

What is JAMstack?

JAMstack is a term that describes a modern web development architecture based on **JavaScript, APIs, and Markup (JAM)**. JAMstack isn't a specific technology, but rather a different way of building apps and websites.

Instead of using a traditional CMS or site builder, a JAMstack site splits up the code (JavaScript), the site infrastructure (APIs), and the content (Markup). These will all be handled in a [decoupled architecture](#) and with a clear split between server-side and client-side. In fact, the main idea behind building JAMstack websites and applications is to push as much of the load as possible away from the server and onto the client. By doing so, it dramatically reduces the number of requests sent to a server and thus eliminates a lot of the waiting time that comes with a server handling a request and sending it back to the client.

The name itself was originally coined by Mathias Biilmann, co-founder at [Netlify](#), who wanted to make it easier to refer to this new way of doing web architecture.

Table of content

- [A brief history of developing websites](#)
- [Why use the JAMstack architecture?](#)
- [Using JAMstack vs. a traditional CMS for your website](#)
- [What are the pros and cons of using JAMstack?](#)
- [What is a static site generator?](#)
- [Is JAMstack the future?](#)
- [Can I use Umbraco for JAMstack websites?](#)

A brief history of developing websites

To understand why the JAMstack architecture is becoming popular today and not 10 years ago, it's worth taking a

quick history lesson. In fact, let's take a look at how websites used to work and how it's evolved from there.

In the very early days of the internet, everything was more simple than we might think about today. The first websites that were built were all really just a folder with HTML files that could be requested by a browser. And that was pretty much it. The HTML would determine how the website looked and the content in it, while the browser's only job was to let the user view the page.

As the internet and the technologies surrounding it evolved, it allowed for more advanced websites and as a website owner, you got a lot more possibilities of what you could do. The first real change here was how content was being served. Static HTML files were no longer enough. Instead, it was necessary to serve customized content to the users of a website.

In order to serve customized content, the HTML of the website had to be built by the server at every request. This was a giant leap from serving simple HTML files. Now the user would request a page and instead of having it ready, the server had to consult a database and then build up the right content for this specific user.

The benefits were clear: you could now serve more customized content to the user.



The downside? It required more server resources to build and thus the content would be delivered to the user slower than if it had been an old-school static HTML file.

Unfortunately, there was nothing to really do about it at the time. If you wanted to serve customized content you had to let your server and database build it. That all changed with the invention and maturity of JavaScript. Today it might be hard to imagine the internet without JavaScript, but that was the case for the first few years before [Brendan Eich](#) wrote the first prototype Called "Mocha". It was later renamed "LiveScript", but that only lasted 3 months until JavaScript became the final name.

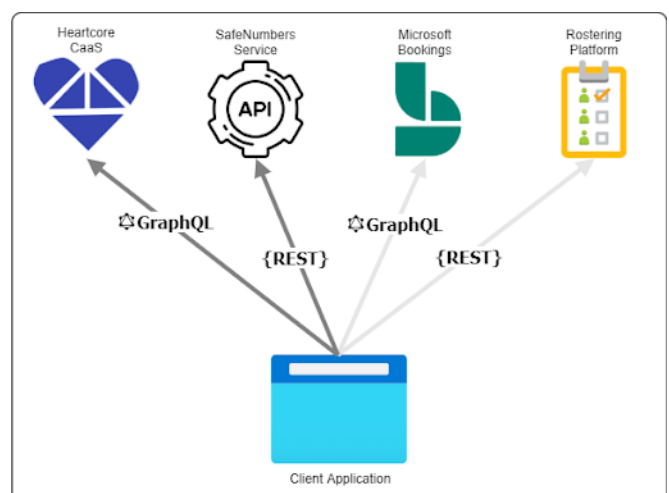
JavaScript was developed to allow browsers to dynamically alter a page after it was loaded in the browser. This fundamentally changed how developers could work with dynamic behavior in websites and since all major browsers adopted JavaScript, it became a universal runtime layer. **The browser was no longer a simple document viewer but could handle complex tasks that altered how content on a website would look and behave.**

This development added huge benefits to all websites. Not only could they now use JavaScript to add dynamic capabilities to their website after it had loaded, but they could also move some of the workloads from the server-side to the client-side instead. This not only meant a better user experience for website visitors, but it also meant faster delivery as the browser would now help carry some of the load of building the page.

