THE SOFTWARE HOUSE

State of Frontend 2022

Table of content

Chapter 01 **Intro**

Is frontend tired of new trends, and looking for stability?

Aleksandra Dabrowska
report's Editor-in-Chief

The last two years haven't been the easiest and prompted a lot of changes globally. The IT industry was directly engaged in digital transformation since life as we knew it basically moved online. Frontend development was also subject to change – from technologies to good practices and working conditions.

It seems that the most famous frontend development joke "new day, new framework" has aged poorly. Sure, new frameworks and libraries do emerge but the race towards trendy innovations slowly gives way to maturity and stability in certain areas.

The goal for the State of Frontend is still the same – we want to see the real day-to-day perspective from frontend professionals of all levels and backgrounds.

Inside you'll find answers to which technologies we love to hate but still use, which practices are sacred and which are neglected, and what future people see for themselves and the frontend landscape.

We're excited to see a varied representation of frontend specialists, as our respondents come from 125 different countries! We hope that in the future we'll manage to expand our reach even further.

Report in numbers

3703
survey filled in

125
countries represented

19
frontend experts comments

On top of that, we've invited 18 contemporary frontend experts to share their thoughts and comment on survey results. Their insights are not only a fact-of-the-matter source of knowledge but will also provide you with food for thought on different frontend development topics. Shout out to each and every one of our commentators - the report wouldn't be so awesome without your knowledge and experience. Show them some appreciation, and meet them in the bio section below.

We encourage you to actively participate in the result analysis too! If we know anything about frontend folks it's that everybody has their opinions and rarely keeps them to themselves – which is great because it pushes the entire industry forward. Every diagram and table has a "share" button, in case you want to start a discussion with your friends or share only one specific data point of the report.

Finally, a big "thank you" to all 3703 frontend people who filled the survey – without you, there would be no report!

Chapter 02 **Developers & work conditions**

For software engineering, the biggest change over the past few years has been remote work

Gergely Orosz
The Pragmatic Engineer, author

A whopping 56% of respondents reported working remotely, and only 5% of them work in the office. The concept of mass remote work is so new that the survey in 2020 did not even measure this data point.

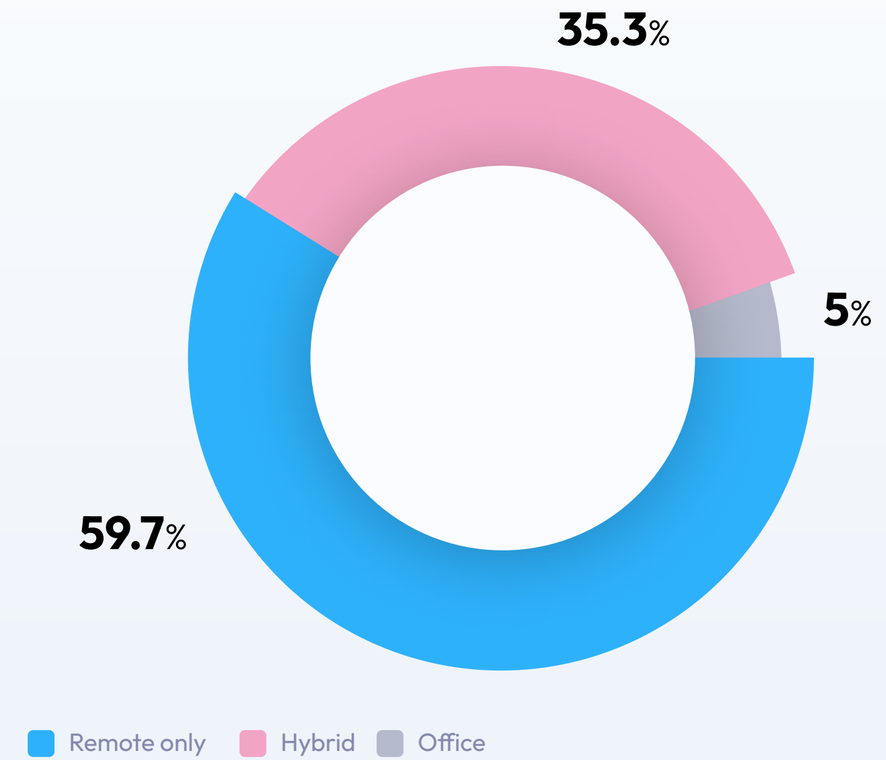
The big question for the year is whether full-remote work will be here to stay, or we'll see hybrid work gain more popularity. Most engineers clearly prefer working remotely – there's no commute involved, there are fewer distracting taps on the shoulder. However, it remains a challenge to share information and replicate spontaneous discussions that have existed in the office

It's not just frontend engineers who do frontend development

This year, some of the job titles people doing frontend development shared in the "other" option included:

- A bootcamp student just starting out learning frontend,
- A self-taught developer studying at a non-technical university who fell in love with frontend,
- A product manager who sometimes pushes code to production,
- Developer advocate who helps out the frontend team every now and then,

How does your work look like now?



- Frontend development architect,
- Design system director,
- A designer who also codes,
- Graphic designer & developer,
- Head of Everything: a one-person developer shop doing everything, including frontend development.

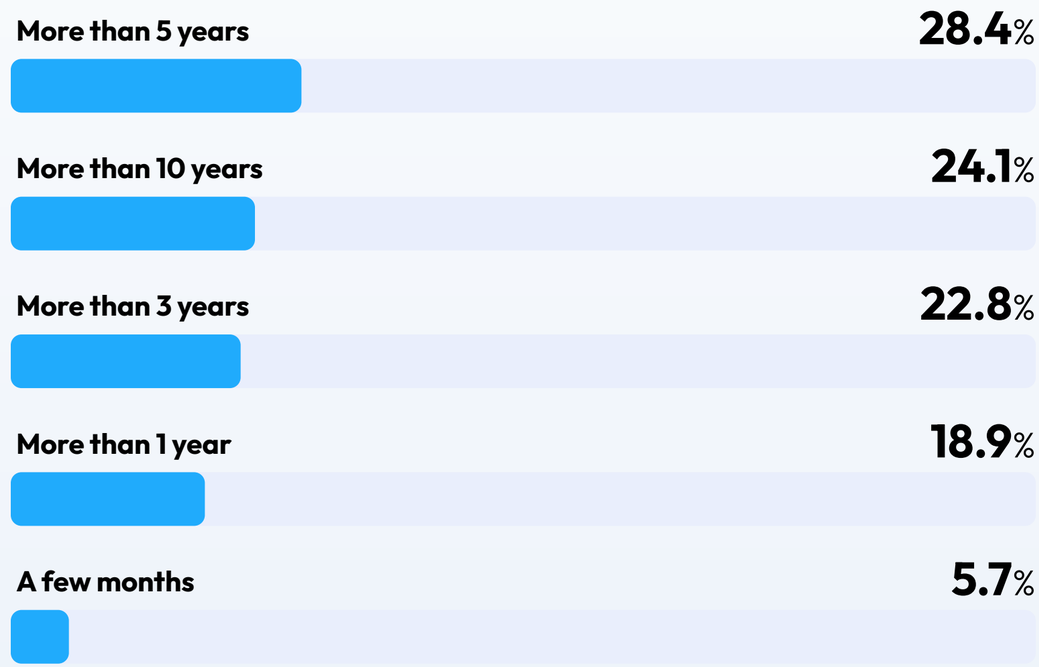
This should be of no surprise: but it's always a nice reminder of how frontend is an accessible area and one where it's still common enough to get involved without much frontend background.

Working within larger frontend teams is becoming more common

27% of respondents reported working at a company with more than 50 front-end engineers. At the same time, 30% of developers shared how 5 or fewer frontend developers work at their company. 50% of respondents work at companies with 10 or more frontend engineers.

This statistic shows an interesting duality. There are almost as many frontend engineers who work at companies with massive frontend teams as there are ones working on few-person teams or alone.

How long have you been in the frontend development game?



The developer experience and the expectations at these companies are vastly different. Large companies will have developer experience and frontend platform teams more often. Mentorship is more common. In smaller places, a lot more is down to each developer, and there are fewer options to get feedback. As a frontend engineer, I'd recommend, over time, working in both environments to maximize learning.

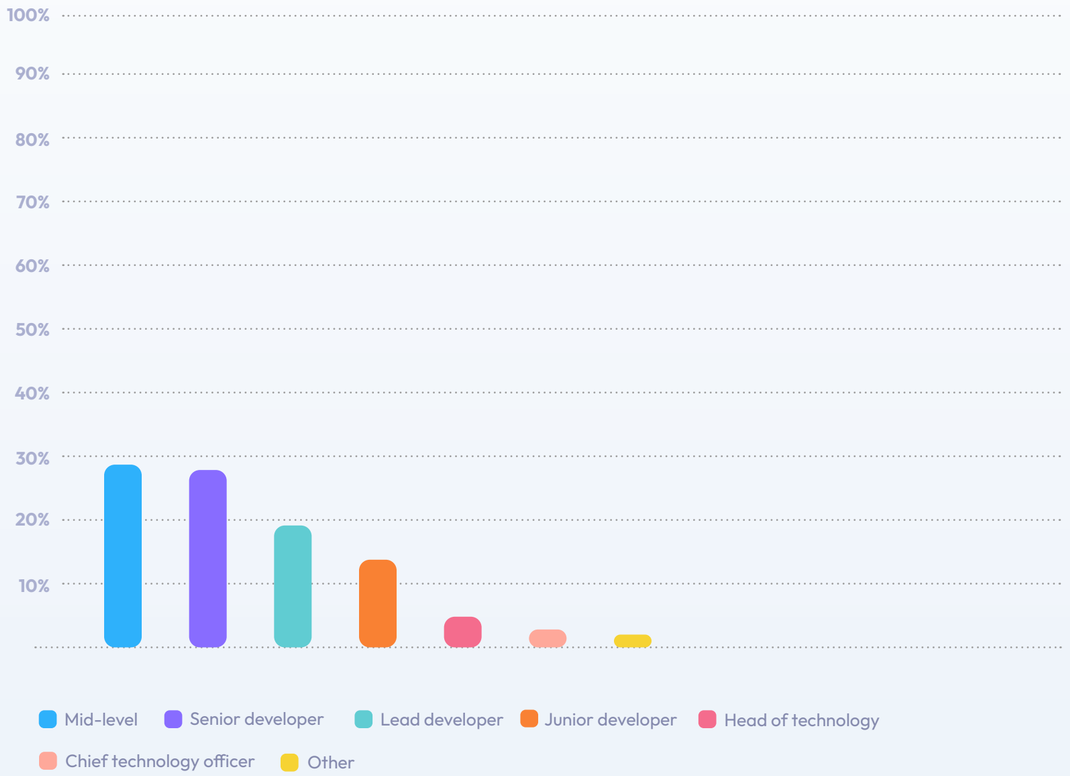
The statistic of 50% of frontend engineers working at companies with 10 or more frontend developers, and 27% of them at places with 50 or more, could also be an interesting prompt for teams building tools for larger frontend teams. There seem to be a growing number of these places.

Few engineers filling out the survey work at non-tech companies

Only 18% of people filling out the survey said they work at non-tech-first companies. 82% identified as working at a software development company, developer agency, or tech-first or digital-first companies. It's hard to tell if the survey didn't reach people who work at more traditional companies, or there really are more engineers working at places where software is core to the business.

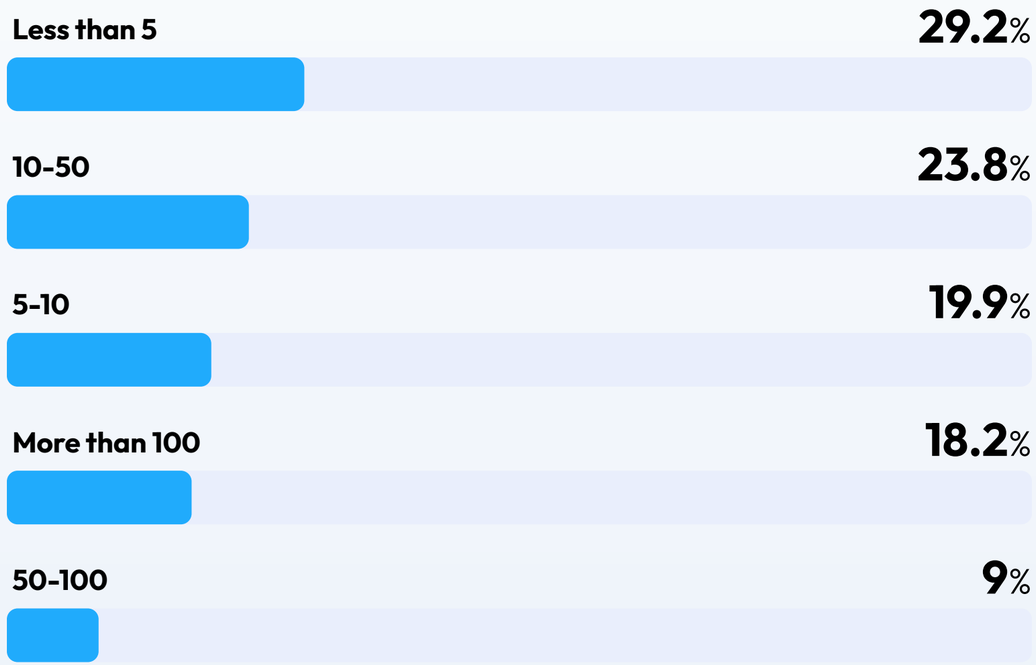
Either way, it's worth keeping in mind that the survey results come overwhelmingly from places where tech and software are more of a profit than a cost center.

How would you describe your current status?

