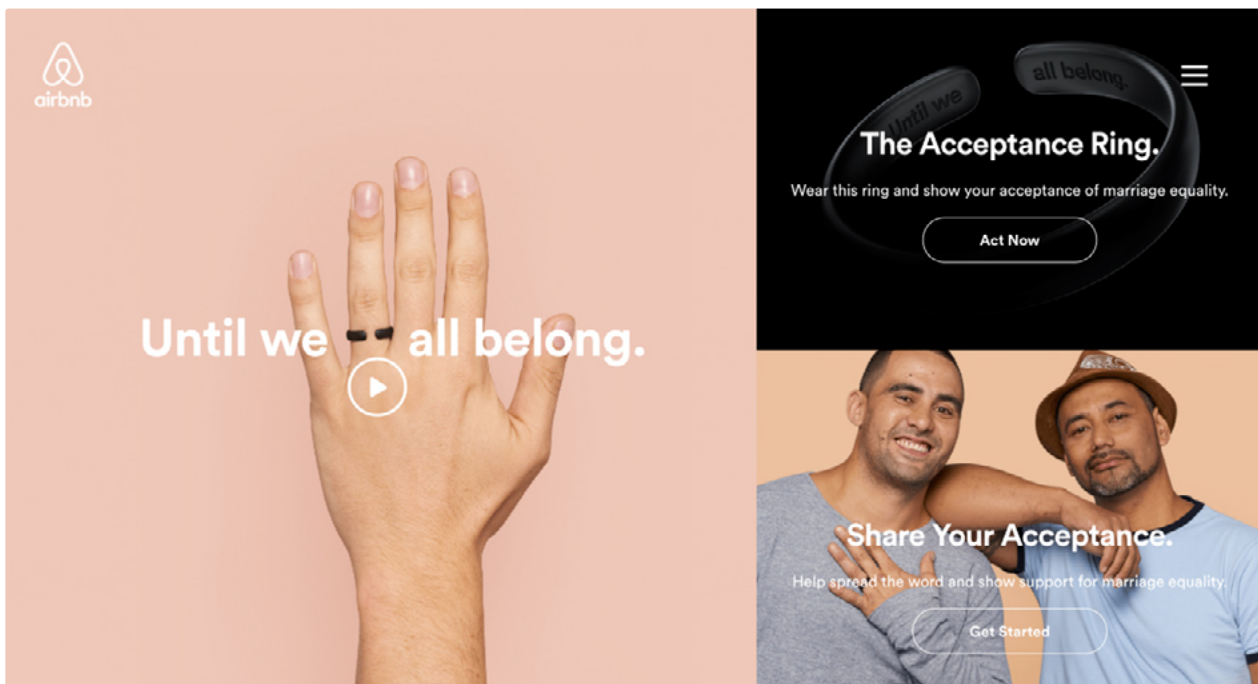
Right now we are changing our way of approaching animations. Since the release of Vue's most recent version, there is now a lot more flexibility with the transitions. We now have a more granular access to the transition hooks, that makes it possible to trigger third-party libraries and deliver on complex animations, while still using Vue at the core. I'm trying to push my team to move to that model.

For Airbnb's campaign website—"Until we all belong"—Vue.js was the technology of choice.



The project is designed as a single page application, based on Vue and webpack at the start. For better efficiency, the web views were hosted in Amazon S3 buckets, which means that we couldn't use any server-side rendering. Every part of the UI and every page have been built using Vue single file components. In this kind of website, where we are anticipating major traffic, performance is key, and that's why everything is loaded on-demand. In one of our projects we were recording 6,000 visitors per minute—it was a big buzz. We need to be ready for that, Sylvain explains.

Vue.js can be a lifesaver in such cases. For the Airbnb project there were big image assets in the background that we needed to load and animate. For that purpose, we've used Vue-router to declaratively list assets or data that required pre-loading, and Vuex to keep track of those on every page. The project was also challenging in term of interactions, but we've managed to deliver the website within 6 weeks.



Outcome

Delivering projects in a timely manner is much easier with Vue.js.

We wouldn't be as fast if it wasn't for Vue. Mostly because of the simplicity of the API. We've recently worked on a prototype for a hybrid app built on Angular 2, the syntax is elegant but the learning curve is steep and doing simple things takes time. With Vue you can prototype really quickly and that's probably its greatest strength.

With Vue.js, the Clemenger team is able to tackle a wide range of different projects

We now have quite a few projects built upon Vue.js. Airbnb's Until we all belong, a campaign for marriage equality in Australia, was recognized with a number of industry awards, including AWWWARDS and CSSDA. Another project—Meet Graham—which introduce the only person designed to survive on our roads, Graham. Within the first week, the project recorded more than 10 millions page views and it got immersive recognition and media coverage. It was highly acclaimed and received numerous awards, including the Grand Prix at Cannes Lion 2017. One of our most recent project is Snickers Hungerithm, where we've decided to rewrite the campaign app using Vue.js for the global rollout. Hungerithm is a hunger-algorithm that monitors online mood using tweets. When anger goes up, Snickers prices goes down in real-time.

Codeship



Roman Kuba
Lead Frontend Developer
at Codeship

Codeship is a Continuous Integration Platform in the cloud that lets you ship your apps with confidence. Open source projects are always free on Codeship.

Vue gave us all the flexibility we needed, to do what we wanted to do. It offers a solid foundation that can be expanded any way we like and it's not opinionated about tools we use in the pursuit of our objectives. That's what I really like about it.

CHALLENGE

- Freezes and crashes inside the application.
- Difficulty running unit tests with Angular.
- Ambitious plans for new features and building new, complex things.

SOLUTION

- Building a proof of concept and convincing other developers to give Vue.js a try.
- Moving away from acceptance tests only.
- Refactoring and rewriting pages.