This last point is important when we're talking about JAMstack websites: **shifting the workload from server-side to client-side.**

Why use the JAMstack architecture?

The main reason to go with JAMstack instead of a traditional approach to website development is to go as server-less as possible. **That means taking away as many tasks as possible and pushing them to the client instead to enable**



fewer server requests and thus better performance and faster load times.

The only way to do that is to think of the tech stack in a different way than previously. The classic way of talking about a tech stack is to mention operation systems and databases.

With JAMstack you instead focus on a different level of the tech stack: the level that is actually the real constraint on a project.

These constraints boil down to the 3 core parts:

1. **JavaScript** in the browser as the universal runtime layer to add extra functionality and dynamic behavior.
2. **APIs** to replace the database and fetch everything you need directly from the services required.
3. **Markup** to serve the actual website content and HTML needed.

Let's have a look at the three individually.

JavaScript

JAMstack websites are built to be flexible and not have one part of the architectural setup limit another. When you use a traditional content management system (often referred to as using a monolith system) you have everything coupled together. The database, content, and frontend templating are all handled in the same system, which means it's all dependent upon each other. While there's nothing inherently wrong with

that (in fact it's often a great help), it can be limiting in terms of functionality and performance.

To get around this lack of flexibility, a JAMstack site is instead completely split up and each part serves more specific and limited purposes.

JavaScript is in charge of displaying the content to the user and ensuring a great user experience. And since it is all decoupled from the content, it has complete freedom in how to do it. And although the "J" in JAMstack stands for JavaScript, it actually doesn't even have to be JavaScript. If you prefer to use something like Ruby, Python, or Go you can do that as well.

And that's the beauty of it all: you have the flexibility to choose which framework to use and how you want to use it.

APIs

Application programming interfaces (APIs) are what is used to define what type of interactions are possible between multiple software services. Initially born in 2000, APIs have evolved a lot since then. As with a lot of other technologies in that age, APIs were developed to fit the workflows and processes of web development back then.

That meant APIs were only meant to be consumed by server-side applications, as that was the way it was done at the time. But with the development of new JavaScript standards, this all changed. Web APIs were developed and became available to any JavaScript client running in a browser.

This was a complete game-changer and is one of the big momentum swings for the JAMstack architecture. With web APIs, it's no longer needed for the server to do all of that heavy lifting. Instead, it can all be moved to the client-side with JavaScript. And with APIs becoming more and more available for pretty much anything, there's a big movement towards using them as microservices. These microservices are essentially small pieces of code that serve specific functions, completely independent of other services. This creates an architecture of services that work independently, but seamlessly integrates and connects to deliver a web of functionality.

The beauty of web APIs is the flexibility you get to use best-of-breed solutions and services instead of being forced to use something by your current system.

Markup

Content is king and in JAMstack it comes in the form of Markup.

HyperText Markup Language (HTML) is at the core of the modern web and is understood by all browsers. The core functionality of HTML is to present a Document Object Model (DOM) that is ready to be parsed, presented, and processed by web browsers. But it is not limited to web browsers only. Different technologies like search engine crawlers, smart devices, and screen readers can also work with the DOM.

While HTML has come a long way, its primary job is still to serve up content and its structure to the browser to display. This has not fundamentally changed with JAMstack sites. But the way it is being done and served has.