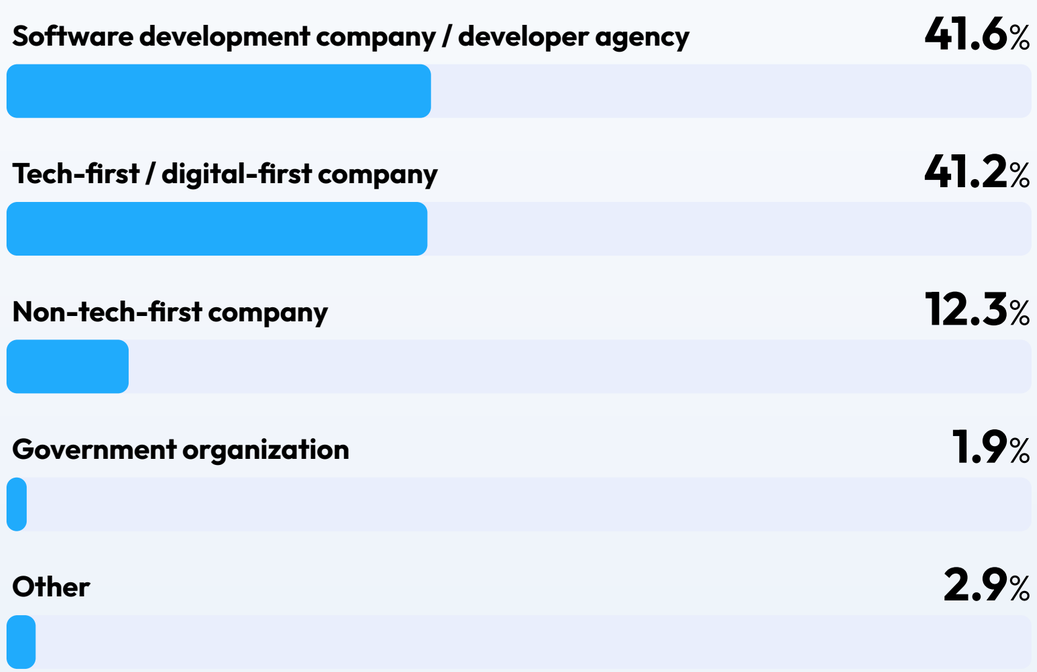


What's the size of the company you work in?

How many frontend developers are there at your company?

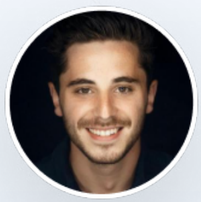


Which one of the following best describes your company?



Chapter 03 **Frameworks**

Developers choose frameworks with good practices in mind



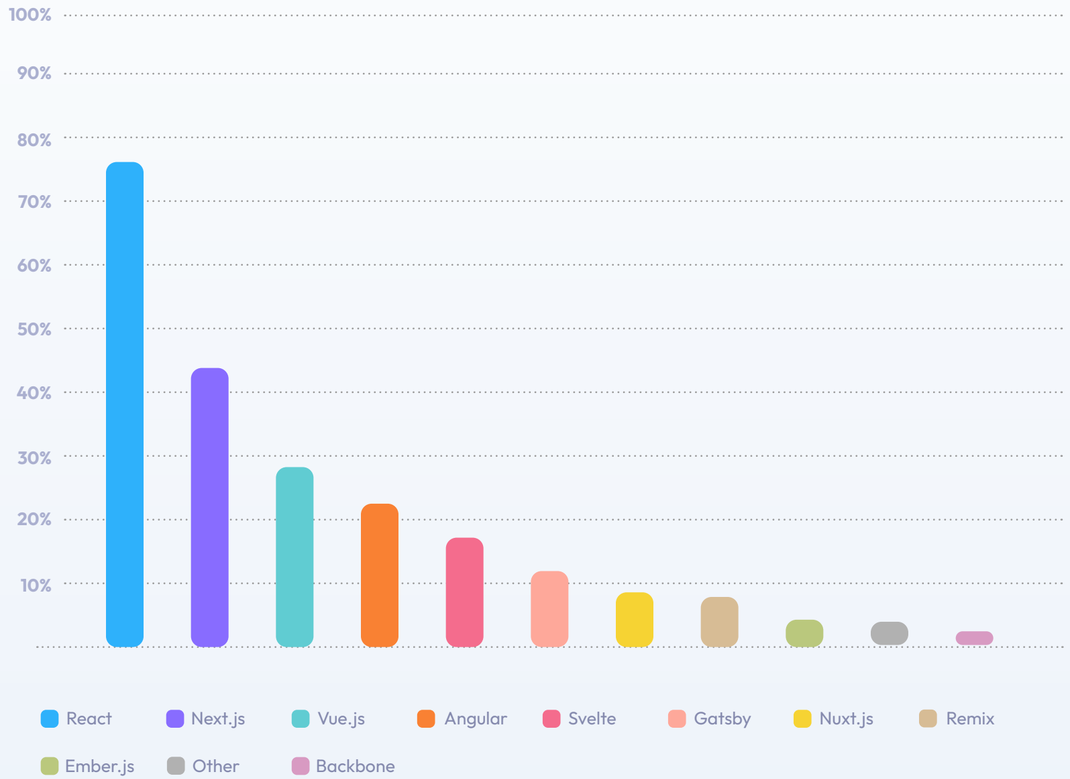
Sébastien Chopin
CEO at NuxtLabs & author of Nuxt

For me, the story of 2022 results is the rise of frameworks. It seems that developers are increasingly looking to meta frameworks to help them work faster and with greater confidence. The survey reveals that respondents are increasingly likely to be concerned with following best practices (e.g. performance and end-user experience) which completely explains this rising move toward meta frameworks.

Accessibility is a major focus for respondents this year, with 63% predicting it will gain in popularity over the coming years. Frameworks tend to provide different ways to solve this, with some notable examples including Next/Nuxt Image, HTML-validator, and WebHint. The Chrome Aurora team is working with meta frameworks such as Angular, Next, and Nuxt to make sure they implement these best practices. I predict we will likely see continued improvement from all these major frameworks in the upcoming years.

Component-driven development is also embraced by most developers, which makes sense given the popularity of React, Vue, and Svelte, and even web components (as in this year's indie success story - Wordle).

Over the past year, which of the following frameworks have you used and liked?



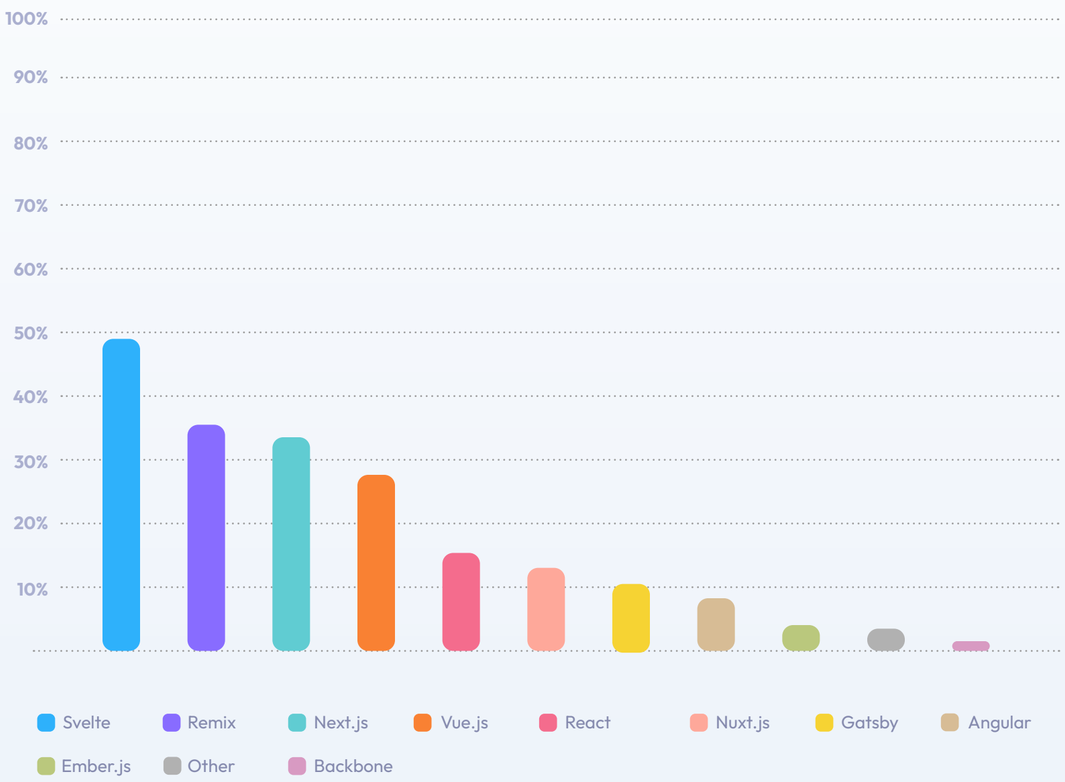
Progressive web applications are gaining popularity as well, with developers keen to make the most of cross-platform development using the same core codebase. We are also seeing groups like Open Web Advocacy push Apple to embrace the open web. This is definitely a space to follow.

Headless CMS is also advancing, with greater adoption and more integration into frameworks. Close to home for me, new Prismic, Strapi, Sanity, Storyblok, and Directus modules have already been released for Nuxt 3, working with zero configuration.

I also noticed another trend that is not mentioned explicitly in this survey. Edge rendering was initially driven by CloudFlare and its worker platform. It's no accident that most of the deployment targets on the survey have released or implemented their own serverless or edge functions, and this is being quickly adopted by users. Frameworks such as Nuxt 3, Remix, or Sveltekit are moving in this direction, enabling on-demand rendering directly at the CDN level. With the corresponding gains in decreased latency and lower cost for server-rendered applications, it's my prediction that this will be a big focus for 2023.

Over the past year, which of the following frameworks have you used and disliked?

Which of the following frameworks would you like to learn in the future?



Chapter 04 **Libraries**

Redux & Lodash – widely used, liked, and... disliked?

Andrzej Wysoczanski
Head of Frontend at The Sofware House

Chapter 04 **Libraries**

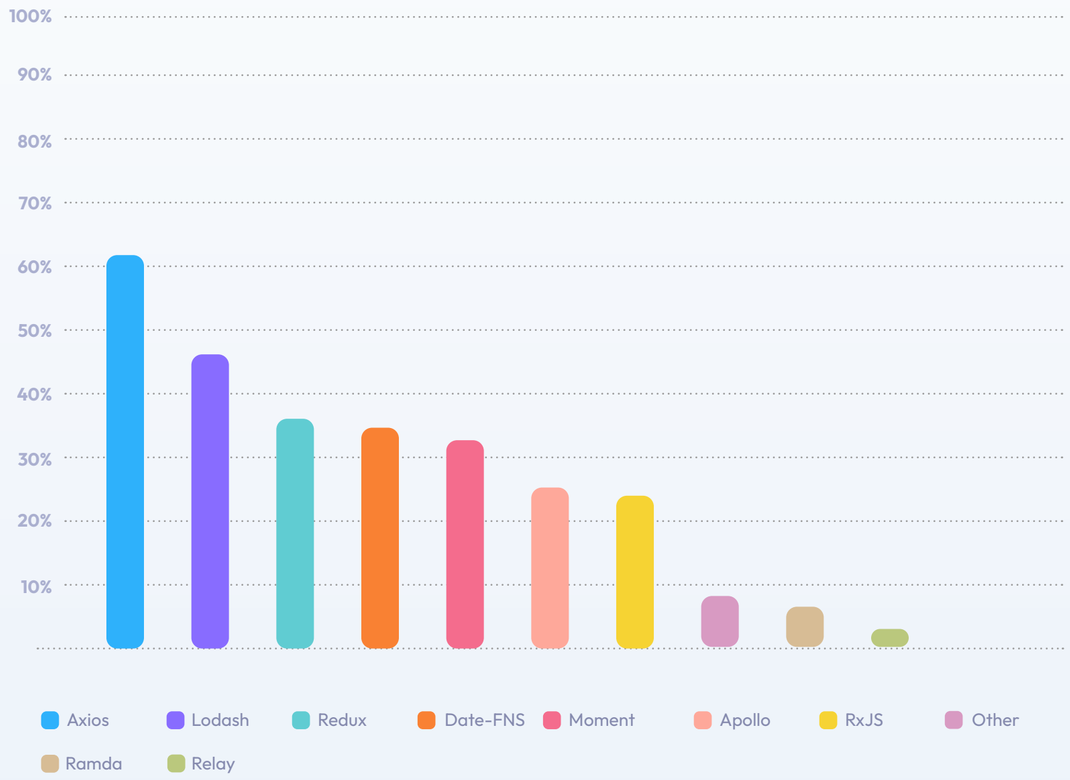
Whatever you think about Redux and Lodash, they are definitely used by frontend developers (willingly or unwillingly). Both landed at the TOP3 of liked and disliked solutions which makes me wonder, why would people use solutions they don't like. I have a couple of theories.

From my experience, Redux is widely used by software companies and their customers because it's great for large projects requiring complex state management. However, Redux has quite a difficult entry threshold. If a developer learns Redux from scratch, and it's something brand new to them, they may not initially like it. But, learn they must and learn they want, as almost 20% of respondents want to master Redux in the future, even though it's so difficult. Or maybe people realize that in order to score a nice job in frontend development, having Redux experience is good for their resumes.

As far as Lodash is concerned, the only logical explanation I have is that our respondents must have entered projects with these solutions in place, and they use them out of necessity, not fun.

It seems that frontend people move from Moment towards Date-FNS, and that's a good sign. I was shocked that over 40% still use Moment in their project, no matter what the sentiment. This library has already lost support, and even its official website has a note from creators stating that if you're considering using Moment, you should probably look for alternatives. Luckily, only 5% of respondents are eager to learn Moment in the future, so it's probable that this library lost its moment and is heading towards a decline.

Over the past year, which of the following libraries have you used and liked?

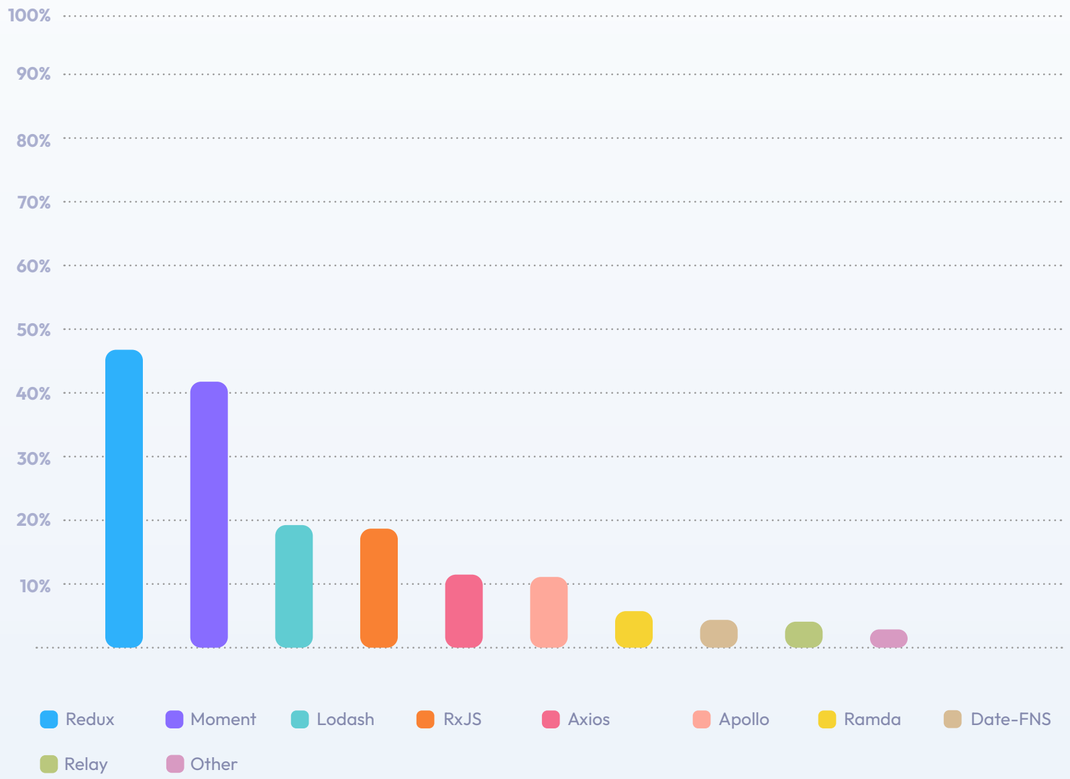


Axios, our “congeniality” prize winner with over 60% votes definitely entered the stability phase. It’s been on the frontend market for a good while, people know it well, and it’s more of a “standard” than a “trend”. No wonder, it offers decent data download, communication, and general cooperation with the backend. The question remains, the Axios naysayers would rather use GraphQL or they just honestly don’t like working with it?