OUTCOME

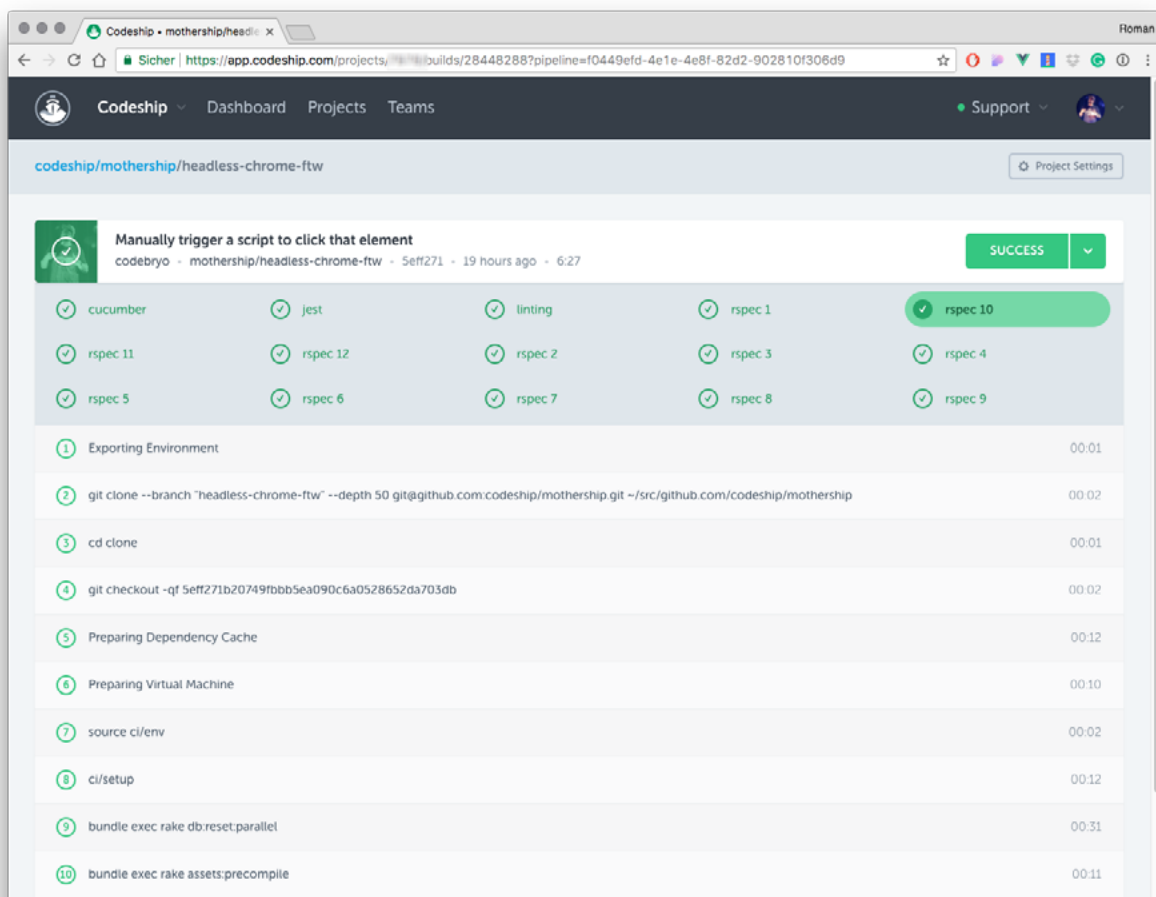
- Not a single crash of the app since Vue.js was implemented.
- Bulletproof, reliable, and easy-to-maintain code.
- Positive feedback from customers satisfied with the current UX.

Challenge

Codeship, a CI platform used by companies such as CNN, Red Bull, and Product Hunt, launched in 2010. With jQuery and CoffeeScript in their stack, they built a successful platform for developers around the globe.

But as time went on, the team realized that it's time to find a technology that would support further development and facilitate building more complex things.

To give you some perspective—Codeship is used by a ton of customers who rely on it in their day to day operations. When we're working on a feature for, let's say, four months, it somehow feels bad, like we're holding something back from our customers. If we spend two months fixing features, this in turn means two months of pain and uncertainty for them. It is absolutely crucial for us to be fast and deliver reliable products, Roman says.



The screenshot displays the Codeship web interface for a project named 'codeship/mothership/headless-chrome-ftw'. The top navigation bar includes 'Dashboard', 'Projects', and 'Teams'. The main content area shows a build status of 'SUCCESS' for a job titled 'Manually trigger a script to click that element'. Below this, a grid of test results is shown, with 'rspec 10' highlighted in green. A detailed list of build steps follows, including 'Exporting Environment', 'git clone', 'cd clone', 'git checkout', 'Preparing Dependency Cache', 'Preparing Virtual Machine', 'source ci/env', 'ci/setup', 'bundle exec rake db:reset:parallel', and 'bundle exec rake assets:precompile'.

Step	Duration
1 Exporting Environment	00:01
2 git clone --branch "headless-chrome-ftw" --depth 50 git@github.com:codeship/mothership.git ~/src/github.com/codeship/mothership	00:02
3 cd clone	00:01
4 git checkout -qf 5eff271b20749fbbb5ea090c6a0528652da703db	00:02
5 Preparing Dependency Cache	00:12
6 Preparing Virtual Machine	00:10
7 source ci/env	00:02
8 ci/setup	00:12
9 bundle exec rake db:reset:parallel	00:31
10 bundle exec rake assets:precompile	00:11

We have pages that basically show the complete terminal output as readable log for our users, so they can see what test went through and what output they have. It was clear very soon that something like jQuery, on which our product was running before, wouldn't cut it anymore because of the growing complexity, Roman reflects.

We started working with Angular 1, which we used for the next six months. Why? Well, mostly because I was already familiar with it.

The company switched to Angular and it was a good fit. Yet, as the service grew, it soon became apparent that sticking with it would not be feasible in the long term.

One of the things that we tried to improve was the performance. That was the biggest problem with Angular. The sheer amount of data we needed to present on the build page was way over Angular's capabilities. Customers were reporting serious issues with the app—the page was unresponsive, and some people were even experiencing freezes and browser crashes.

Roman, however, didn't want to give up on Angular right away.

Of course, we tried to optimize as much as possible. I even tried to move parts of the rendering out of Angular's default list rendering and use plain JavaScript instead, but it didn't work, Roman sighs.

At some point, Angular was trying to grasp what was going on the page as Angular tried to keep track of its scope and ran the relevant digest cycles... That killed performance, regardless of whatever we tried to assuage the hit. There was no way it could have run smoothly.