Instead of relying on the server to build all of the content with every single request by a client, it is relying on cached content. This is done by using a build tool (like a static site generator or a frontend build tool) to pre-build the markup and deliver it to a Content Delivery Network (CDN). With this approach, the server will not have to work in real-time on new requests, but instead work only when new changes are made to any content or assets.

With the markup prebuilt and ready to be consumed, it is up to JavaScript to call in the content it needs. If dynamic or customized content is required that'll be handled with APIs instead of having the server request it from a database.

Using JAMstack vs. a traditional CMS for your website

The main difference between a JAMstack website and a website built with a content management system is how tightly the content, code, and design are connected.

With a CMS you'll be handling everything in the same system. You'll have a backend interface where your developers will be handling all of the code and design templates. Your content will also be created, updated, and managed from a backend interface and it will all be stored in databases on a server. Once a page on the website is visited, it will be assembled by the server and delivered to the visitor and the page is shown.

With a JAMstack site, you're decoupling all of these parts and handling them separately. This decreases any dependency and gives you the freedom to choose your

own tech stack. One key difference is how pages are generated for a website visitor. With a JAMstack site, you're almost entirely relying on the client-side instead of the server-side. Your pages will be stored in a cache - typically using a CDN - as HTML and JavaScript, which will be ready and waiting for the visitor to come by. Once a visitor visits your page, it'll all be ready for it to handle - without having to go back to the server and assemble it.

What are the pros and cons of using JAMstack?

Pros of using JAMstack

- **It can be blazing fast.** Since the JAMstack relies less on a server and the database, it'll often load very fast for the page visitor. By effectively caching all of your content on a CDN and focusing on a clean front-end code you avoid a lot of the processes normally involved in loading a web page.
- **The website is secure.** The decoupling of the backend and frontend effectively means that any security flaws that might be exploited in the frontend code won't mean a security breach in your backend application. These are separated by an API and will mostly be read-only. At the same time, you're much less reliant on a database, which minimizes the potential issues that can occur.
- **Scaling is easy and cheap.** Since you're not relying on heavy server or database processing you won't run into the same limitations as you can normally run into. You're only serving static assets and lets the client handle most of the heavy lifting. With a powerful CDN, you'll be needing much fewer resources to scale your website. This means that scaling is not only easier but will often be cheaper as it won't require you to upgrade servers.
- **Great developer experience.** As a developer, you get all the flexibility in the world, since you can choose your own tech stack. Developers can do their thing without worrying about the limitations of a certain platform or technology. And with the rise of microservices and reusable APIs, it's becoming a lot simpler to reuse functionality across multiple websites or applications.

Cons of using JAMstack

- **Not content editor-friendly.** Developers might love it, but that is not necessarily the case for content editors and marketers. Since you have to serve your content as Markup, your editors will need to be fairly technical to create and update content. This can often require teaching your editors new skills and will slow down content production as they lose editor features they're used to from a CMS. At the same time, they'll be responsible for proper media management, which can be a tedious process.
- **Updates = coding.** If you want to make any updates to your templates, you'll need to do it by coding. With the content being decoupled from your frontend there's no easy way for editors to customize the templates. This will often mean that developers will have to spend more time on making the updates since they can't be easily changed in other ways.
- **Dynamic features require more heavy lifting.** JAMstack sites are great as long as we're building pages with text and images. As soon as your site requires dynamic features you'll potentially run into issues. Without having a database to process your requests, you will need to do more of the heavy lifting yourself with your own code or API calls. This doesn't mean it's not possible, but it requires more resources to achieve as you don't have these dynamic features as part of your architecture.
- **Live and die by third-party systems.** Since your website is heavily reliant on third-party systems and APIs you're relying on them to be consistent. If a third-party system goes down or an API goes down, so does your website (or parts of it). This is no different from a normal website where the server goes down, but with a JAMstack site, it's very limited what you can do yourself to fix the problem if it's an issue with a third-party provider.

What is a static site generator?