Having mentioned GraphQL, I need to comment on two more solutions here. Since Apollo is used for seamless connection to GraphQL, I thought it will be much higher on the “used & liked” category. My hope was revived when I noticed that 40% of devs want to learn Apollo in the future (which saved it the first spot). That means Apollo’s community is steadily growing, and I expect more users of this library in the next report.

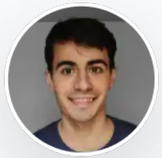
Apollo, with its easy-peasy configuration is the most famous one here, but maybe Relay can be its biggest competition soon. Relay is more complex and works only with React and React Native apps but 26% of frontend devs want to learn this library. If more people use Relay, the more projects implement it, and that can result in bigger engagement. I’ll keep my eye on GraphQL clients because I have a feeling that it will be the place where the frontend world can be surprised in the future.

Over the past year, which of the following libraries have you used and disliked?



Which the following libraries would you like to learn in the future?

Design systems with no clear winner

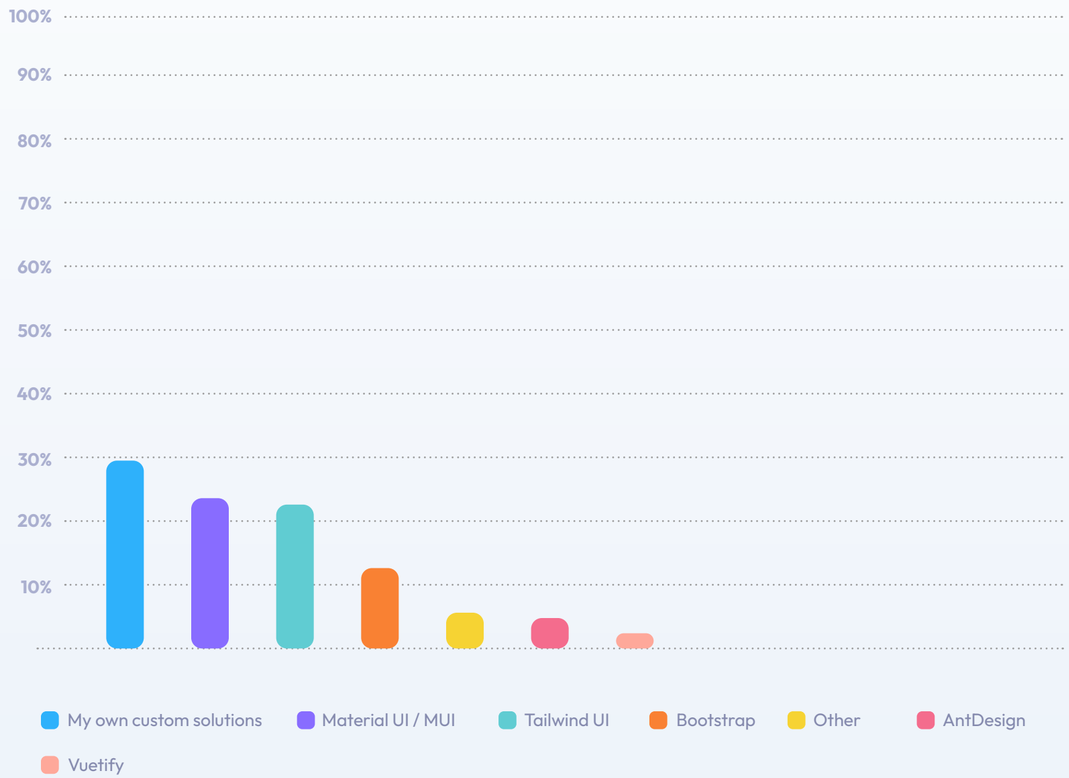


Olivier Tassinari
CEO at MUI and co-creator of Material-UI

The design system space is very fragmented. There is no single design system that goes beyond 24% of the market. This is a spark difference with React which has the large majority of the frontend market. **I think that this can simply be explained because the choice of the design system for a company is mostly an "artistic" one, and no two people have the exact same design tastes.**

As a side observation, there might be a possible bias in the results. The survey proposes "Material UI / MUI" as a predefined answer, I'm glad to see we are the leading option, however, for most people Material UI is synonymous with Material Design. So it's not clear if the respondents chose this answer from a design (design system) or code perspective (Material Design React UI library/framework).

Over the past year, which of the following design systems was your favorite go-to solution?



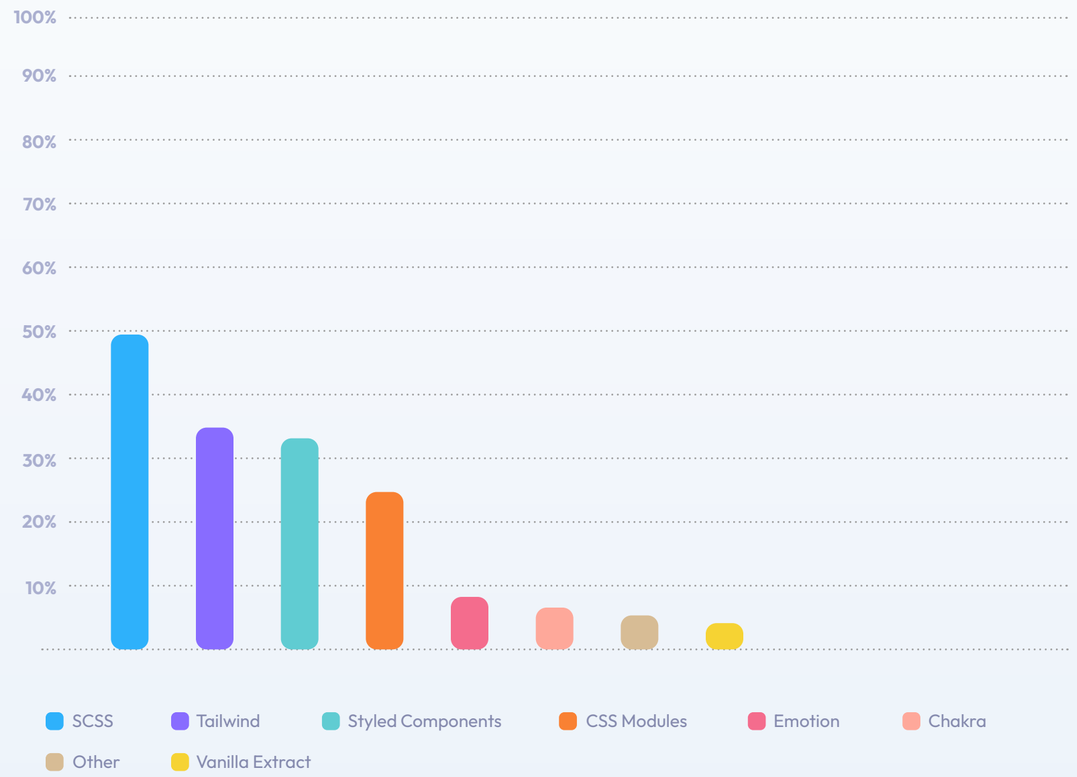
Styling tools. SCSS eats half of the chart pie

Chris Coyier
Founder of CSS-Tricks and CodePen

Wow, look at SCSS go! If a kid was born the day SASS was released, they'd be learning to drive today. That's incredible longevity for any software tool, especially in the fast-moving world of front-end development tools. **Having nearly half of respondents say they don't only use SASS but it's the favorite is incredible to me, and I happen to agree since it's a favorite of mine as well.** I think the syntax of it is quite nice, even though I tend to only use a handful of features like nesting and light mixin usage. Sass is, in a sense, up against CSS itself these days. I would guess variables are one of the top reasons developers reach for Sass, but Custom Properties have arrived in CSS and their support is ubiquitous, all but eliminating the need for Sass variables. Even nesting has momentum in CSS standards bodies, so we'll see if that one makes a dent in Sass usage as the years tick by.

Sass is a tricky one though, it doesn't mean that's all you're using. For example, PostCSS (only represented in the „Other" section here) is somewhat designed to be used in conjunction with Sass, at least optionally. Similarly with CSS Modules. While you can use CSS Modules alone, you can almost just as easily use it with Sass. That happens to be a favorite combination of mine, and it's not particularly esoteric as the wildly popular Next.js ships out of the box supporting this combo.

Over the past year, which of the following styling tools was your favorite go-to solution?



A monster showing for Styled Components as well! What strikes me about this is that the question is just about the usage of styling tools, but Styled Components all but implies the usage of React as well. So to see this big of a slice of the pie, especially combined with Emotion, Chakra, and Vanilla Extract, all of which I would guess primarily see usage in a React environment, shows off just how wildly dominant React was for participants of this survey. It makes me think of the other big JavaScript frameworks a bit. Where are the Vue people at? I don't see anything specifically called out in Other. They might just… not think about it? Styling is a built-in thing in Vue Single File Components land. You don't make a ton of styling tool decisions in Vue, as it's just there for you. And this brings me back to Sass a bit. Just as it's trivial to use Sass in Next.js, so too can you use easily use Sass in Vue, Svelte, or newer meta frameworks like Astro.

It's not terribly surprising to see that **regular ol' CSS is barely a blip**, knowing how heavily JavaScript framework usage is represented here. Once you're in a component-driven architecture, having CSS that is scoped to those components and offers additional utility through the availability of JavaScript, it makes sense that people take advantage of that, despite the thickening of the stack, as it were.

Tailwind's popularity also comes as no surprise here. If you asked me five years ago if I thought something like Tailwind would become popular, I would have said "no", but I would have been wrong. I've heard from countless developers that the idea of using HTML classes to style things just clicks for them. I have my own suspicions. If done well, the CSS produced by Tailwind is highly likely to be smaller (extra important for a blocking resource like CSS) which is a nice performance benefit from a tooling choice people seem to like anyway.

It's nice to see tools like **Vanilla Extract** trying to offer a modern variety of developer ergonomics in styling, and also be very focused on ensuring that good performance is the default behavior. Which generally means „extracting" „vanilla" CSS, if you follow their naming pun.

All this makes me think what the results would be if we could see data on, say, the styling choices of the top 5000 websites by traffic. Or the choices made on the last 5000 websites published on the internet. Or the top 5000 most actively developed websites on GitHub. Would it be similar? It's hard to say whether they would be completely different. **But I think of that staggering statistic following WordPress around: 43% of the internet.** It's not that you can't build a JavaScript framework-powered WordPress site, some people do, but I'm sure a tiny slice of all those WordPress websites. So what are they doing? Are they the big Sass users? Wouldn't you think a decent amount of them are vanilla CSS just because WordPress itself doesn't offer any built-in styling tools? Or maybe most of those sites aren't really built by developers, but just self-serve deployments?

It's certainly fascinating to watch styling tool choices change over the years. The only thing I'm quite certain of is that a few years from now, there will be surprises on this survey that would be impossible to guess today.

Development tools influenced by cloud

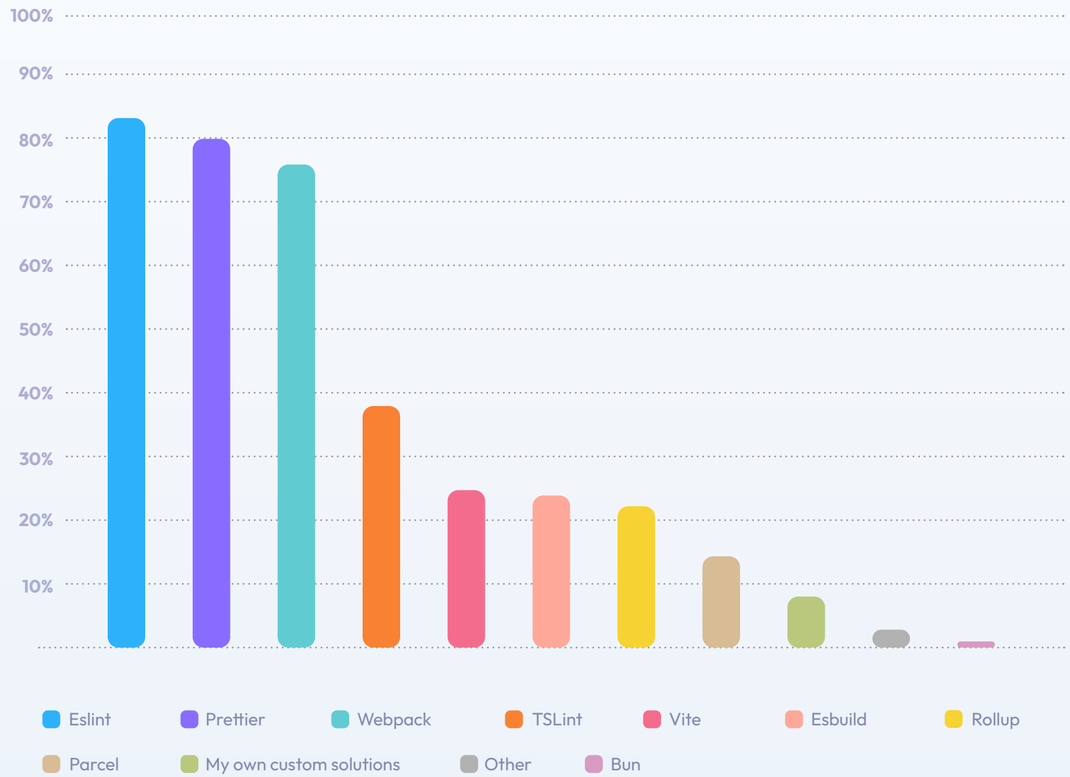
Ives van Hoorne
Co-Founder of CodeSandbox

It's great to see this topic being covered in the State of Frontend survey. You can see that more people are getting interested in using online code editors for some of their work, which is super exciting. Cloud development will only continue to grow, and I expect to see even more programmers and companies moving their development environment from the local to the cloud.