Another significant challenge that Codeship faced was improving the testing process and making the app more reliable.

With Angular, we still leveraged acceptance tests as much as possible. We basically ran user stories in the entire application. It was incredibly painful to run unit tests with Angular itself and test component, module or controller only. It was barely giving us the full picture we so badly needed, Roman explains.

Solution

The first step of transitioning away from Angular was getting the approval of the staff and the VPE.

At first, it was a struggle to get everyone on board with with Vue. The team had never heard of it, whereas they knew of Angular 2 and knew that Google was throwing its weight behind it, and they knew about React, backed by Facebook, Roman says.

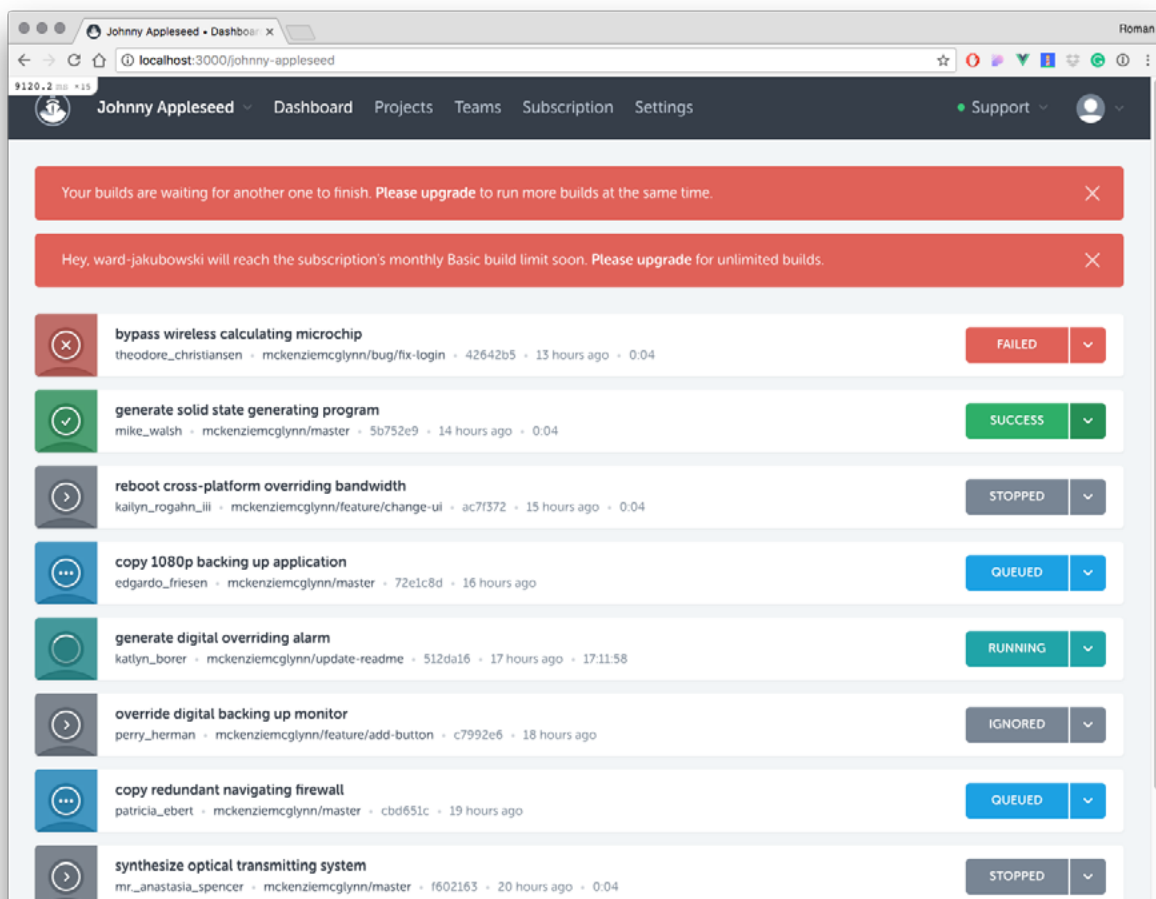
In conversation with the team, usually the first question was about the size of Vue.js community—people wanted to know whether there was any chance of getting support should the need to do so arise. Because the majority of our staff has roots in backend, they wanted to stick with trusted names they heard of.

Roman decided to use his knowledge and research to convince them to move to Vue.js.

I made some samples and an internal presentation to at least make them trust the decision and the reasons behind it,” he says. “If you read briefly through the source code of Vue, you’d quickly see that the code is not so hard to extend by yourself. It’s not a giant entity like Angular or similar.

Before Codeship jumped straight to development, they needed a proof of concept.

By then I had little experience with Vue, my expertise in the framework was definitely limited. However, starting with Vue seemed effortless and I quickly felt that it would be the solution to most of the problems that plagued us. In one evening or so, I rebuilt a crucial part of our rendering with Vue, and tried it against a large amount of Loglines as a proof of concept. Then