The number one thing that static site generators have in common is that unlike conventional website builders they don't depend on a web server. Instead of building each page on demand, a static site generator generates all the pages of the site when there are actually changes to the site. **This means a visitor doesn't have to wait for the database to create a page before loading it.** Instead, the page will be ready right when

the visitor lands on the page and the client will be doing the heavy lifting. This makes caching much easier and the site faster by utilizing a content delivery network and client-side processing instead of relying on a server.

Is JAMstack the future?

Yes and no. For the right project, JAMstack is definitely a good solution and the decoupled architecture brings a lot of advantages to many websites.

The state of JAMstack right now is still on the technical side though, with both developers and editors needing a certain level of technical skills to be successful. So if your team does not possess the necessary skills to work efficiently with the JAMstack, then you might not see the results you want to.

It won't matter how fast your website is to a visitor if all they see is the same old content, that your editors are struggling to update.

If you want to negate the primary con of a JAMstack site - the editor experience - you should take a look at using a headless CMS. By doing so, you'll still be able to rely on a decoupled architecture with RESTful APIs serving the content, without compromising the editor experience.



Report: The state of Headless CMS 2020

Topics covered in the report:

- Why discussing headless is important
- What the main reasons are for choosing headless
- Whether demand is increasing - and why
- Which industries are using headless, and for which type of project

[Get the report](#)

Can I use Umbraco for JAMstack sites?

Yes, you can use Umbraco to build a JAMstack website or application.

The Umbraco CMS is not a headless CMS out-of-the-box, but it's built with the flexibility and extendability to make it into a headless CMS if you want to. Everything in the Umbraco backoffice is connected with open APIs, so if you want to use a different front-end to serve your content, you can build it. This would require you to customize Umbraco, but it is definitely a great use of the open source version of Umbraco.

If you don't want to build it yourself, you can go with Umbraco Heartcore.

Umbraco Heartcore is the official headless CMS, hosted and managed by Umbraco HQ. It's based on the open source CMS and the editor experience that you know from the open source CMS, but packaged as a managed and hosted SaaS solution. This gives you a content management system that is focused on giving your content editors a great experience, without having to

Umbraco Heartcore: A headless C



worry about learning Markdown or having to deal with managing media.

For developers, it's ready to use with open RESTful APIs and GraphQL that you can connect to the frontend of your choice. And with Cloudflare CDN included and a managed API, you don't have to worry about maintaining another codebase or API.

If you're interested to learn more about [Umbraco Heartcore you can go take a look at the product features here.](#)

Do you want to try it out for yourself? Then you can [take a 14-day free trial.](#)

Related words:

[CMS](#)

[Headless CMS](#)

More info:

[Umbraco Heartcore](#)

[Umbraco Heartcore documentation](#)

[Umbraco Heartcore trial](#)

Loved by
developers, used by
thousands around
the world!

One of the biggest benefits of using Umbraco is that we have the friendliest Open Source community on this planet. A

Number of active installs

731,438

Number of active
members in the
community

221,745

community that's incredibly pro-active,
extremely talented and helpful.

If you get an idea for something you
would like to build in Umbraco, chances
are that someone has already built it. And
if you have a question, are looking for
documentation or need friendly advice,
go ahead and ask the Umbraco
community on Our.

Known free Umbraco
packages available

1,211

Want to be updated on everything Umbraco?

Sign up for the Umbraco newsletter and get the latest news and special offers sent
directly to your inbox

What is your first name?

What is your email?

Sign me up for the Umbraco newsletter

Why Umbraco?

Case studies &
testimonials

Why choose
Umbraco?

Products

About Umbraco


About us

Work at Umbraco


How to download
Umbraco

Become an

Other resources

 Developers &
Community

 UmbracoTV

 Umbraco on
Github

Follow us

 Twitter

 Facebook

 LinkedIn

Training

Trust Center

Umbraco Tech
Partner

Terms and
conditions



Blog



Knowledge
Base