The survey confirms what we've already noticed at CodeSandbox. We've seen more and more people moving their development online, which also suggests improved general interest in cloud development. Over the past year alone, people have created over 12 million sandboxes, which makes for half of our total sandboxes ever created!

I'm very excited about the future because I believe that the cloud will make software development more accessible and collaborative. And I'm very happy to see that interest reflected in frontend developers' answers. As for my expectations for the future here, moving to cloud development may happen much sooner than we all think...

Over the past year, which of the following development tools have you used?



Chapter 05 **Typescript**

Typescript continues to make web development less frustrating

Marcin Gajda
Frontend Team Manager at The Software House

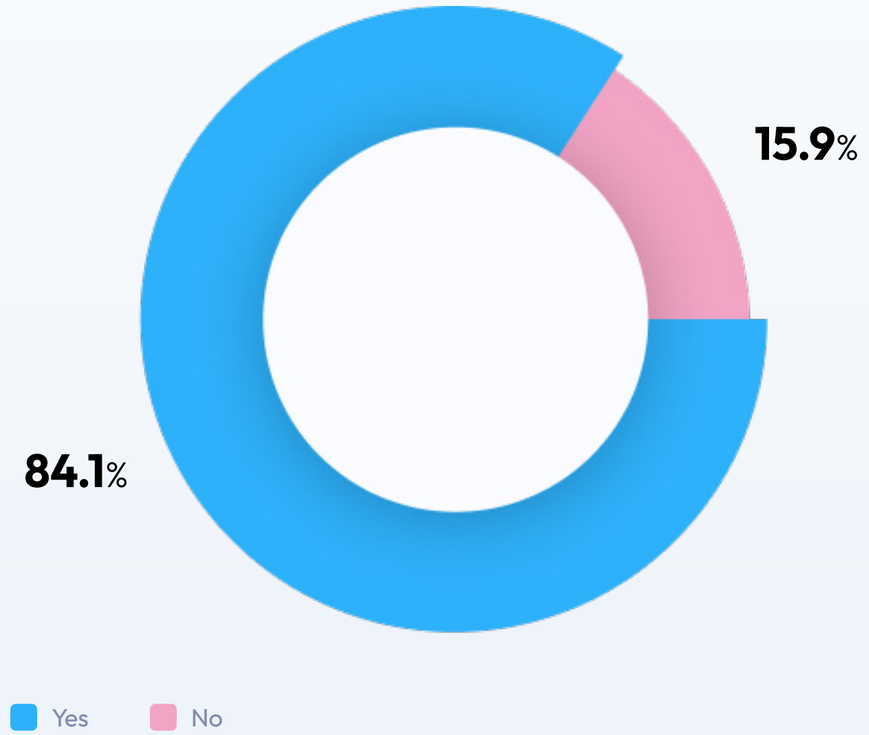
Chapter 05 **Typescript**

TypeScript doesn't intend to stop gaining more and more publicity with each passing year. You can especially see it, if you compare 2022 answers with those from two years ago. The number of people using TypeScript raised over 7 percentage points, already being at 84%!

We all can probably agree that TypeScript is universally embraced by developers, and the industry won't let go of this technology in the upcoming years. How did it happen though? In online discussions, people often praise TypeScript for how it prevents a whole class of bugs before they even happen. That in turn, makes development faster and apps more reliable.

I don't intend to argue here but since you asked me what really makes so many developers love TypeScript, I'm going to say that **TS made web development way less frustrating than it was before.** After far too many years of web development feeling laborious and painful, frontend developers don't want to re-live the experience of switching between the code editor and the browser back and forth multiple times to guess why "undefined is not a function". These "snow is white" kinds of errors were mostly caused by misspelled variables or misplaced parameters.

Over last year, have you used Typescript?



Chapter 05 **Typescript**

Then TypeScript came to our rescue, being baked by Microsoft and armed with the support in all major IDE. Writing code on frontend feels way more controlled and straightforward now. Personally, I also enjoy the extra layer of fun added to the whole development process by the possibility to design data structure shape before writing the code that utilizes it.

TypeScript not only attempted to win over developers' hearts but also fought its way to become the frontend industry standard, not only for Angular projects. It's safe to say that new commercial projects NOT using TypeScript at all have already been scarce, and it will only be harder to find them in upcoming years.

If you compare the questions about using TypeScript with company type, it's clear that the tech industry confidently moves towards Typescript in their software projects.

To support my claim further, people who didn't touch TypeScript over the past year work more often in non-tech companies or government organizations. Nothing surprising, because these organization types are infamous for being set in their ways and sticking to older solutions. This in turn often infuriates frontend developers, who don't enjoy working with obsolete technologies. The results: ~13% vs ~20% for the more tech-related and dynamic competition

Company type	TypeScript	
	Not using TypeScript	Using TypeScript
Not provided	**32.67**%	**67.33**%
Government organization	**19.35**%	**80.65**%
Non-tech-first company	**19.21**%	**80.79**%
Software development company / developer agency	**12.93**%	**87.07**%
Tech-first / digital-first company	**12.16**%	**87.84**%

Future of Typescript

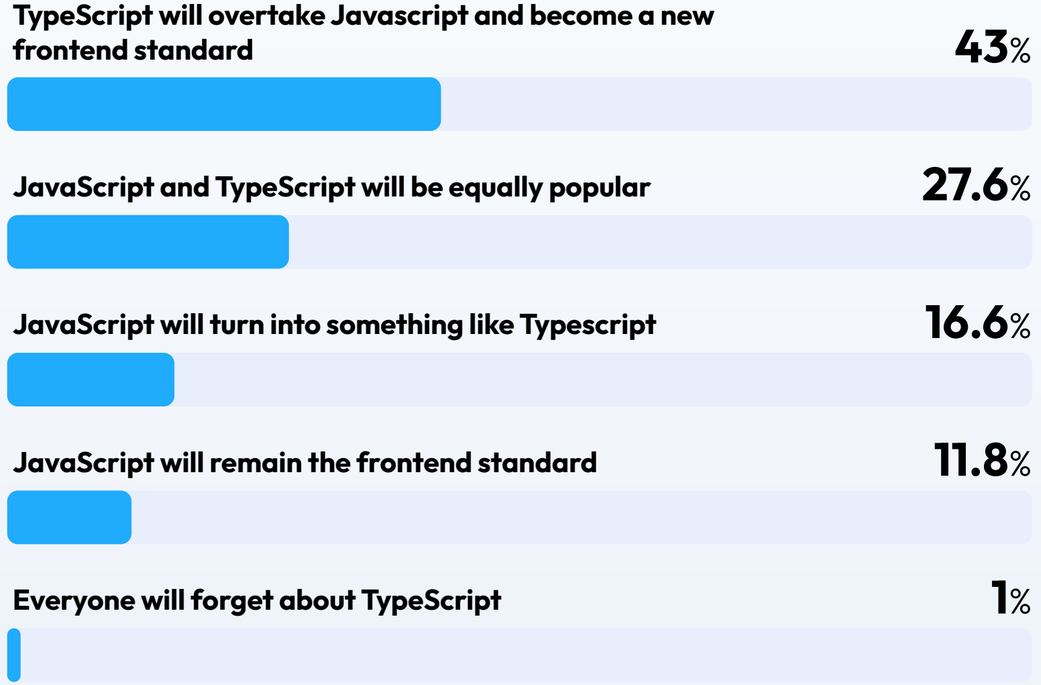
As far as the future of TypeScript is concerned, predictions changed quite a bit. In 2020 it was a close call between the top three options. The respondents didn't give a definitive answer about what will happen between TypeScript and JavaScript. **My bet is that two years ago we still weren't exactly sure if TypeScript is just a temporary fad or something that will stay with us for longer.** Considering how much constantly happens in the world of frontend, and how often new solutions emerge, a bit of caution is perfectly understandable.

The State of Frontend 2022, brought a clearer answer. **People who think that TypeScript will become the primary solution for web development are in the great majority with 43%.** I know, the result is still not over 50% yet, but if you were playing „Who Wants to Be a Millionaire?" and that was your result in the "Ask the audience" question, wouldn't you bet on it? I think the main reason behind the shift becomes clear when we look at new, emerging solutions. There's a noticeable increase of libraries written in TypeScript natively, and most of the new development tools come with out-of-the-box TypeScript support.

Last but not least, we could observe in real-time how the "JavaScript will turn into something like TypeScript" question slowly comes to life. In March 2022 it became surprisingly more realistic than ever when Microsoft announced their proposal to introduce the type syntax from TypeScript in JavaScript.

It means the browsers will understand TS but won't support type checking. For now, the frontend community gave the proposal a cold reception and I don't think it has a chance to be accepted into the ECMA standard in the current form. It also doesn't mean that JS will morph into a TS clone, but definitely, something is in the air.

In your opinion, which of the following future of Typescript scenarios seems most likely to happen?



Chapter 06 **Static-Site generators**

SSG solutions are on the rise

Samuel Snopko
Head of DevRel at Storyblok

Chapter 06 **Static-Site generators**

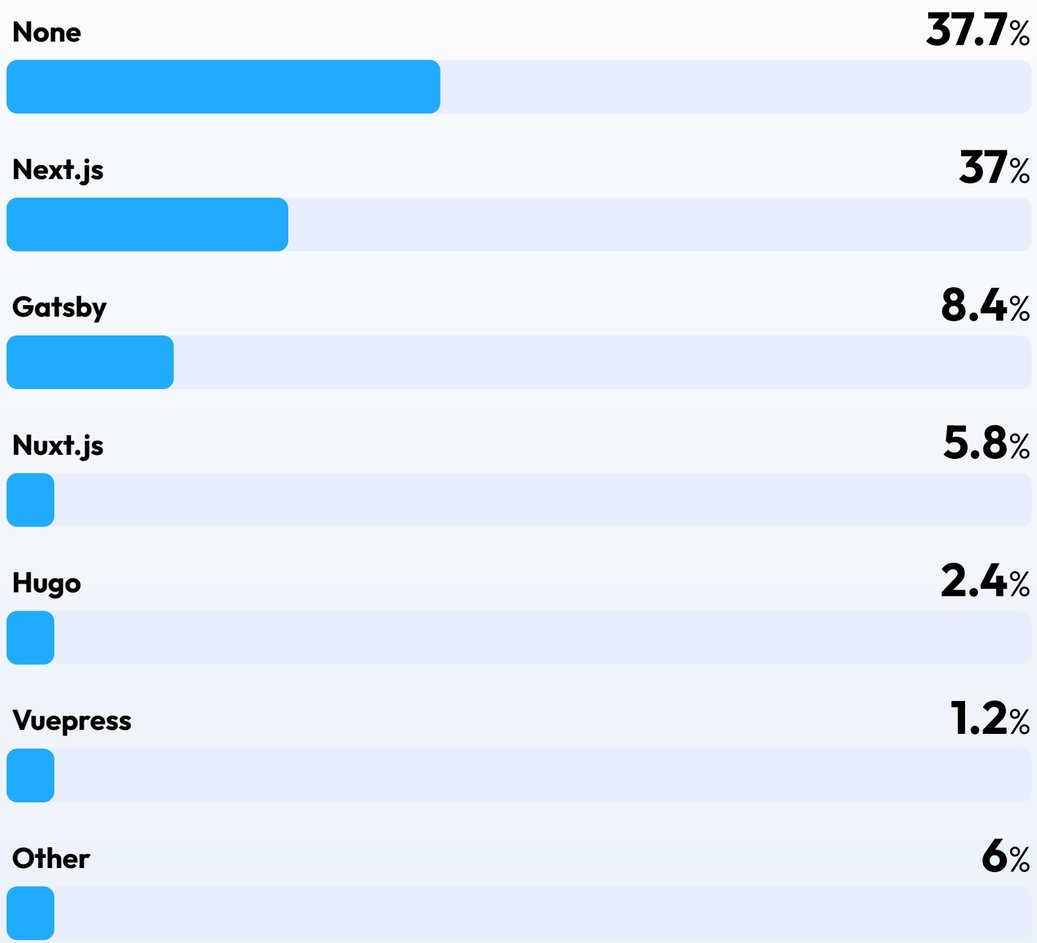
Increasingly, huge companies are not afraid to switch to headless CMSs with SSG - Jamstack solutions are no longer a new cutting-edge technology, and they don't seem experimental to them anymore.

This change is leading to a rise of SSGs with unbeatable performance and caching, and we can already see many new frameworks like Remix, SvelteKit, and Astro, who want to grab their piece of the market cake. **I think this new competition will lead to some exciting surprises in the following months, and the leading trinity of Next, Gatsby, and Nuxt will need to evolve even faster!**

The most important feature will be the incremental generation, which will soon become a must for every framework. This makes faster and easier - without the need to regenerate the whole website, but only the portion that needs to be up-dated. Additionally, I expect a massive jump in localization and personalization strategies, which will become internal parts of the frameworks.

Over the past year, which of the following static-site generators have you used?

Which of the following static-site generators is your favorite to work with?



Chapter 07 **Hosting**

Deployment. Will mass migration to the cloud mean the end of traditional hosting?

Gift Egwuenu
Developer Advocate at Cloudflare

Chapter 07 **Hosting**

The first thing that I've noticed is that more folks are moving away from the traditional hosting on their own servers, as the result dropped by 8% points compared to 2020 answers.

Personally, I think this was always bound to happen and I don't think it's a bad thing that we are moving away from traditional hosting. Developers are looking to optimize their time and productivity and if they can find a way to take out most of the work required for initial setup, they will adopt those services. And that's what I see happening here, I think in a long run more people will move away from it but will it ever stop existing? No, I don't think so, some systems still require very custom hosting that they may not get from a provider so they choose to make their own. The migration is something that will continue to happen though as cloud hosting evolves.

The alternative, moving frontend hosting towards cloud providers, received a combined result of 64%! Amazon Web Services still remain top of the list with 45% responses, which is unsurprising considering AWS is one of the biggest cloud providers on the market.

GCP and Azure take the back seat in this year's results, both falling behind AWS and landing around 13% of votes each. Amazon must be doing something differently, and I genuinely wonder, would the results be different if Azure pushed Azure Static Web Apps more?