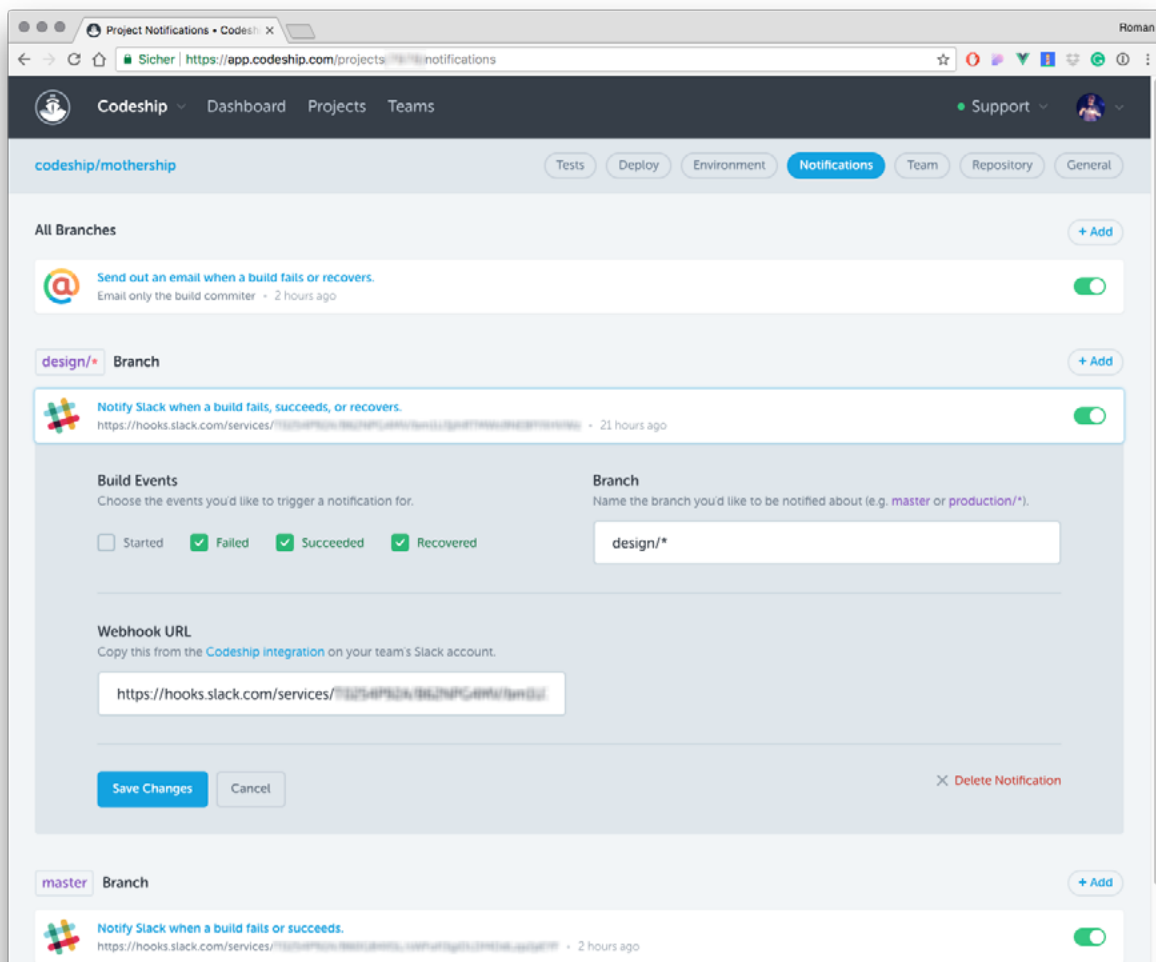


I did all the CPU profiling. That was the one thing that immediately demonstrated to my team that Vue.js has already given us a huge performance improvement. We cut the rendering time from 30 seconds to 7 or so, Roman recalls.

Proof of concept in hand, Roman and his staff could finally start the transition to Vue.

We tried to move the proof of concept and replace what we had with Vue. The actual risk here was quite minimal. We had a system that was breaking for users, so... What was the worst thing that could happen? Roman laughs.

I spent a week refactoring and rewriting the page and then shipped it to users for feedback, trying to get validation fast. After just one day we knew that all that problems that plagued us in the past



were gone. Even with 15Mb of log rendered. With rendering times between 30 and 40 seconds (we're currently working to decrease that number further), the app works splendidly across all browsers and we haven't recorded a single crash.

Moving from acceptance tests made the testing part more pleasant and reliable.

We moved away from acceptance tests and started wondering what we can take away from and use Jest and Vue tests for. We use multiple components even for complex pages in Vue itself, but only test it through Jest for example, because we have snapshots and verify whether the render HTML is actually the one we want, Roman explains.

Outcome

Engineers who rarely do frontend work now feel empowered to touch pieces of code.

Angular, with all its structure, modules, models, and controllers, and dozens of other things... introduces an unnecessarily high level of complexity. For these engineers, most of it sounds like weird magic spells. But when they actually saw Vue.js, they felt empowered to dig into it right away. That's a pretty big win for our company, Roman reflects.

Vue.js helps Codeship organize their code and improve user experience.

It helps us deliver features faster and our users love that they don't have to wait for months for something they need or expect. One of our pages that was running on jQuery had this weird kind of structure. We moved it to Vue and rebuilt it. Now, it provides a more granular experience and interaction with the UI, and so it significantly improved the UX. People tell us about it all the time.

With jQuery, the code would be very messy, hard to maintain. With Vue, it's different, you're able to harness the power of its components and start leveraging its ecosystem, like Vuex for example. What we do right now is page state management—something we have never done before, not in such a clean way.

For Codeship, Angular testing was an incredibly painful process. With Vue.js, they know that their code is bulletproof.

Vue.js seriously upgraded our testing protocols. We have Jest in place, which is a smart test runner for us. With Vue, we feel like we have way more control over every aspect of the app, Roman elaborates.

I can run 15 tests that try to perform specific actions. Such an approach allows me to easily identify code breaks. This is not something I could do this way in an acceptance test due to the sheer amount of time it would take to run. And the result is simply not worth the effort. Unit tests are great in that respect. Code-wise I know that it's bulletproof, as we test it in a completely new way and it's incredible.

GitLab



Jacob Schatz
Frontend Lead at GitLab

GitLab is an open core, integrated solution for the entire software development lifecycle.

Every framework will struggle in certain areas. With Vue.js, every struggle will be your own, not Vue's. It's just a perfect framework.

CHALLENGE

- Difficulty with implementing complex features and maintain current ones.
- Large Rails + jQuery application that was hard to scale.
- Insufficient app speed.

SOLUTION

- Gradually introducing Vue.js to GitLab to be used along with jQuery.
- Using Vue.js for all applicable new features and migrating old ones as needed, without doing a complete rewrite or refactor
- Using webpack to create optimized bundles.

OUTCOME

- Easier maintenance of a unified style guide within the whole code-base and code architecture.
- Highly improved time and cost efficiency.
- Improved user experience leading to better sales thanks to the ability to implement more sophisticated features.
- Improved page load times by reducing asset size.



Challenge

After six years on the market, GitLab has established itself as a well-known solution for developers hailing from thousands of companies. Only two years back, most of its code was still written in Rails and jQuery.

By 2015, the company had no frontend developers on staff and that set-up worked really well. Rails devs were writing frontend code and doing a fantastic job. Yet, the company's plans for the future required a new solution.

