

Learn best practices you can apply to every Express API you create in my [Guide to Express API Validation < https://simonplend.com/a-guide-to-express-api-validation/>](https://simonplend.com/a-guide-to-express-api-validation/)

[NODE.JS < HTTPS://SIMONPLEND.COM/CATEGORY/NODE-JS/>](https://simonplend.com/category/node-js/)

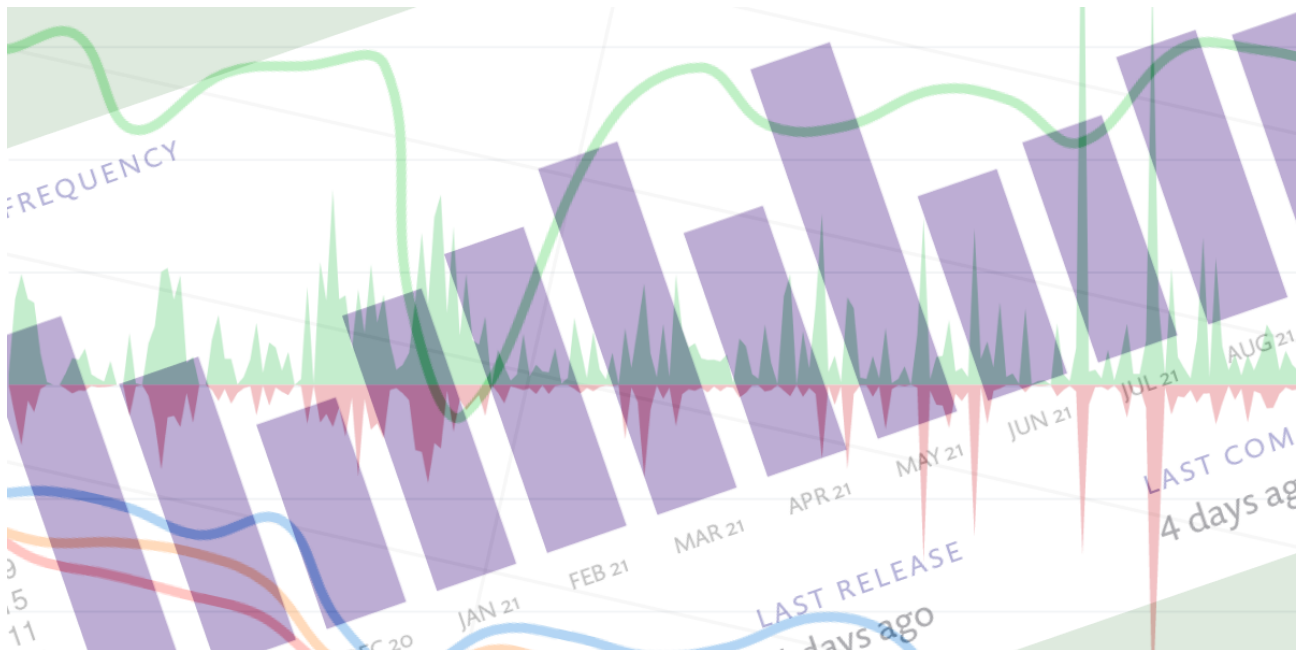
Guidelines for choosing a Node.js framework



[September 28, 2021 < https://simonplend.com/guidelines-for-choosing-a-node-js-framework/>](https://simonplend.com/guidelines-for-choosing-a-node-js-framework/)



[No Comments < https://simonplend.com/guidelines-for-choosing-a-node-js-framework/#respond>](https://simonplend.com/guidelines-for-choosing-a-node-js-framework/#respond)



7 min read

I often see the question "What's the best Node.js framework?" crop up on message boards and social media. The replies tend to be full of strong opinions. Some developers will even get into arguments about it. If you're trying to figure out the right framework for your project, none of this is going to be much help.

With lots of frameworks to choose from, and all those strong opinions, it's easy to feel a little lost. Comparing frameworks based on the features they list can be a headache,

and features are only part of the picture. It would be awesome if you had a clear checklist you could evaluate Node.js frameworks against.

I can't provide you with an exhaustive checklist, as every project, team and developer have different needs. I do have some guidelines though to help you think about what matters to you when choosing a Node.js framework.

Thank you to the following folks who shared [on Twitter <https://twitter.com/simonplend/status/1438479239041404932>](#) what they consider when choosing a Node.js framework: [Matt Hinchcliffe <https://twitter.com/i_like_robots>](#), [Alex Wilson <https://twitter.com/AlexWilsonV1>](#), [Alexey Antipov <https://twitter.com/aantipov>](#) and [Anthony Bouch <https://twitter.com/58bits>](#).

Jump links

- [What's your use case?](#)
- [Framework styles](#)
- [Support for promises and async / await](#)
- [Documentation](#)
- [Practical examples](#)
- [Community and ecosystem](#)
- [Project health](#)
- [Conclusion](#)

What's **your use case?**

Context is everything. Try and get clear on your requirements before you start looking at frameworks and their features. This will help you figure out if a framework is designed to support your intended use case. If it's not, you could find yourself fighting against it when building your application.