It's also quite interesting for me to see increased adoption of services like Vercel and Netlify. Over the years, these companies have proved to be on top of their game by offering cutting-edge services and including a free tier for developers. In turn, that creates a low entry barrier for anyone willing to learn and use their services to host their projects.

Where do you deploy your applications to the most often?

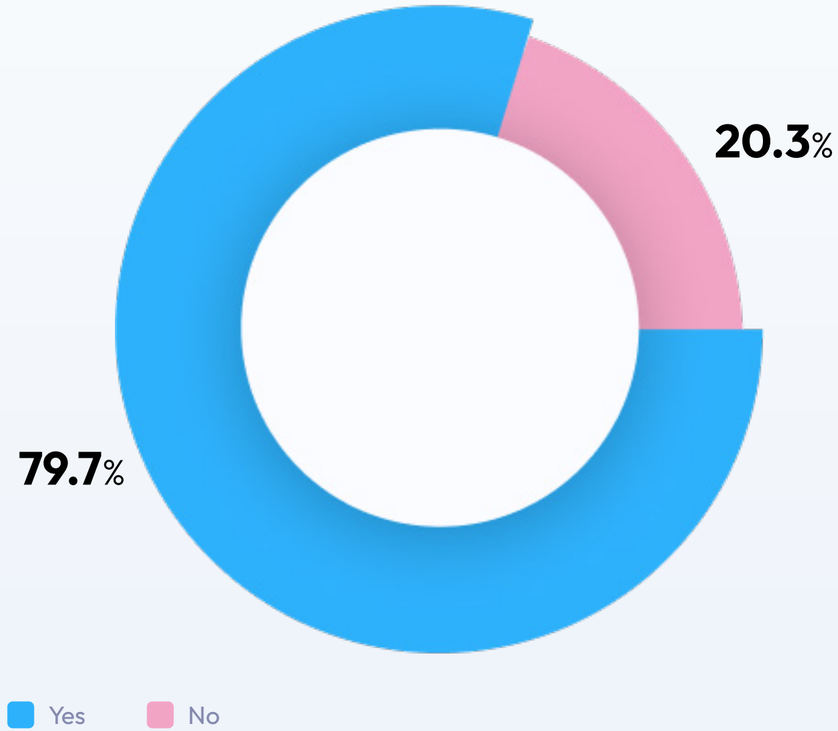
I also think that Cloudflare Pages should be proud of themselves. The solution is relatively new to the survey, yet nearly 4% of respondents chose CP as their preferred hosting option. What's more, even Cloudflare Workers made a frequent appearance in the "other" section. This only means that frontend developers are open to experimenting and adopting new services for deploying serverless applications.

CI/CD. Frontend connecting all stages of the Software Development Lifecycle

A majority of frontend folks (80% of respondents answered "yes") add Continous Integration to their workflow. **I believe this is great news, and it shows that people tie up all stages of SDLC (Software Development Lifecycle) into their workflows.**

As far as individual solutions are concerned, GitHub Actions takes the front seat in this survey, with a result of over 56% in 2022 compared to 35% in 2020. This shows that more frontend people shifted to GitHub Actions in their day-to-day. Maybe it's because GitHub pushed for Actions being the go-to option when you think of CI. The influence of being affiliated with Microsoft could be a reason why it received more love over the years.

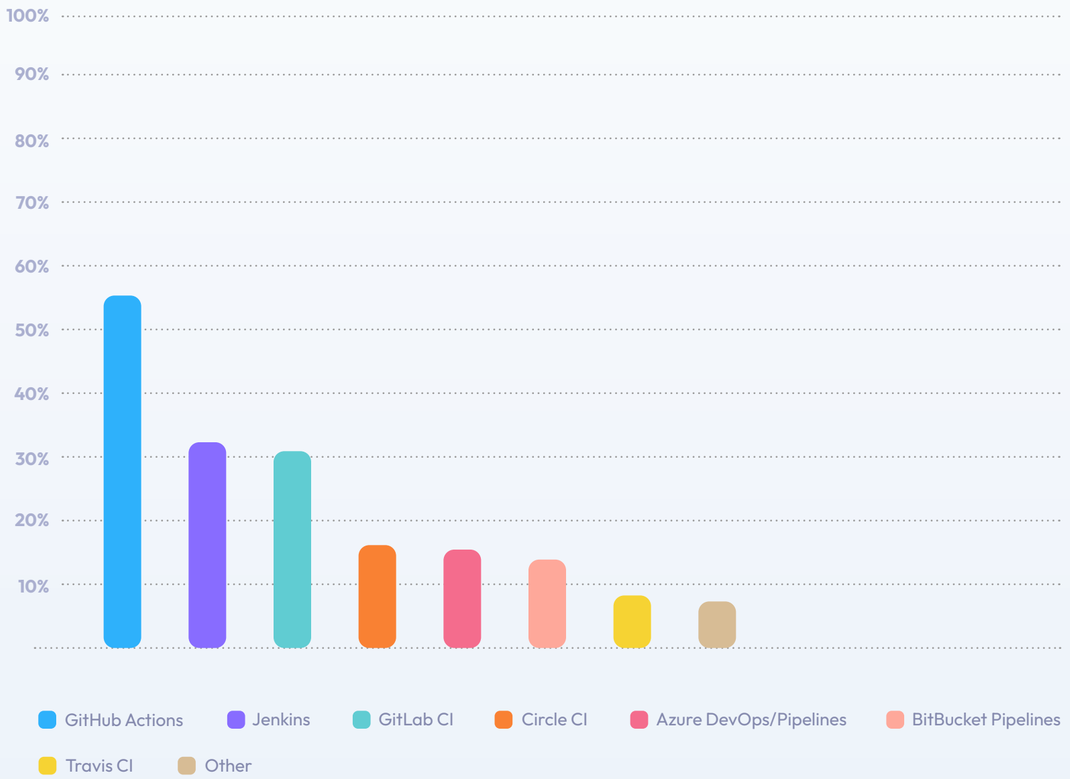
Do you use Continuous Integration?



From my personal experience, these results seem true indeed. I used Circle CI and Travis CI in the past, but now I default to GitHub Actions when I need to set up Continuous Integration.

We could also see more solutions (not included in set answers in the survey) popping up in “other” options. I’m talking about services like Teamcity, Click Deploy, Envoyer, etc. being the preferred options for Continous Integration. To me, this means that there are some niche providers that you may not have heard of but they still must be stable and dependable because developers do pick them as a go-to choice for CI.

Which of these Continuous Integration solutions have you used over the last year?



Chapter 08 **Micro Frontends**

Micro Frontends are on their way to maturity

Luca Mezzalira
AWS Principal Solutions Architect

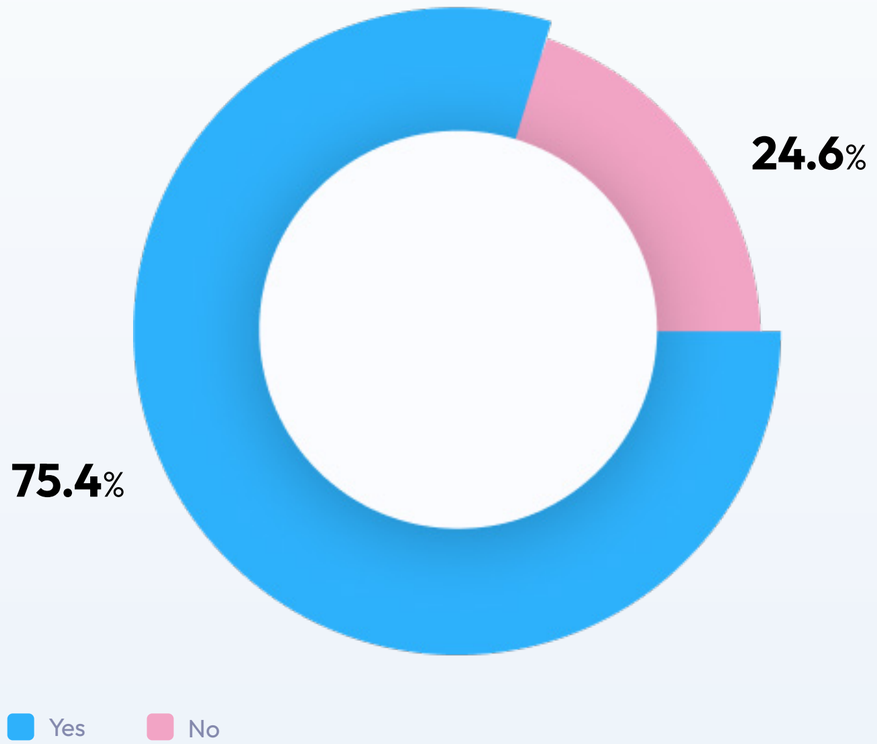
Chapter 08 **Micro Frontends**

Micro Frontends are embraced by a variety of companies nowadays. Among others, Netflix, PayPal, and Amex have implemented this architecture approach in some of their systems. **I'm convinced that this is the right path for micro-frontends maturity. Large corporations embracing this architecture will only provide a faster feedback loop for the community highlighting best practices and anti-patterns.**

Moreover, the industry discusses micro-frontends more and more. Nearly every frontend conference I've seen has at least one speaker, panel, or case study presentation on this very topic.

The community started to have more mature tools like Single-SPA or Module Federation for client-side rendering applications but we are still finding "the way" on the server-side rendering.

Over the past year, have you used micro frontends?



There is still a lot to do and discover. For instance how to deploy micro-frontends in production using a canary release or blue-green deployment? Or how to leverage partial hydration when using server-side rendering frameworks like Preact or React 18?

Having said that, micro-frontends have definitely moved forward in comparison to two years ago, and the aforementioned results prove it clearly. I think in the next few years, even more organizations will embrace this approach and new tools and patterns will be shared with and created by the frontend community.
I am excited to see what the future holds for micro-frontends.

What solution do you use for micro frontends the most often?

Chapter 09 **Browser technologies**

What’s up with 42% to WebSockets? I’d expected it to hit less than 5%

Jay Phelps
Web Platform at Netflix

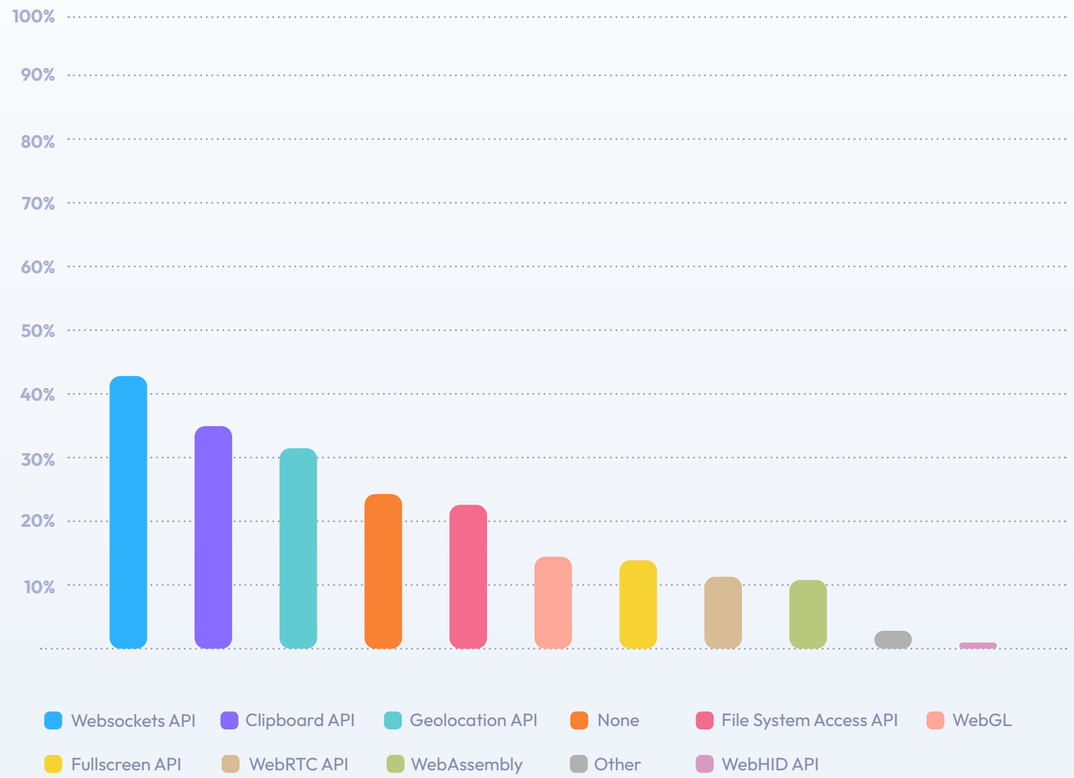
Historically, I (and others around me) have only needed these APIs for more advanced, native app-like experiences. **So the results are quite surprising, especially 42% of respondents who have used WebSockets, whereas I would have guesstimated less than 5% would actually have had a need.** I wonder what the primary motivators behind picking WebAssembly were: performance, possibility to use other languages besides JavaScript, lack of better options?

I have a few theories. The simplest explanation is that some sort of sampling error still does exist, or that the interpretation of the question by myself and the respondents is not necessarily inline. Had the developer used the given technology business-wise, on the production website, or they had simply experimented with them on private coding „out of work". That would help make up the difference between my expectation and the results.

However, I think the real reason is a combination of factors, including browser technologies being in fact used more often than ever before. WebSockets are used even in cases where real-time isn't necessarily required, with Firebase-like platforms as popular as ever. The relative usage order of various technologies also seems plausible.

The File System Access API is still pretty new (e.g. not yet supported by Firefox) so I'd be curious how many sites using it are still falling back to the good-old **`<input type="file">`.**

Over the past year, which of the following browser technologies have you used?



Chapter 09 **Browser technologies**

I have a personal admiration for WebAssembly. It's still young, and some improvements are necessary (especially client-side in the browser) but WebAssembly is the first truly standardized bytecode. That's an attractive feature for lots of use cases, not only within the browser but rather server-side or offline apps. Since WebAssembly is a compilation target, virtualized machine code, it is not intended to be written manually in the same way that x64 or ARM. That means most developers are compiling to WebAssembly from some other higher-level language.
I'd be curious to know the popularity of such languages these days. I expect the first three to be C/C++, Rust, and AssemblyScript, with an honorable mention for Golang since its popularity in the WebAssembly community has taken off too.

In the long run, tooling should make WebAssembly an implementation detail that most developers don't really need to care about. Much like iOS developers don't often care that they're compiling to ARM. But standardization and community growth are slow processes, so I think we're still more than a decade away from that reality.

Chapter 10 **Code management**

Browser editors are on the rise. Is this just an effect of remote work?