When I came in I saw that we have these individual jQuery things which were very simple, but for more complex things that we wanted to do, the really big ideas that we wanted to achieve, we would need something else, Jacob explains,

The screenshot shows the GitLab Pipelines interface. The left sidebar contains navigation options: Overview, Repository (8,830), Issues (474), Merge Requests (474), CI / CD (selected), Pipelines, Jobs, Schedules, Charts, Snippets, and Members. The main content area displays a table of pipeline runs:

Status	Pipeline	Commit	Stages	Duration
canceled	#11501382	enhance-logging -> a2d8acbc Re-add underscore to after_j...	[Progress icons]	00:22:54 43 minutes ago
running	#11501247	tc-test-admin-... -> ddf892b9 Put loggers in before_action f...	[Progress icons]	
canceled	#11501139	enhance-logging -> d1291186 Add logging for all web authen...	[Progress icons]	00:02:01 about an hour ago
running	#11501104	fly-out-top-le... -> 4728d839 spec fixes	[Progress icons]	
running	#11501058	fix/sm/33281-a... -> f7da15ba Use before_save :set_protect...	[Progress icons]	
skipped	#11501019	32665--refactor... -> 3b6d8bdc remove unnecessary props	[Progress icons]	
canceled	#11500979	enhance-logging -> f884bdc9 Add logging for all web authen...	[Progress icons]	00:01:13 about an hour ago

jQuery is fantastic, but it creates potentially more bugs as you're responsible for literally every state change.

To meet their goals, GitLab started looking for a new solution.

We checked Backbone, which I had previous experience with, we obviously looked at React, but also Knockout, Ember, and all those different frameworks. I wanted to do a small project using each. By then I didn't even bring up Vue.js at all! Jacob recalls.

Testing all these frameworks helped Jacob identify their advantages and drawbacks.

Backbone has a lot of tools to get stuff done and a good structure, but you're still in the same boat as with jQuery. With React it scared me a bit to get involved with a framework that depended on a big company. Also, it didn't seem to scale well for me. I really liked Mithril! The only problem is that it's not pretty to write at all. If they could put a couple of extra layers of niceness on it, I'm sure people would start adopting it.

Another big challenge was to make a mature switch to a new technology. It was a bit risky, and thus had to be well executed.

In GitLab, we have tons of code. When I joined, there were already eight thousand lines of JavaScript in our codebase. I obviously didn't want to do something that would be a complete rewrite. We actually still have some chunks of jQuery here and there.

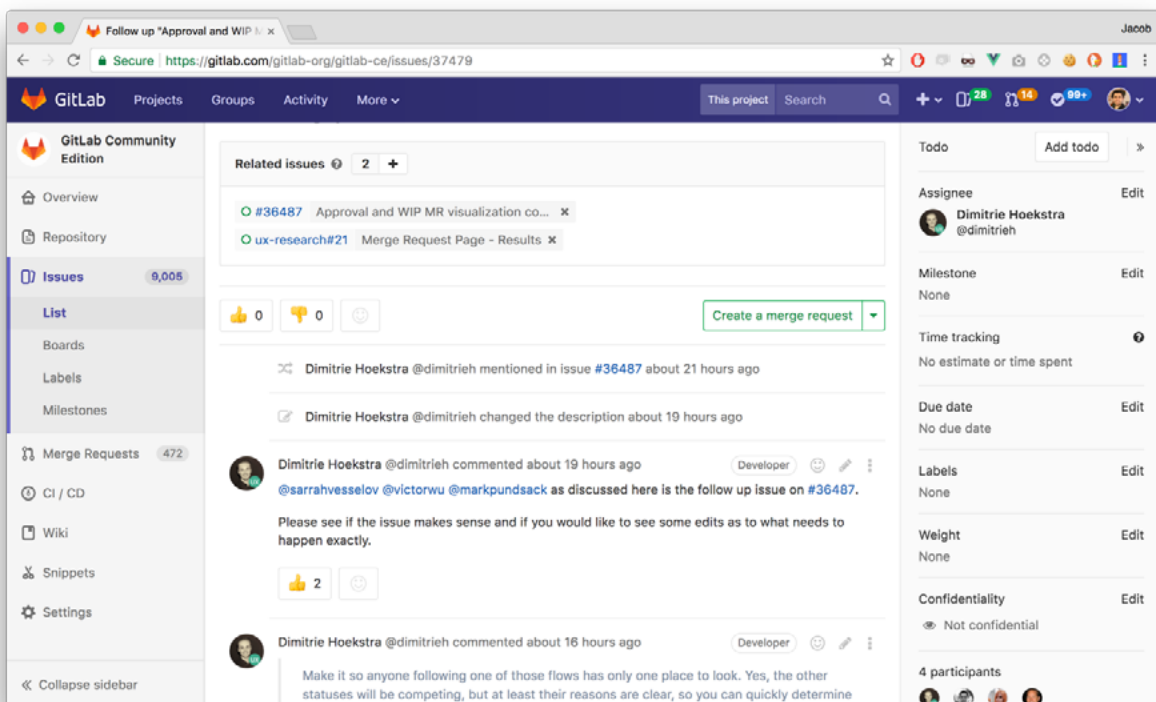
Solution

After testing a few frameworks, Jacob still didn't have a perfect match on his hands. It was only after he wrote a pretty big project with an early version of Vue.js he realized that he may have struck gold.

When I had this one project together, I knew it was something we could write a lot of code with. It wasn't just about writing a simple to-do app. When you get to work on this large application—that's where all problems actually start, Jacob explains.

Before GitLab started diving into Vue.js, they needed a proof of concept.

Phil Hughes [Sr. Frontend Engineer at GitLab], created a proof of concept where we took a major feature that we were doing—issue boards. Phil wrote that using Vue.js and it was immediately apparent



that we got a tremendous amount done in a small amount of time!

Without all those bugs which typically came with jQuery, Jacob says.

Vue.js supports the approach evangelized by Jacob within his team—it-erate small and create proofs of concept.

We constantly have 4 or 5 proof of concepts going on, he says.

Using the same approach, GitLab introduced webpack to be able to split the assets into smaller chunks downloaded by the browser and thus improving the app load time.