Here are some use cases you might want to consider:

- **Full stack application vs API.** Will your application be serving up HTML pages or will it be an API sending JSON responses? If it's an API, will it be [REST <](#)

https://en.wikipedia.org/wiki/Representational_state_transfer> or [GraphQL < https://graphql.org/>](https://graphql.org/) based?

- **Server-side rendering.** Do you plan to use React or Vue components in your application? Some frameworks provide helpers to simplify integrating these frameworks in your applications.
- **Serverless.** Will you be deploying your application to a serverless platform? For some serverless platforms like AWS Lambda, [cold start time < https://www.techtalksbyanvita.com/post/serverless-cold-starts-can-we-mitigate-these>](https://www.techtalksbyanvita.com/post/serverless-cold-starts-can-we-mitigate-these) really matters. If a framework has a lot of initialisation to do before it can handle a request, it could have a big impact on your response times.
- **Real-time.** If you want to use WebSockets, look for built-in framework support, or a community maintained library which you integrate with the framework.
- **TypeScript.** Do you want to write your application in [TypeScript < https://www.typescriptlang.org/>](https://www.typescriptlang.org/)? Some frameworks are designed only for TypeScript, some include type definitions, and others only have third-party types available which are maintained by the community. Even if you don't intend to write your application in TypeScript, your code editor might be able to give you helpful hints based on a framework's type definitions.

Framework styles

Some developers love frameworks which come with "batteries included", whereas others loathe them. "Batteries" in this context refers to features beyond HTTP request/response handling and routing. This might include things like validation, logging, authentication, database abstractions and dependency injection. [Frameworks in this category tend to be highly opinionated](#) about how applications should be built. They require you to structure things in a specific way so you can take advantage of the features which they offer.

[At the other end of the spectrum you'll find minimal frameworks.](#) They tend to offer HTTP request/response handling, routing, and not a whole lot more. They're not particularly opinionated and provide you with the freedom to structure your applications as you wish. As they provide a limited amount of functionality "out-of-the-box", you will need to choose and integrate other packages to provide any other functionality you require.

There are some frameworks which sit in the middle. They have *some* opinions and provide *some* additional functionality beyond the basics, such as logging and validation.

The style of framework you choose can be guided by the use cases I've mentioned above. It's also likely to come down to your personal preferences (or those of your team). You might not have any preferences yet, but it will help if you pay attention to each framework's style.

Support for promises and async / await

As Node.js and [ECMAScript](https://en.wikipedia.org/wiki/ECMAScript) < <https://en.wikipedia.org/wiki/ECMAScript> > have evolved, applications designed around the [callback pattern](https://nodejs.dev/learn/javascript-asynchronous-programming-and-callbacks) < <https://nodejs.dev/learn/javascript-asynchronous-programming-and-callbacks> > have faded away. Instead, we can now happily write applications with promises and `async` / `await`. This means it's important for Node.js frameworks to be promise-aware. They should allow you to write `async` route handler functions and safely handle uncaught promise rejections for you.

If a framework doesn't meet these basic requirements, it increases the risk of memory leaks or crashes in your applications. I've written an [in-depth article](https://simonplend.com/are-you-using-promises-and-async-await-safely-in-node-js/) < <https://simonplend.com/are-you-using-promises-and-async-await-safely-in-node-js/> > which explains why this matters and what to look out for.

Documentation

Firstly, does the framework have documentation?! You're likely to struggle building an application with a framework if it doesn't have documentation. If you don't see any, you should probably avoid using it.

Once you've established that a framework has documentation, try and get a sense of its quality. Not all documentation is created equal, so here are some things to consider:

- **Can you comfortably navigate and search it?** The structure of documentation really matters. It can also be a big help if there's a built-in search feature.
- **Does it make sense to you?** There might be lots of documentation, but if it's poorly written it's not going to be much help.
- **Do you think you will be able to apply it when writing your own code?** It's one thing to read and understand how to do something, and totally another to apply it yourself when building a real application.

Practical examples

Reading through endless documentation to learn how to use a framework can feel overwhelming. Once you've got a general sense of what a framework can do, it really helps if there are *practical* examples available which show you how to use it. A "hello world" example might show you the basics of how to use a framework, but it's often not much help if you want to do something more complex. Practical examples will show solutions for real problems you might need to solve. Ideally they'll also demonstrate the "best practices" for using that particular framework.

You might find practical examples in the framework documentation, or perhaps in an **examples** folder in the project's GitHub repository. Some frameworks even have full example applications on GitHub which you can browse, run and pick apart.

Seeing how the people who created a framework use it can be an excellent way of learning how to use it yourself. Instead of just grinding your way through documentation, practical examples will help you accelerate your learning curve with a new framework.