Santosh Yadav
GDE for Angular, GitHub Star, Auth0 Ambassador

Desktop code editors

Visual Studio Code has been a desktop code editor leader when it comes to frontend development, the team has been doing lots of improvements to make it faster and work on cross-platform. The ability to use VS Code online with GitHub has disturbed the online editor war too, if you are not aware you can press "." in GitHub and it will launch VS code online for you. No one thought it would enter this market too, post launching codespaces.

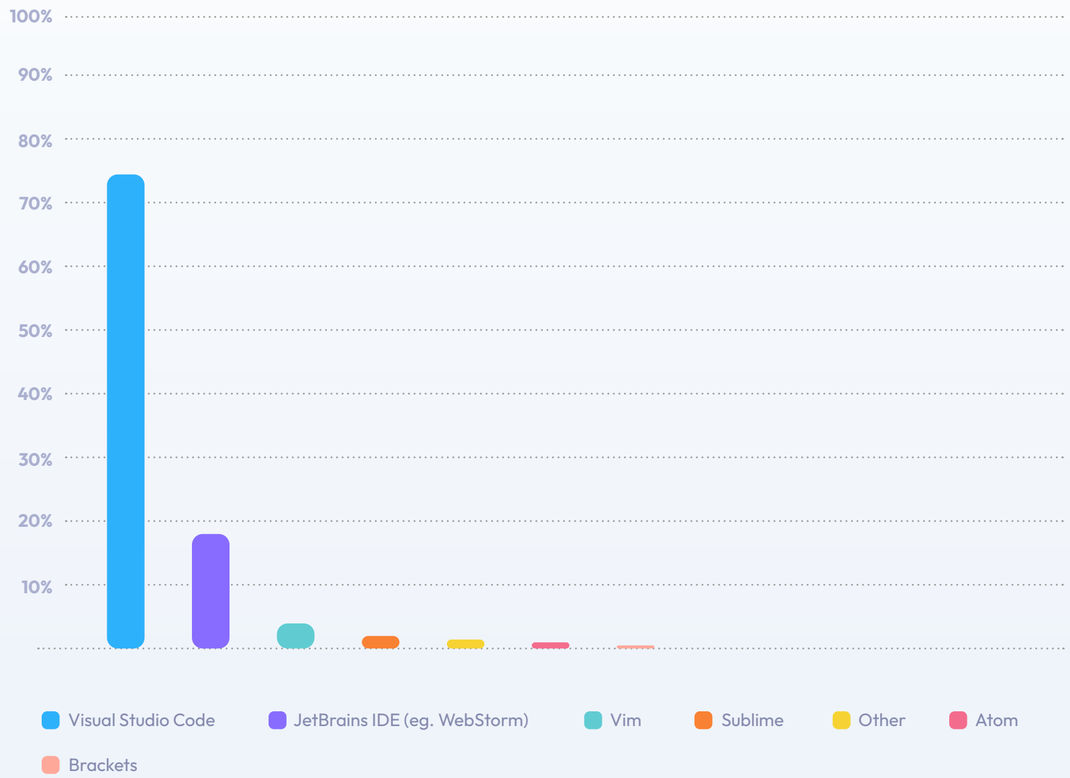
It will take some serious efforts to take the crown away from VS Code when it comes to the desktop editor. Developers have been creating some amazing extensions for VS Code that give a clear advantage as compared to other Code Editors like WebStorm.

Online code editors

For online code editors, I am seriously amazed by what StackBlitz has been doing. Especially introducing the web containers, so one can run NodeJs in a browser is amazing! CodeSandbox has been there for years as one of the leaders, but I can see serious competition from Stackblitz. You can do a lot of stuff using Stackblitz after web containers, notably running your npm scripts online. I love the deploy options available on CodeSandbox – you can deploy on Netlify or Vercel with a click of a button which is cool.

Online Code editors' use is only going to rise from here I think. **Many companies are going fully remote now and online editors are a great option to reduce the costs.** You don't need to invest in high-end laptops - CodeSandbox or StackBlitz can do it for you. Every developer knows how painful it is to set up the local dev environment, and online code editors can do it in a few minutes.

What's your favorite desktop code editor?



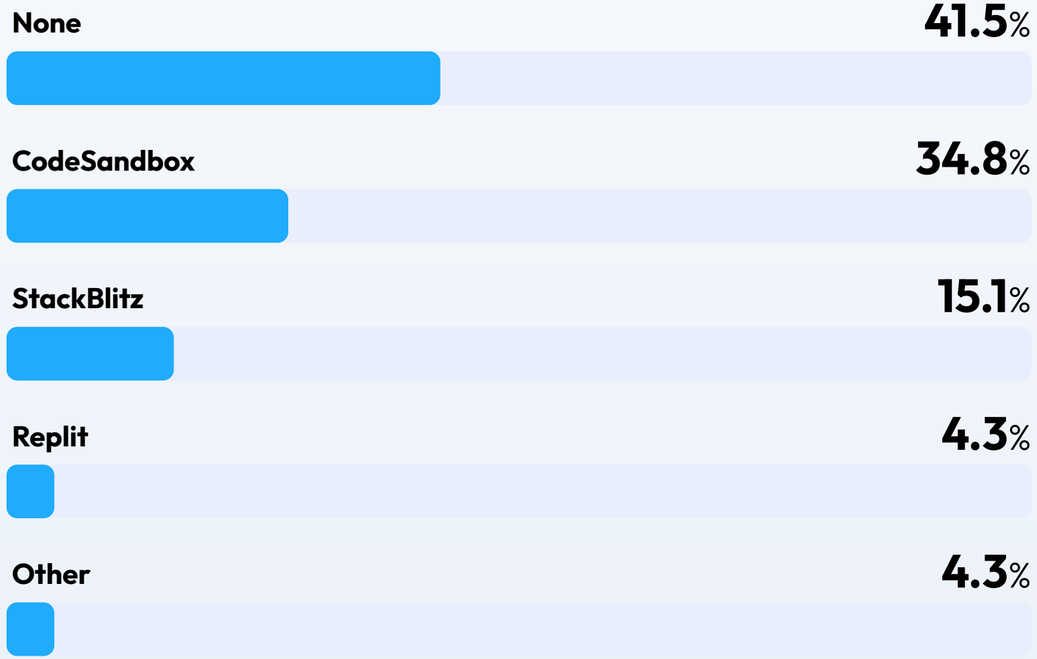
Chapter 10 **Code managment**

Version control providers

For version control, GitHub is the clear choice for many developers, and no wonder – the variety of features GitHub has introduced over the years has been astonishing: GitHub Action, CodeSpaces, VS Code Online, the new GitHub code search, co-pilot AI... I can go on about how all these features make developers' day-to-day life easier. GitHub Actions removed the dependency on external providers for Open Source developers and they get the free builds for Open Source work.

Gitlab and Bitbucket offer the advantage of a self-hosting option, which many enterprises desperately need. But nonetheless, GitHub is home to Open Source Developers, and it will only grow and the State of Octoverse is a clear indicator of that.

What's your favorite browser code editor



What's your favorite version control provider?

Chapter 11 **Testing**

Positive surprises in the state of testing: frontend developers have never tested more!

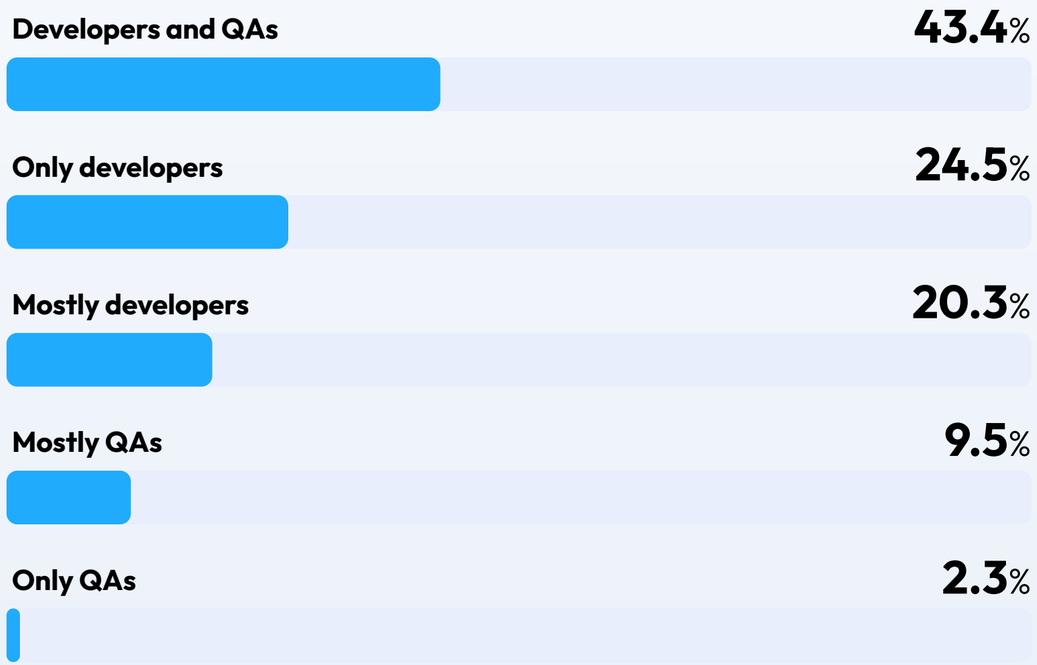
Dawid Dylowicz
Staff Test Engineering Lead at Onfido & curator at Software Testing Weekly

Chapter 11 **Testing**

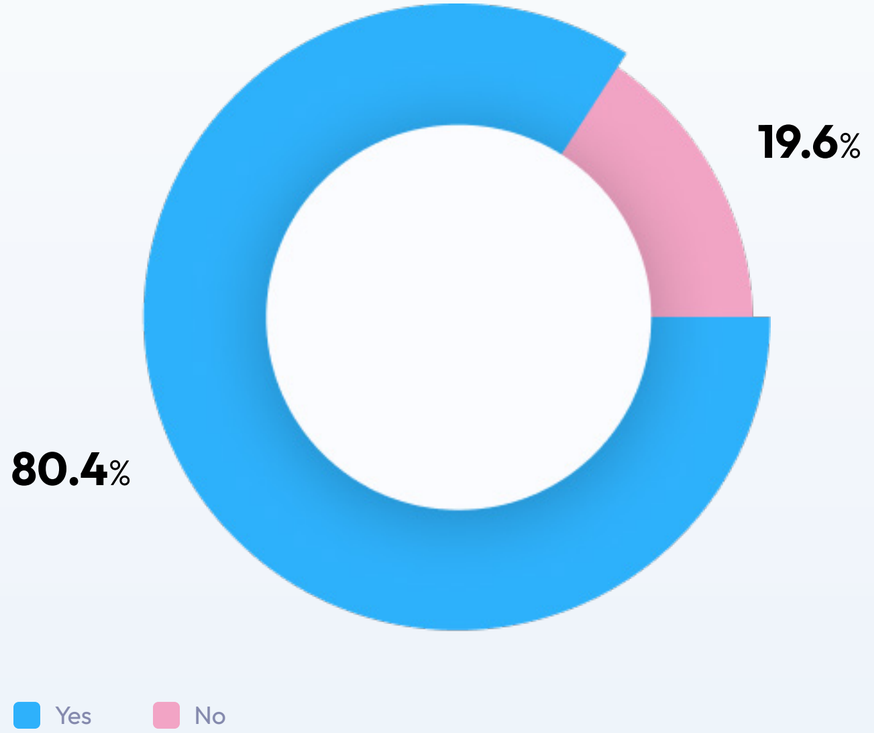
I've been working in software testing for nearly a decade now and testing front-tend applications has always been one of the most popular activities done by Quality Assurance folks. But what about developers? This report tells me a surprising story.

The most staggering result to me is the shift in testing responsibilities from testers to devs. It turns out that in 88% of cases, developers are at least as involved in testing as QAs.

Who's responsible for testing in your software development teams?



Have you performed software tests yourself over the last year?

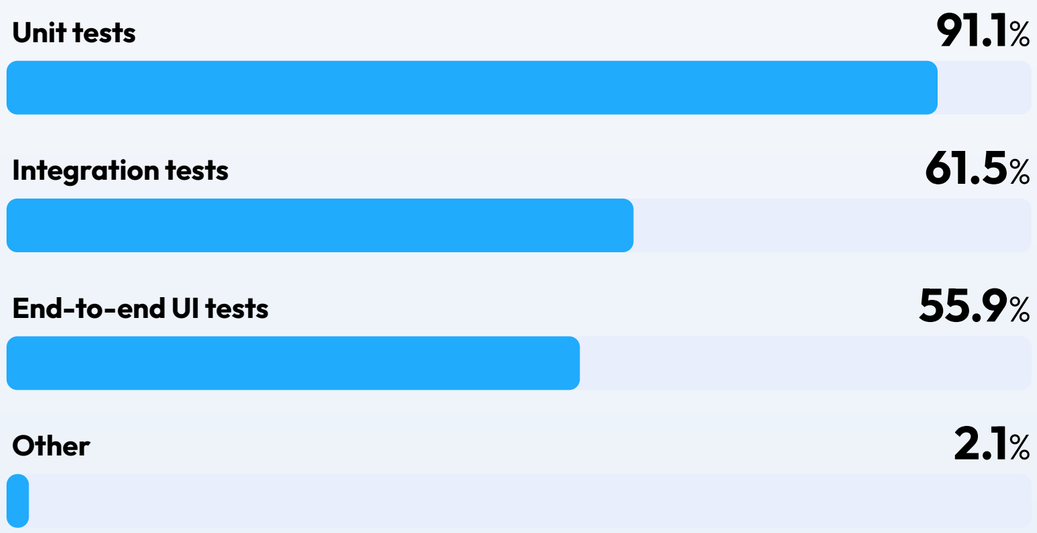


One of my core responsibilities as a Test Engineering Lead is encouraging our Quality Assurance people to become coaches and help developers get involved in testing. So I'm delighted to see my own experiences reflected in the survey, showing other teams making huge progress in this regard too.