We created a small proof of concept to see if webpack was even feasible. When we found out that it was, we went the whole way and ended up writing and entire Trello application in Vue. And replacing the billion-dollar industry in one month. Good job, Phil! Jacob laughs.

One feature of Vue.js turned out to be the most useful out of all—reactive templates.

It's a very, very simple thing that Vue does. One of the first things I programmed in GitLab was to take the issue page, and when you clicked close, you had to refresh the page. And now when you click close, it just changes the status of the merge button, changes the status of the button below. It does all those things automatically. In jQuery, there was a ton of code. At least 30-40 lines to make sure that the buttons were in the right state. With Vue.js it was literally one line of code. The view always reflects the current situation, Jacob explains.

And now that we use Vuex, it can be done better than before. The state management stuff made a HUGE difference.

With all its advantages, Vue turned out to have one drawback.

Currently, there are 15 developers at GitLab. With frameworks like Angular you kind of work in the same way together. Vue is much more open, and so we had to create documents explaining how we write in Vue.js. What patterns you're going to follow. Still, it's something that we've solved. The openness of Vue is also its beauty, but you need to make sure everyone's on the same page.

VUE.JS STYLE GUIDE BY GITLAB

► Outcome

Scaling up the application and introducing new features would be possible with jQuery, but it would be much harder to maintain.

What we do right now would require a tremendous amount of code and a lot more organization. Vue has a lot of these problems solved, Jacob says.

With something as reactive as Vue.js, you give it a variable, and it's going to bind it into the DOM directly and take care of everything else. Especially in Vue 2.0 with its virtual DOM. We wanted to increase our performance, and it was one way to simplify our workflow.

Thanks to Vue.js, GitLab can iterate quickly and improve their usability.

We can finally focus on usability and UX, where before we were focusing on little tiny things and code. Now we think about the much bigger picture.

Vue.js is so open and accessible that it's pure pleasure for GitLab frontend developers to deal with it on a daily basis.

Vue, in contrast to other tools, does not follow any strict guidelines. It's open and that's fantastic. I like everything what it does right now. Of course, it's got the most amazing documentation you can imagine. It's really straightforward to get started with it and onboard new people.

Vue.js helped GitLab improve time and cost efficiency.

We know for a fact that our development is faster. That's an easy thing to see. From the sales perspective, those nicer UX features we're creating bring people to GitLab and make it a much more desirable product. People love the new things we put in there with Vue.js. We increase the user experience, and so we increase sales.

Jacob agrees that they will definitely use Vue.js in the future.

We're all set! We have other challenges now. Currently, we're trying to improve our process and speed up our testing. Vue.js solved so many problems for us that we're definitely keeping it for the future.

Livestorm



Gilles Bertaux
Co-founder
& CEO at Livestorm

Livestorm is a all-in-one web-based webinar solution. It helps companies like Workable, Pipedrive or Instapage do live sales demo or customer training.

We didn't have to spend a month setting everything up like with React, Vue had us operational in a week. We would never be at the point we are now if it weren't for Vue. I'm 100% sure of that.

CHALLENGE

- Building reliable real-time webinar software from scratch and having it make an impact in a highly competitive market
- Only a handful of Vue.js experts in Paris.
- Attracting initial customers and validating the product idea.

SOLUTION

- Building a quick MVP.
- Using Vue.js and Ruby to create a high-performance app.
- Appearing at Vue.js community meetups to seek out potential hires for the team.

OUTCOME

- Instant positive feedback from customers.
- Reusable components and fast development.
- Rapidly growing business with 20-30% revenue growth month to month.



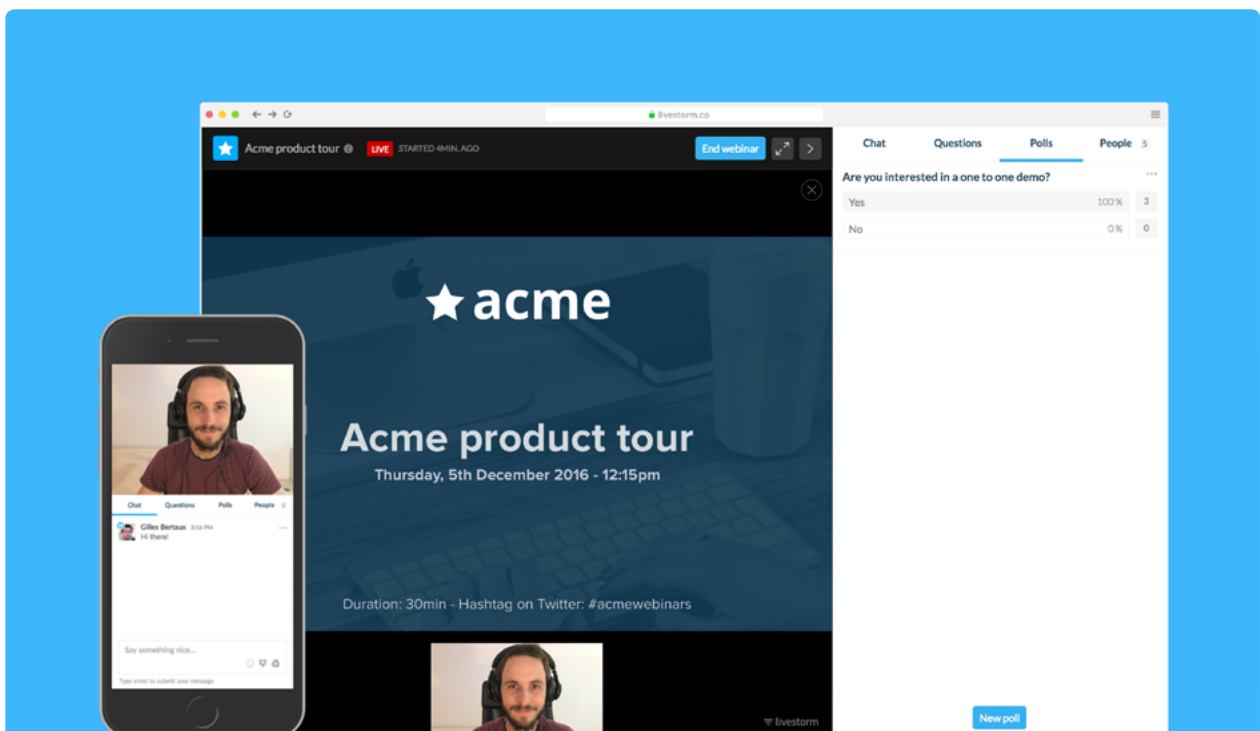
Challenge

Unlike other webinar platforms, Livestorm renders everything in the browser. The service provides actionable insights through analytics, integration with popular CRMs, and marketing automation software.