Community and ecosystem

The community which exists around a framework really matters. When shit hits the fan and you get really stuck or run into a weird bug, it's important to know where you can go for help. Check if there's a message board, Slack or Discord server for the framework you're looking at. Dip into it and get a feel for the community. Do the people there seem welcoming and supportive? Are they happy to accept contributions? The community around a framework might not matter to you on day one, but you'll definitely need it in the long term.

Framework **popularity** isn't everything, but you'll be building your whole application around it, so it's important to check that it's used by other developers. **If a framework has widespread adoption it will make it easier to find libraries which have been written to work with it (e.g. middleware or plugins).** It can be fairly straightforward to swap out one library for another in your application, but that's generally not the case with a framework.

The stars for a repository on GitHub are often cited as an indicator of popularity, but I think they're better viewed as a measure of general interest rather than actual usage. **A simple way to determine if a framework is being used is by looking at its package download counts.** They can't be taken as a measure of how many projects are *using* a framework — many frameworks are bundled with other libraries but not used — but

they can give you an idea of whether a framework is widely used. You can view weekly downloads on a package's npm page, and the tool [npm trends < https://www.npmtrends.com/>](https://www.npmtrends.com/) shows package downloads over time. It also allows you to compare packages.

Project health

When choosing a framework to build your application with, you want to be confident that the project is in good health and will continue to be maintained for the foreseeable future. Here are some indicators to look for:

- **Releases.** Some developers hold the opinion that a framework doesn't need new releases if it's already "feature complete", but security updates and bug fixes are necessary. There is always the potential that major vulnerabilities exist in a framework (or one of its dependencies). It's wise to avoid any framework which is unmaintained and no longer putting out releases.
- **Activity on issues.** Lots of issues with no recent responses from maintainers could be an indicator that the project is unmaintained. On the flip side, if there are no issues at all, it could suggest that usage of the framework is very low.
- **Pull requests.** A healthy project will typically have some recent pull request activity. Lots of old and inactive open pull requests could indicate that the project is no longer being maintained.
- **Contributors.** If there are only one or two contributors to a framework it could suggest there isn't much of a community around it. It also creates a risk around the long term maintenance of that framework if those contributors step back from the project. A healthy project will have numerous contributors, adding changes large and small.
- **Dependency graph.** The more dependencies a framework has, the larger the attack surface area. It can also make debugging issues in your applications much more difficult. You don't need to find a framework with zero dependencies, but you should have some awareness of a framework's dependency graph. The tool [npmgraph < https://npmgraph.js.org>](https://npmgraph.js.org) can provide you with an excellent overview.

The following tools can help you determine the health of a framework project:

- **Snyk Open Source Advisor < https://snyk.io/advisor/>.** This tool generates a "health score" for packages. It pulls in data from several difference sources and summarises them to help you determine the health of a project.

- **The 'Insights' tab on GitHub repositories.** This tab provides a comprehensive overview of recent project activity (releases, pull requests, issues and contributors).
- **Moiva < <https://moiva.io/> >.** This is an open source tool which provides similar data to the Snyk and GitHub tools, but it conveniently allows you to compare metrics between frameworks.

If you're looking for a list of Node.js frameworks, check out the list on [Awesome Node.js < https://github.com/sindresorhus/awesome-nodejs#web-frameworks >](https://github.com/sindresorhus/awesome-nodejs#web-frameworks). It includes popular Node.js frameworks, as well as some lesser known alternatives.

Conclusion

It turns out there are plenty of things to consider when choosing a Node.js framework. There's one last thing, however, which you might want to ask yourself when evaluating a framework: do you think you'll enjoy writing applications with it? Personally, if I don't enjoy using a framework, every time I need to work with it becomes a chore. The sweet spot is a framework which meets your requirements *and* which you enjoy using.

Choosing a Node.js framework can be tricky, but hopefully the guidelines I've shared will help you refine your own personal checklist. Good luck choosing the "best" framework for your project!

Want to create better Node.js applications?

I write articles to help you level up as a Node.js developer.

Drop your email in the box below and every couple of weeks I'll send practical advice straight to your inbox.


GET ME ON THE LIST!

I won't send you spam. Unsubscribe at any time.

Share this

 < <https://simonplend.com/guidelines-for-choosing-a-node-js-framework/?share=twitter&nb=1> >

 < <https://simonplend.com/guidelines-for-choosing-a-node-js-framework/?share=linkedin&nb=1> >

 < <https://simonplend.com/guidelines-for-choosing-a-node-js-framework/?share=reddit&nb=1> >

//

← [Workshop recording: Get Started with Fastify](https://simonplend.com/workshop-recording-get-started-with-fastify/) < <https://simonplend.com/workshop-recording-get-started-with-fastify/> >

→ [How to cancel an HTTP request in Node.js](https://simonplend.com/how-to-cancel-an-http-request-in-node.js/) < <https://simonplend.com/how-to-cancel-an-http-request-in-node.js/> >

//