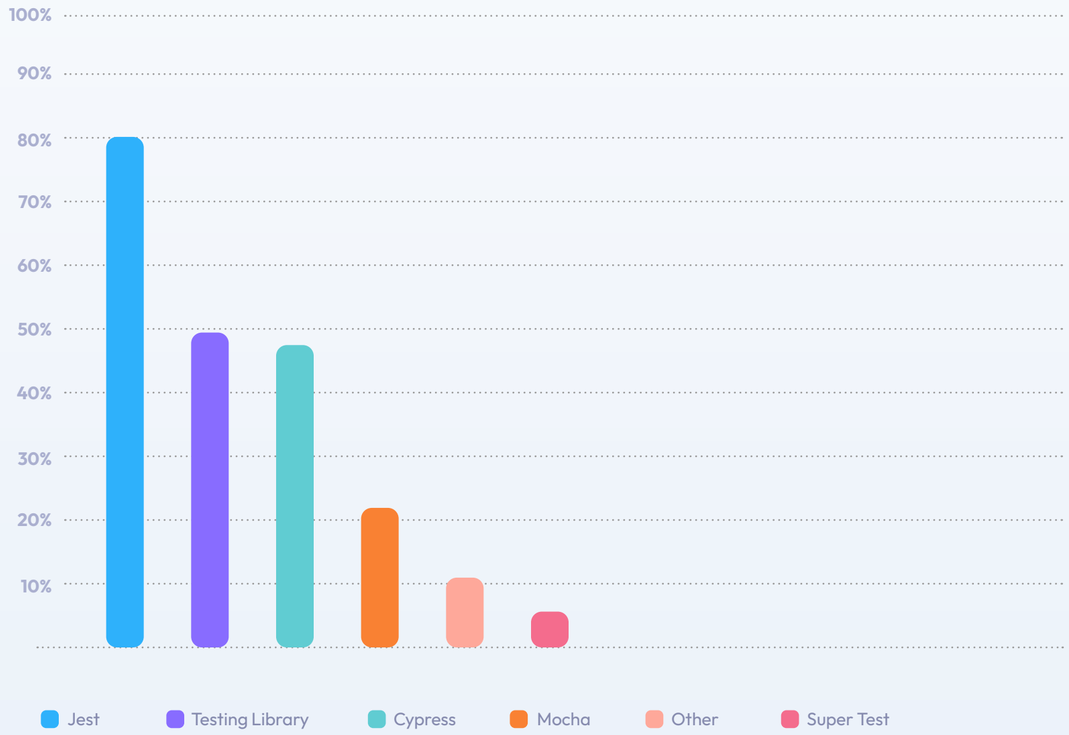
As a curator at Software Testing Weekly, I noticed that a lot of testers and devs choose JavaScript for testing. Nowadays, more people write about tools like Cypress and Playwright, rather than Selenium.

So I'm not surprised to see that nearly half of the respondents have already tried out Cypress. Alongside Jest, it's the most popular test tool. It suggests a growing interest in more robust testing and favoring tools with great DX (developer experience) and using the same language as for development.

What kinds of tests have you written yourself?



Over the past year, have you used the following testing tools?



Chapter 12 **Good practices**

Good practices are not one size fits all - they depend on your team

Gergely Orosz
The Pragmatic Engineer, author

Chapter 12 **Good Practices**

For project management, **69% of respondents use Scrum or Kanban**. Scrum at 52% is somewhat more common than Kanban - at 33% - and 17% of respondents use both. Two out of 3 frontend developers use one of these two methods when getting projects done.

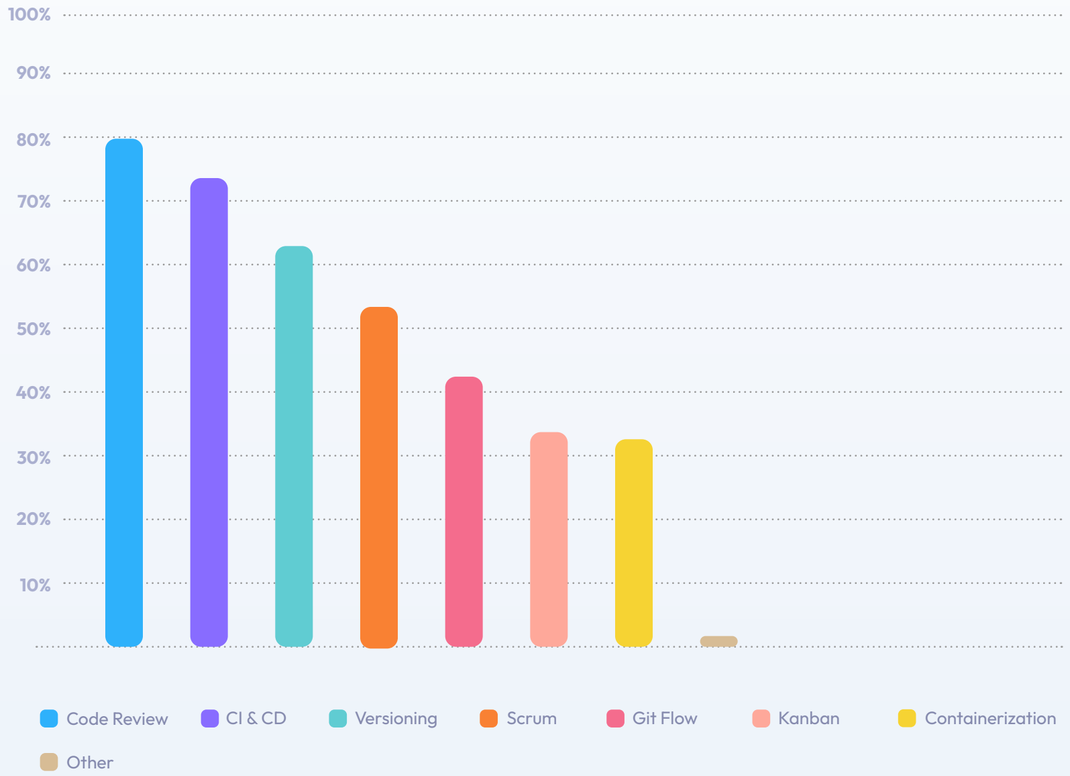
Companies where respondents did not report using either of these methodologies, tend to be mostly tech-first or digital-first companies, which rhymes with findings in my article How Big Tech runs tech projects and the curious absence of Scrum.

Unit testing is widespread among frontend engineers with close to 75% of respondents writing these kinds of tests. Integration and end-to-end tests are also common, with about half of respondents having written these tests.

Code reviews are common enough within the industry, with 80% of respondents mentioning they follow this practice. What was interesting to dig into is where code reviews are less likely to be a practice? Going through those who don't do code reviews, there is a strong connection between the size of the frontend engineering team and whether engineers do code reviews:

- Engineers working at large companies are the are more likely to do code reviews.
- Engineers working at companies with 50 or fewer employees are twice or more as likely to not do code reviews than those working at larger companies.
- One-person companies - understandably - are the most likely to not do code reviews.

What methods/good practices do you use in your frontend projects?



Chapter 12 **Good Practices**

Much of the above should not be surprising: the more engineers there are, the more value code review bring not just in spotting issues, but also in spreading knowledge better.

CI/CD is widespread within the industry. It's curious to see about a quarter of engineers not using CI / CD.

Not writing unit tests, not having CI/CD, and not doing code reviews are correlated.

This was one of the more interesting findings of this survey. Engineers who don't do two of writing unit tests, having CI/CD, and code reviews are likely to not do all three.

This finding should not be a huge surprise, as these three tools are connected. CI/CD makes less sense when there are no automated tests to run. Most code review tools integrate seamlessly into CI/CD tools.

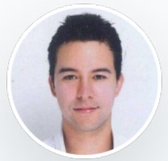
Still, this finding suggests that by introducing unit testing and setting up CI/CD, code reviews will likely follow. Or, perhaps the other way: engineers who want to do code reviews tend to write tests and will set up CI/CD.

Engineering practices worth mentioning which survey respondents brought up are below. Use these as inspiration to try out other approaches, if you've not done yet:

- Trunk-based development,
- Feature flags and feature toggling,
- Pair programming,
- Linting,
- Code style guidelines,
- JSDocs,
- Stakeholders Map,
- Design sprints,
- Prototyping,
- Semantic releases,
- "Fix-them-together days",
- Visual regression tests,
- Extreme Programming.

SEO is still the underdog - but for how long?

Rad Paluszak
CTO at Husky Hamster



Matt Diggity
Founder and CEO of Diggity Marketing

What is important for SEO on the frontend side?

For starters, the **responsiveness**. It is a must for most projects that are meant to work on different devices. Recently some platforms (Twitter) started hinting at the retirement of AMP (another approach to SEO-friendly mobile version for web apps). This makes RWD even more important.

When it comes to **performance**, this can obviously include many things. From an SEO perspective, it's all about PageSpeed measured by not only speed but also Page Experience. In November of 2020, Google added three new page experience signals that makeup what they called Core Web Vitals. These became implemented as Google's ranking algorithm around June 2021: Largest Contentful Paint (LCP), First Input Delay (FID), and Cumulative Layout Shift (CLS). All three of them are almost the sole responsibility of the frontend.

Next is **User Experience**, which is a very broad subject. However, one thing is certain: a lot of SEOs already use the term SXO to indicate that the typical Search Engine Optimisation must include User Experience.

The Search Experience Optimisation, then, is about combining the technical optimization for Google (and other search engines) together with making the website best for the Users. What can be better to serve the users in the exact way they expect, behave, and prosper in our app's ecosystem?

But there's also another very prominent upside to this. Happy users = retaining (or returning) users.

Good UX affects conversions, helps retain the users, or makes them want to return for more experience. All of these usually mean better monetization, which is at the center of most commercial apps.

How often do you take care of application's...

	Never	Rarely	Sometimes	Often	Always
Search Engine Optimization	28%	23%	22.5%	16.1%	10.4%

Accessibility

Dileep Marway
technology leader, educator & author

Firstly, I will start with the very important notion that accessibility should be already considered at the early stage of gathering product requirements i.e. do you aim for WCAG 2.1 AA standard, and do your customers expect this implementation. Back to the results!

I was shocked that the "always" answer was as low as 17%! This leads me to think that frontend developers are still reactive, rather than being proactive in their approach to accessibility. What I have seen in the past few years is that engineers are now proactively running automated accessibility tools as part of their CI/CD which is certainly helping to upskill engineers and other key members of the software team.

The surprising aspect for me is that accessibility is not deemed as important as user responsiveness, performance, or user experience. I have a feeling that **now, being aware there can be legal** ramifications towards companies due to poor accessibility of their products, I expect this to change in the future.

Comparing this with the question about taking care of accessibility from 2020 - given that the categories are somewhat a blanket 'yes' or 'no' - I feel that most frontend programmers feel they care about accessibility, though in most cases it will only be an ad-hoc review or audit.

I hope in the upcoming years we see accessibility like security, and we will design software products with accessibility in mind. Similar to how security has been of utmost importance in the past few years.

How often do you take care of application's...

	Never	Rarely	Sometimes	Often	Always
Application accessibility	9.2%	20.1%	26.5%	27.1%	17.1%

User Experience, responsiveness & performance

Adrian Twarog
“Development & Design” creator

User Experience is often the last consideration after things go wrong. Since functionality comes first, then user interfaces to accommodate those functionalities come second. Therefore, user experience is an expensive and timely exercise that only occurs in larger, specialized companies.

While the majority of respondents still prefer to use their own design systems with styling such as SCSS, the UX requirements still often fall upon the programmers who are simply given a feature to implement without any storyboarding or user flow examples.

Testing is such an important task in coding that nobody can imagine anymore that something could be released without being properly tested. The same goes for user experience testing which in my option, should be another element written into the workflow of CI/CD.

Sometimes users fail to complete a task because of CTR failure. **But just as often the task itself is being filled with errors that haven’t been spotted because there was no user experience testing involved.**

The same goes for Code Reviews versus Design Reviews. **A design review should always consider User Experience with iterations to improve not just the speed of compiles and performance of code tasks, but also user tasks (such as time for a user to complete an action via the UI).**

I’ve come to a conclusion that the 2020s have brought greater consideration to testing and reviewing, especially in code, and these same considerations should be applied to user interfaces and user experience as they are just as vital to the success of any system.

How often do you take care of application’s...

	Never	Rarely	Sometimes	Often	Always
Responsivness	1.5%	4.7%	13.2%	30.2%	50.3%
Performance	1.2%	4.6%	20.7%	37.9%	35.7%
User Experience	0.7%	1.9%	9.4%	32.4%	55.7%

Who came first, the user or the developer?

Kent C. Dodds
Remix Co-Founder

I'm actually pleasantly surprised by some of these results. **Sometimes it seems like the primary focus of developers is their own development experience, even at the expense of the user's experience. These results prove otherwise. Developer experience is input to UX, and that's how it should be prioritized.** If we're not building software for the user's experience, then what are we even doing?

I'm unfortunately not surprised by the results for accessibility. At least we're being honest with ourselves. Acknowledging our shortcomings is the first step to improving them! Hopefully, these results will be a wake-up call for us to remind ourselves that accessibility is an important input to the user experience for a great number of users (both those using assistive technologies and those not). In fact, for some users, the only way they have any "experience" at all with apps you build is if you account for accessibility, so we would do well to act accordingly.

How often do you take care of application's...

	Never	Rarely	Sometimes	Often	Always
Developer Experience	2.5%	5.8%	18.4%	37.2%	36.1%

Chapter 13 **Future of frontend**

The frontend may be entering a stability phase

Marek Gajda
CTO at the Software House

Chapter 13 **Future of Frontend**

The first thing that I've noticed behind the numbers is my favorite sin(x)/x function and its relation to programming in general. Software development is still at an early stage, a toddler among industries. There are evangelists who proclaim that everything has already been said in topic X, hence current methods should be considered standards. A minute later heated discussions start, with people having 180° opinions about the subject, so the technology shifts to Y. Then lo and behold, the first method is making a comeback with slight adjustments that prevent it from being too extreme and closer to the "center". After that, the Y fans adjust their solutions to be closer to "central opinion". Eventually, two extremes become a "compromise" that turns into something of a standard. It's called the annealing of a function (yes, I wanted to be a maths teacher back in the day). Just take a look at the graph below.

It seems that frontend development is entering a more "stable" phase. Some issues like accessibility or server-side rendering are not up for discussion anymore. However, a few years ago, frontend was at the beginning of this path, methods reflecting completely different visions, ideas, and approaches. Everybody in IT knows the "new frontend framework every day" jokes. But there's less of that, and we're at the point where the sin function is slowing down and flattening, and the stabilization process begins.

Let's break down some examples of trends stabilizing that I can pick up from the survey responses.