For an app like this, Gilles and his team had to pick a high-performance tech stack. Starting from scratch, they were intent on validating their idea and building a stable and reliable product.

The backbone of Livestorm is a Rails app—everything in backend is made with Ruby. For all our frontend components we chose Vue.js, Gilles explains.

We started working on our project in January of 2016 and from day one we knew we would be using Vue. We needed something that's entirely open-source, highly performant, with a particular logic of components. Vue was the only framework that met all our requirements.



A startup created by four co-founders, Livestorm sought to assemble a robust staff from the company's very beginning.

We thought a lot about hiring. There were only a handful of Vue.js developers in the Paris area where we work. We also considered recruiting developers proficient in other frameworks similar in structure to Vue, but in that case the onboarding process itself could have taken much longer and that was problematic for us.

To build a successful live streaming product, the team had to focus on reliability.

Reliability is a top priority for us. If you lose the live stream, webinars and demos crash and lose stream, our business basically makes no sense, Gilles says.

If the app's down or there's a bug making it impossible to use, we lose customers. We needed a technology that would somehow both guarantee top code quality and run really fast. We're still working on implementing end-to-end unit testing. Something we haven't done yet with Vue, it's completely new to us.

▶ Solution

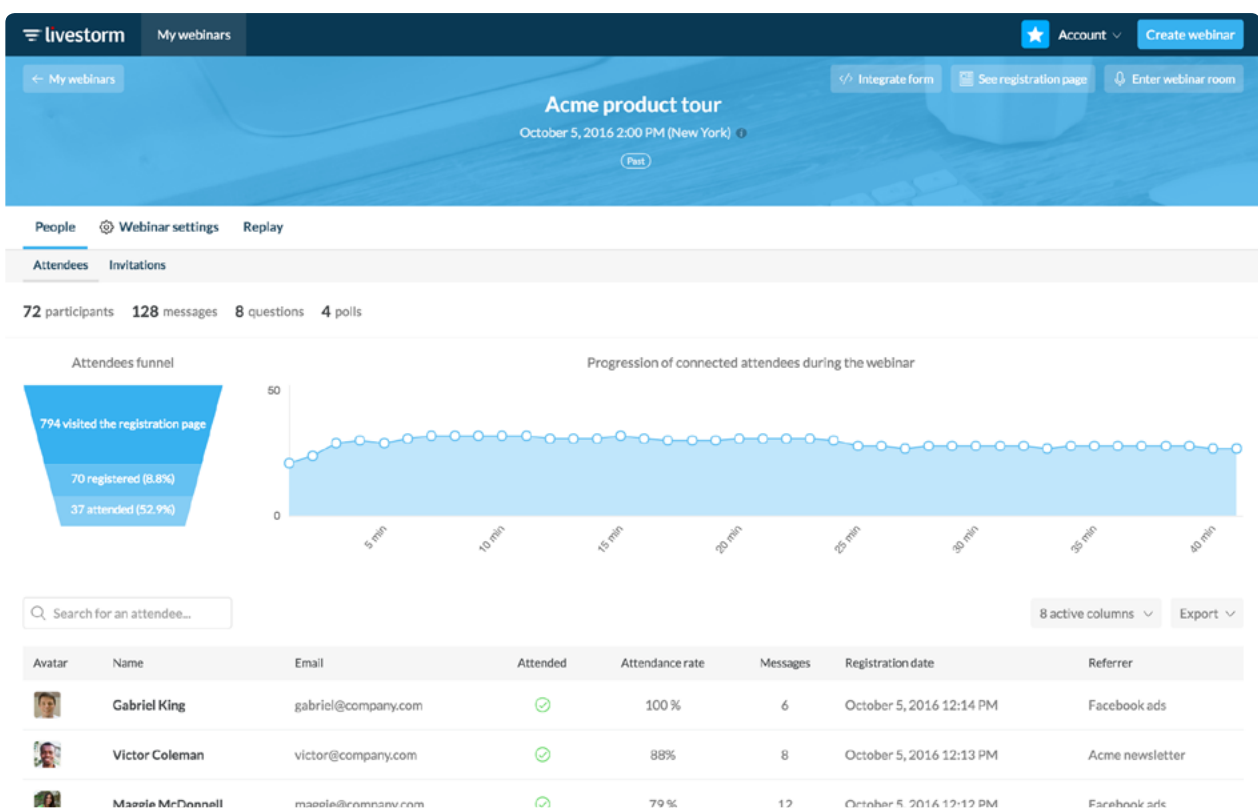
Most of developers still choose React and other popular frameworks, but Gilles believes there's a shift coming.

To hire experts for our staff, we attended Vue.js meetups in Paris, where we met highly experienced people. We also tried recruiting via job sites. Interestingly, most of the developers we spoke with said

that they use Vue.js for their own side projects but their jobs have them working on a daily basis with Angular, React, and other frameworks. Often in big, well-established companies, Gilles points out.

One thing I've noticed, however, that companies often use these technologies because they're forced to by legacy code or because they want to get a taste of this new hot tech everyone else is trying out. In the startup community, on multiple Slack channels and meet-ups I'm part of, CTOs and co-founders I spoke with who were interested in moving towards Vue.js were very excited with the fact that we've chosen Vue at Livestorm and have been asking a lot of questions about it. Frankly, I believe there's a significant shift coming—people will be more interested in moving towards something reliable and high performance, like Vue.js, while hyped technologies like React will gradually wane in popularity until they're finally phased out, Gilles adds thoughtfully.

As Gilles wanted to put his product out there as quickly as possible,



the team created a quick MVP to get some initial feedback from the outside world.

We spent less than a month building the first MVP. It was enough to present the product and the basic concept. In the end, we got a lot of positive feedback, which confirmed that we have a market fit, he recalls.

It took us 5 months to sell the first subscription. That's quite long, but we needed to first work through some stuff that was not necessarily technology-related.

The list of features that Gilles' team built into their platform to make it a competitive solution is really awe-inspiring.

WebRTC real-time streaming, full HD streaming during live sessions, webcam and screen sharing are the primary video-related features. We also offer an analytics-focused section that runs on Vue.js and integration with popular sales and marketing tools like Salesforce. We also developed a one-of-a-kind feature that no other browser-based webinar software has, which allows users to switch, on-the-fly, from WebRTC to HLS to make the stream compatible with Internet Explorer users and a range of mobile devices.

Outcome

After a year on the market, Livestorm has customers from all over the globe and a profitable product.

We have around 150 paying customers. They are all impressed with

how fast Livestorm is, they also love the UI and UX. Business-wise, we have an app that runs on its own, without our interference, so to speak—we don't have a sales team. We have a staff of seven, including product experts, engineers, and one marketing person. That would be me. But just because the product is that good and reliable, we grow by 20-30% each month, Gilles explains.

With Vue.js, Livestorm can release new features faster to the delight of their customers.

We try to ship new things as fast as possible, of course. Right now, we're two months into a phase that will conclude with the release of a big feature we're working on, but we usually ship features in a week or two, Gilles elaborates.

With Vue.js we don't have to reinvent the wheel every single time.



We can reuse all the components we already have to speed up the development. Currently, 39.5% of our codebase is built with Vue.

Gilles claims that choosing Vue.js over other frameworks made his company succeed faster.

Only benchmarks tell the truth and right now the benchmarks clearly demonstrate that Vue.js is definitely the choice for new and existing products. So if anyone has to make a technology choice in the near future, they should rely on concrete facts, figures, and benchmarks rather than opinions, he says.

If you have a lot of developers on staff who are used to Angular or the more classic frameworks, having them shift to React may be painful for the entire team. Transitioning to Vue, on the other hand, is much smoother, which in turn translates to lower costs. We didn't have to spend a month setting everything up like with React, Vue had us operational in a week. We would never be at the point we are now if it weren't for Vue. I'm 100% sure of that.

The Future of Vue.js



Evan You
Creator of Vue.js



Sustainability

As a project, Vue.js has come a long way to become what it is today. It has grown from a small experiment into a mature framework that is used by hundreds of thousands of developers all over the world. It has also grown from “yet another side project” into an ecosystem with over 300 contributors across the vuejs organization, and maintained by a core team of over 20 active members across the globe. Core team members have taken up responsibilities such as core library maintenance, documentation, community engagement, and major new features such as type declaration improvements and test utilities. Saying that Vue is a “one-man project” is no longer accurate and would be disrespectful to all the amazing contributions from the team and the community.

Financially, the Patreon campaign has been receiving stable recurring pledges since February 2016, which has allowed me to commit to full-time work on the project for over one and a half years now. In addition, the recently started OpenCollective campaign, which aims at providing financial support to community initiatives, has received over \$11,000 in annual budget in only two weeks and continues to grow. More im-

portantly, these open financial contribution channels mean that your company can actively help ensure the sustainability of the project by becoming a sponsor.

Today I am confident to say that as an open source project, Vue.js has surpassed the critical mass and the project's survival is no longer a concern for anyone considering adopting the project.



Stability

The frontend landscape moves fast and we understand how frustrating constant breaking changes can be. That is why we take stability very seriously. Looking at the project's history on GitHub, you will see a rock solid track record of consistent releases of new features and improvements, prompt bug fixes, and a meticulous standard for code quality (yes, we maintain 100% test coverage).

All Vue.js package releases adhere to semantic versioning and we try our best to communicate any potential required actions ahead of time. 1.0 was released in October of 2015 and didn't go through a single breaking change in the public API until 2.0 was released almost a year later. Before the release of 2.0, we conducted open design discussions and released multiple alpha/beta/RC releases to ensure the stability of the final release. We worked hard to retain API similarity with 1.0 and provided comprehensive guidance and tooling for the upgrade. Today, 2.0 has been out for over a year, used widely in production all over the world, and we don't see a need for major breaking API changes in the foreseeable future. It is our commitment to evolve the framework with minimal churn for the users.

▶ Continuous Evolution

Of course, we are not going to sit on what we have done up until today. We have many ideas in the roadmap that we plan to explore and implement in the coming years, and I will divide them into three categories:

○ **Near-term improvements**

These are new features/improvements that will be continuously delivered in 2.x minor releases. They can come from feature requests, inspirations from the broader web dev community, and use cases we have encountered ourselves in actual development.

○ **Mid-term improvements**

There are new JavaScript language features (e.g. ES2015 Proxy, Promises) that may simplify or improve the current API, but are currently not suitable for the main branch due to having to support IE9. We plan to start taking advantage of these features in a parallel branch which requires latest evergreen browsers.

○ **Long-term improvements**

We are also keeping an eye on emerging standards such as ES class syntax improvements (class fields and decorators), Web Components (custom elements & HTML modules) and Web Assembly. We have already started experimenting with some of them, and will definitely leverage them to further improve Vue's development experience and performance as they mature in terms of browser adoption.

Long-Term Vision

Many people have asked me why I started working on Vue.js. To be honest, the original goal was to “scratch my own itch,” to create a frontend library that I would enjoy using myself. As Vue got adopted by more and more users along this journey, I received many messages from users telling me that Vue has made their work much more enjoyable, so it seems my preference happens to align with that of many fellow Web developers. Today, I envision Vue’s goal to be helping more developers enjoy building apps on the Web. I believe that happier developers are more productive, and ultimately create more value for everyone. The goal entails delivering a framework that is approachable, intuitive, and at the same time solid, powerful, and scalable. I believe we are on the right track, but there is also a lot more we can do, especially with the Web platform evolving faster than ever before.

We are excited for what’s yet to come.

© Monterail, October 2017

Monterail is a close-knit team of 80+ experts offering Web & mobile development for startups and businesses. And we kinda love Vue.

www.monterail.com

hello@monterail.com