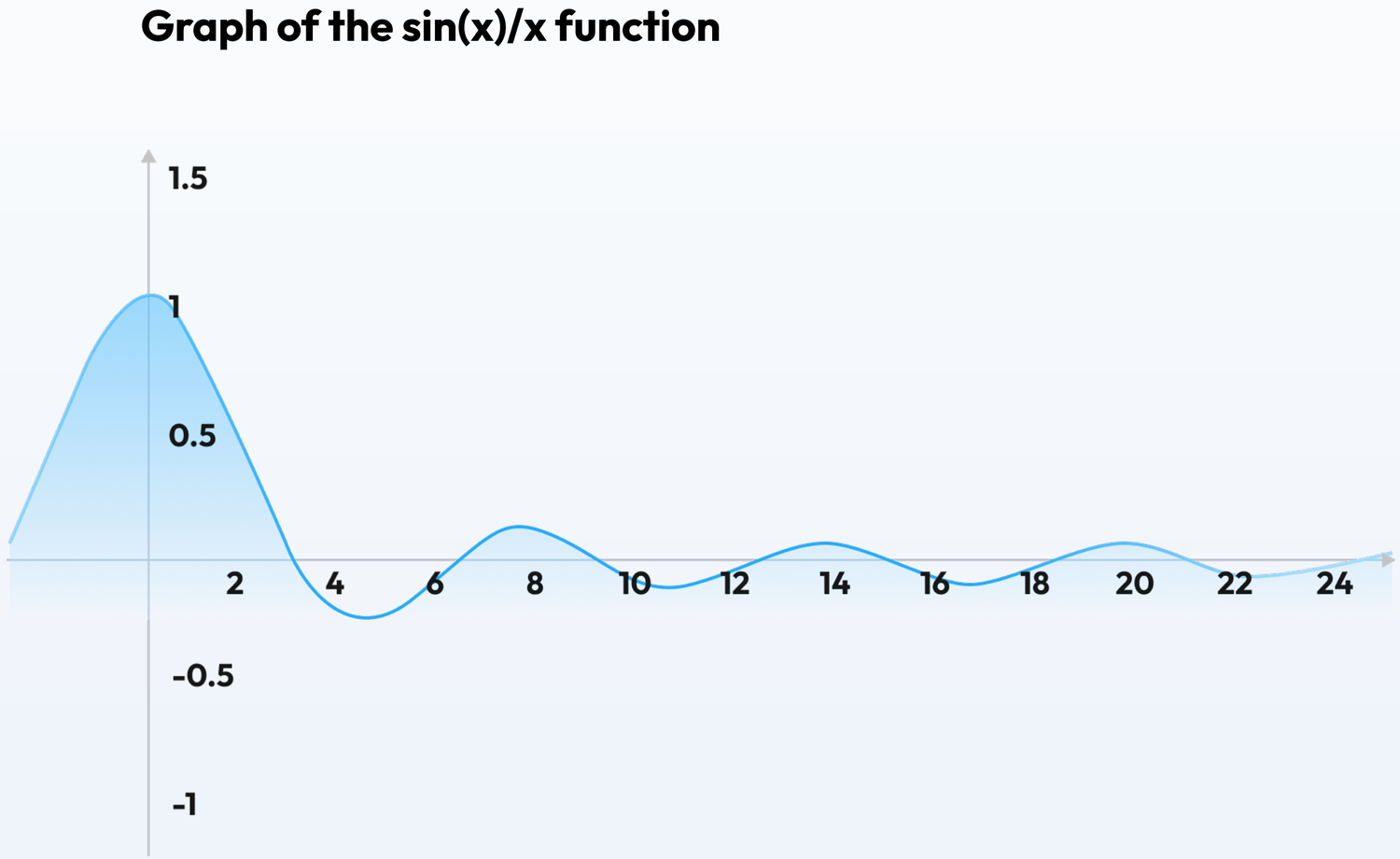
In your opinion, which of these trends/solutions will rise in popularity, and which will be pretty much dead in 2 years from now?

	Gain popularity	No changes	Die	No opinion
Accesibility	63.1%	30.5%	0.4%	6%
Atomic design	23.9%	37.1%	8.6%	30.5%
Component-driven development	58.4%	27.9%	1.5%	12.1%
Cross-platform applications	60.6%	24.8%	3.3%	11.3%
GraphQL	42.4%	34.7%	9.1%	13.8%
Headless CMS	39.4%	30.8%	4.8%	25.1%
JAMstack	26.5%	29.8%	10.8%	32.9%
Micro frontends	37.2%	23.5%	13.3%	26.1%
Online page builders	29.8%	35.8%	11.8%	22.6%
Progressive web applications	42.6%	34.4%	11.8%	11.2%
Server-side rendering	60.5%	27%	4.8%	7.6%
Web components	45.2%	27.1%	11%	16.7%
WebAssembly	45.5%	25.4%	5.6%	23.6%

Back in 2020, 20% of respondents predicted the death of micro frontends and it seems like they're not going anywhere. Micro frontends still have borderline opinions, and I'm wondering what the compromise will look like in the future. Luca Mezzalira in his "Micro-frontend" book presented 12 different concepts for micro frontends which means that the solution itself is still crystallizing internally. I suspect that people who vote "for" micro frontends support a different concept than those who vote "against".

Server-side rendering is already heavily flattened (60% vs 5%) but I'm quite surprised that this is where the stable axle landed. History lesson: pages were initially rendered on the backend. Then people went "It's kinda silly making rounds all over the internet, being incredibly slow, a browser on the front should do it". Then the opposition went: "Okay but it's still kinda slow, maybe we should go back to the backend?" To which the response was "Hey, what about a bit on a server and a bit on the client-side?" So we're basically back to square one, that's exactly how it worked 20 years ago. But this time, we were able to modify the method after years of new experiences, experiments, and changing things internally.

My guess is that domain-driven design is next in line. 9 years ago the idea was to separate the business logic from technical issues (routing, databases, performance, optimization, etc.). **There was code that described what the app does and engineering&technical code. Everybody was crazy about the idea but hardly anyone was able to do this. The concept was right, only the timing was wrong - paradoxically, we didn't have mature enough technology to carry out this task.**



Currently, this technology is shifting in a way that if someone vocally supports DDD, they might be right. We finally got the tools to turn theory into practice and separate business logic from technical bits for real. Some solutions on the list above, e.g. headless CMS, do exactly that.

Developer empowerment and responsibility

Back in prehistoric times, there used to be one gigantic, unified app. When someone new entered the project, they were told what and how to do it, and there was no discussion or choice.

Now, if you want to build responsibility and ownership in your frontend team, make sure that developers have a lot to say about technology and engineering issues. It's not about the app's features, it's about HOW it will be built. Companies finally began to give developers more autonomy in decision-making on how things are done, instead of making important decisions over their heads. Not only because frontend programmers earn a lot of money. The more people have to say about what they do, the more ownership they feel.

Among the trends included in the question, we've got **component-driven development, GraphQL, micro frontends, and web components - all divide apps in such a way that everybody in the software team can work on their part which later will be incorporated into a larger whole. But every developer is responsible for their area and determines how it is to be done.** This 100% fits the concept of broader developer autonomy with countless problem-solving options.

One app to rule them all. What's next for mobile?

60% votes on growing cross-platform applications (why make two separate apps when there should be one app that works everywhere), and 42% on progressive web applications (how to make that app work everywhere) may prove that we do not need native mobile anymore.

Back in the day, the only reality was native: one app for the web, one for mobile. Nobody could imagine a business without a mobile application, even when they didn't need them at all. Even I used to work with a "mobile app generator" client who offered simple mobile apps with the store's name, contact, promotions, loyalty points, opening hours, and address. The only advantage of doing a native app out of it was that you could click on the address and Google Maps would open with directions. Groundbreaking indeed.

Then, everybody noticed that creating and maintaining mobile apps actually cost a fortune. A lot of companies dropped their dedicated mobile teams because people figured out that all you need to do is open a website that will scale for smartphones. Not building an entire app from scratch! Then, according to the trend stabilization phase, we went through: "Hmm, what about a hybrid?" - "Nope, back to native" - "So, you're back for a new, hybridized hybrid after all?" swings.

I'm really interested if the progressive web applications trend will stabilize here, or are we going to have another pendulum swing. I have a feeling that "one app policy" will stay with us for longer but I'm not so sure whether PWA is the best solution to this problem that we can come up with.

Just a reminder that Google came up with Trusted Web Activity (TWA) and they tried to set it as a standard. It's still a fresh topic but I have a feeling it will wither soon enough, I don't see any interest in it from frontend developers, managers, or companies. Apple is unlikely to come up with their own "standard" because they have an original iOS and they would have to admit that native apps are a thing of the past.

Possible performance problems ahead

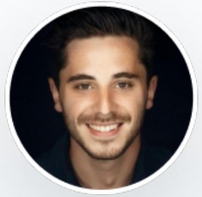
I saved the most surprising for last, have you seen WebAssembly's results? It really seemed to me that WebAssembly was a solution used for greater optimization in a handful of companies, giants like Facebook or Gmail. I was proven completely wrong. 46% of respondents predict the growing popularity of WebAssembly, and honestly, I'm shooketh! Maybe I live in a world where WA is the last resort option when you used the frontend capabilities and hit the wall, since it's hard to write, and difficult to maintain. When all the methods have been used and the solution is still too slow, only then you go for WebAssembly.

We all know that there's growing pressure for app performance to be constantly improved. Are web apps so complicated now that people need last-resort options because they have such performance issues that nothing else will do? Should this increased WebAssembly interest be the first omen of general performance issues? I'll definitely follow this topic closer in 2022.

BIOs

Meet our expert commentators

BIOs

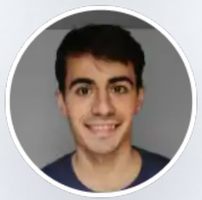


Sebastien Chopin

CEO at NuxtLabs and author of Nuxt framework. Passionate about open source and developer experience. He strives to make the web faster and create a flow feeling for developers by making the best tools to express their full creativity.

Chris Coyier

Founder of CSS-Tricks, co-founder of CodePen, co-host of „ShopTalk" podcast. A web designer and developer that tries to help other people get better at those things.



Olivier Tassinari

Software engineer, CEO at MUI, and co-creator of Material-UI.

Andrzej Wysoczanski

Head of Frontend at The Sofware House. With many years of experience in software development, Andrzej found his place managing a brood of frontend developers. He loves his jobs for giving him opportunities to share knowledge and help others enter and thrive in challenging tech projects.

Dawid Dylowicz

Staff Test Engineering Lead at Onfido. Quality Assurance Lead with a decade of experience in software testing. Helping over 4,000 testers discover the best news via the Software Testing Weekly newsletter.

Gift Egwuenu

Developer Advocate at Cloudflare. She has over five years of experience in web development and building tools to help businesses grow. Her career moved from front-end development to developer relations. Gift gladly shares her experience in web development, Jamstack, and career-related topics to help other people in the tech industry level up their skills.

Marcin Gajda

Frontend Team Manager at The Software House. Marcin is a software engineer with a decade of experience in different fields of web development and a handful of delivered products. He aims to make other developers write better code and strive to always find the best solution. Marcin is also an occasional speaker, a better code review evangelist, and a space exploration lover.

BIOs

Marek Gajda

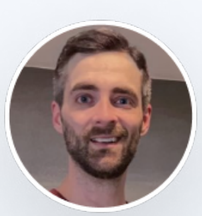
CTO at the Software House. A former full-stack developer and an experienced Scrum Master who completed projects in PHP, Node, Java, Ruby, Python, and .NET. He now leads dozens of product development teams and teaches technology managers how to scale up their IT departments effectively.

Gergely Orosz

The creator of The Pragmatic Engineer, the #1 technology newsletter on Substack. Formerly an engineering manager and engineer at Uber, Skype, and Microsoft, he now researches and writes about engineering management and software engineering topics especially relevant to Big Tech and high-growth startups. Gergely is an author of multiple books, including “Building Mobile Apps at Scale” and “Growing as a Mobile Engineer”.

Ives van Hoorne

Co-Founder of CodeSandbox. Ives loves building things that other people can use to build things. He started CodeSandbox as an open--source project when he was studying, and as it grew, it became a company of 29 people that are now working full-time on it.



Jay Phelps

Web Platform at Netflix. WebAssembly Community Group member and RxJS core team alum. Over 20 years of experience across a large number of platforms, frameworks, and languages, with a focus on libraries, tooling, and Developer Experience in the last 7 years.

Luca Mezzalira

Luca is a Principal Solutions Architect at AWS, an international speaker, and the author of “Building Micro-Frontends”. Over the past 18 years, he’s mastered software architectures from the frontend to the cloud, providing the right solution for the context.

Dileep Marway

Powerful technology leader of 16 years, who is working towards a CTO role. Previously worked for “The Economist” and created an ”Engineering centre of excellence” in Central Birmingham from scratch, eventually growing the technology practice to 80 members in 3 years. Having graduated from Aston University, Dileep cares about giving back to those who educated him, so he mentors undergraduate and MBA students as a board member for TedX Aston University.

BIOs

Samuel Snopko

Head of DevRel at Storyblok. Samuel is responsible for the developer relations at Storyblok. As the headless system's head of DevRel, he spends most of his time buried in the documentation and creating various experiments and demos. He always defines himself as Creative FrontEnd Knight & DesignOps enthusiast with a passion for Jamstack and the beautiful web.

Santosh Yadav

Google Developer Expert for Angular, GitHub Star, and an Auth0 Ambassador. Co-founder of This Is Learning, author of the Ngx-Builders package, and part of NestJsAddOns core Team. Also runs "This is Tech Talks" talk show, where he invites the industry experts to discuss different technologies. Santosh works as a software consultant and loves contributing to Angular and NgRx.

Kent C. Dodds

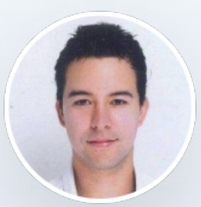
Remix Co-Founder, JavaScript engineer, and teacher. He's also active in the open-source community. He likes his family, JavaScript, and Remix.

Rad Paluszak

A software developer and solutions architect with 20 years of experience. A technical mastermind in the SEO industry since 2010. He helped Matt Diggity run his SEO agency The Search Initiative and recently founded Husky Hamster - an outreach link-building company.

Adrian Twarog

A full-stack developer and designer who runs his own agency working on a range of small to large projects in the website and app spaces. He's a creator of the "Development & Design" YouTube channel where he teaches others about current and upcoming trends and industry standards in programming and user interface space.



Matt Diggity

A search engine optimization expert focused on affiliate marketing, client ranking, lead generation, and SEO services. He is the founder and CEO of Diggity Marketing, The Search Initiative, Authority Builders, LeadSpring LLC, and host of the Chiang Mai SEO Conference.

THE SOFTWARE HOUSE

The Software House

A product development partner for technology-first companies that build their competitive advantage on technological excellence.

Drop us a line at **hello@tsh.io**, book a free 1-hour consultation with our frontend experts, and speed up your development and innovation with the right strategy.