Choose a JavaScript library or framework

- What are JavaScript libraries and frameworks
  - Example call to a JavaScript library
  - Example use of a JavaScript Framework
- Why use a library?
- Why should you care about library usage?
- Performance
- Security
- Accessibility
- Conventions
- Updates
- Licensing
- Community
- Documentation
- Conclusion

This article shares insights on how you might pick a library or framework to use within your web application. The discussions herein will help you to weigh the pros and cons in finding the JavaScript library or framework that is right for the business problem you're trying to solve. Understanding which pros and which cons apply in different situations is key to vetting the large number of JavaScript library choices that are available.

What are JavaScript libraries and frameworks

What is a JavaScript library? In its simplest form, a JavaScript library is prewritten code that you can call in your project's code to achieve a specific task.

This post mainly mentions "libraries". However, many of the discussions are also applicable to frameworks. Basically, the difference between the two can be summarized as follows:

- For a library, your application code calls the library code.
- For a framework, your application code is called by the framework.

The following practical examples help to illustrate the differences.

Example call to a JavaScript library

A JavaScript library performs a specific task and then returns control to your application. When you use a library, you control the application flow and choose when to call the library.

In the following example, application code imports a method from the lodash library. After the function completes, control is returned to your application.

```
import capitalize from 'lodash.capitalize';
capitalize('hello'); // Hello
```

When the `lodash.capitalize` method is executed, it calls pre-written JavaScript code that capitalizes the first character of a string.

Example use of a JavaScript Framework

A JavaScript framework is a predefined code template within which you construct your application's behavior. That is, when you use a framework, the framework controls the application flow. To use a framework, you write your custom application code, and then the framework calls your application code.

The following example shows a code snippet that uses the Preact JavaScript framework:

```
import { createElement } from 'preact';

export default function App() {
  return (
    <p class="big">Hello World!</p>
  )
}
```

In the example, notice that the framework has a lot more control over the code you write, and in some cases, the framework even takes control over when to execute your code.

Why use a library?

Using a JavaScript library can help to avoid unnecessary code repetition. Libraries can abstract away complex logic, such as date manipulation or financial calculations. A library can also help get your initial product out, rather than having to write all the code from scratch, which can take time.

Some client-side JavaScript libraries help abstract away quirks of the web platform. A library can also serve as a learning tool. For example, if you're unfamiliar with animation easing functions, the source code of a library can teach you how such easings work.

Some libraries are backed by large companies that invest time and money into keeping libraries up to date and secure. Many libraries are accompanied by extensive documentation, which offers you and your team a quick way to familiarize yourself with the library's usage.

Ultimately, using a JavaScript library saves you time.

Why should you care about library usage?

Technically, you can develop your web application from scratch, but why go to the trouble when you can use free (open source) software, or purchase a solution that, in the long run, can save time and money? There are a large number of JavaScript libraries and frameworks available, each offering a unique approach to solving problems, and each having different characteristics. For example:

- A library can be written and maintained internally rather than by a third party.
- A library can have specific legal licenses that make it suitable or unsuitable for your web application.
- A library can be outdated or unmaintained.
- A library can simplify a set of complex tasks and save you a lot of time and money.
- A library can be widely used in the community, and can be well known amongst developers.

As you might suspect, different characteristics can affect your web application in different ways. Sometimes, the decision is just not that deep, and you can safely swap out a library if you don't like it. However, sometimes a library can have a significant effect on your work and your web

application, which suggests a more informed approach could be necessary.

{% Aside 'important' %} Many of the points raised in the following performance section—and also points raised throughout this article—are applicable to client-side environments. Client-side code is run on the end user's device, such as through their web browser. {% endAside %}

There are some non client-side JavaScript environments, such as on the server (run in a cloud environment) or on a Raspberry Pi, where you may need to adjust the criteria you use to vet libraries and frameworks.

Performance

The performance effect of a JavaScript library on a client-side web application should not be ignored. A large JavaScript library can disrupt the loading performance of your page; remember, milliseconds make millions.

{% Aside %} Many JavaScript libraries are open source, and the efforts of the library maintainers cannot be overstated—open source is not easy. {% endAside %}

Consider a scenario where you use a JavaScript library for animation. Some libraries can easily add tens of kilobytes, and in some cases, even hundreds of kilobytes. JavaScript resources like this can add a significant delay to your page load as the browser needs to download, parse, compile and execute the code.

The bigger the JavaScript library, the bigger the performance effect on your users.

When evaluating or using a JavaScript library or framework, consider the following suggestions to improve performance:

- Given a large JavaScript library, consider using a smaller alternative. For example, date-fns offers a lot of functionality at a more reasonable size than some other options.
- Following on from the previous date-fns example, import only the functions that you need, such as: `import { format } from 'date-fns'`. Be sure to combine this approach with tree shaking, so that a minimal JavaScript payload is built and sent to your users.
- Use performance testing tools such as Lighthouse to observe the performance effect of using a certain JavaScript library. If a library adds a one-second delay to your page load time (don't forget to throttle your network and CPU during testing), then you may need to reevaluate your library of choice. In addition to checking page load, be sure to profile any web page behavior that invokes code from the library in question—page load performance does not tell the full story.

- If comments are welcomed by the library author, then submit your performance observations, suggestions, and even contributions to the project. This is where the open source community shines! If you decide to make a contribution, then you may need to check with your employer first.
- Use an automated bundle tracking tool, such as bundlesize, to watch for unexpectedly large updates to a library. It's common that a JavaScript library will grow over time. Feature additions, bug fixes, edge cases, and others, can all add to the file size of a library. Once you/your team have agreed to use a library, updating the library may be less of an issue and could raise little-to-no questions. This is where it's helpful to rely on automation.
- Look at your requirements for a library and evaluate whether or not the web platform offers the same functionality natively. For example, the web platform already offers a color picker, which removes the need to use a third-party JavaScript library to implement the same functionality.

Security

Using a third-party module carries some inherent security risks. A malicious package within your web application codebase can compromise the security of both your development team and your users.

Consider a library published to the NPM ecosystem. Such a package may be legitimate. However, over time, the package can be compromised.

Here are some security tips to consider when using or evaluating third-party code:

- If you use GitHub, then consider the code's security offerings, such as Dependabot. Or, consider alternative services that scan for vulnerabilities in your code, such as snyk.io.
- Consider using code-auditing services, a team of engineers who can manually audit the third-party code you're using.
- Evaluate whether you should lock your dependencies to a specific version, or commit your third-party code within your version control. This can help lock your dependency to one particular version—that is presumably deemed safe. Ironically, this can have a counter-effect within security, as you may miss out on vital updates to the library.
- Scan the project home page, or GitHub page, if one exists. Research whether outstanding security issues exist, and whether previous security issues were resolved within a reasonable timeframe.
- Third-party code that uses other third-party code can carry more risk than a library that has zero dependencies. Be mindful of this risk.

Accessibility

You may be wondering how software libraries are related to web accessibility. While a software library can be used in different environments, in the context of a client-side JavaScript-based library, web accessibility is of high importance.

A client-side JavaScript-based library (or framework, for that matter) can increase or decrease the accessibility of your website. Consider a third-party JavaScript library that adds an image slider to a page. If the image slider does not account for web accessibility, you as the web developer may overlook such an important feature, and release a product that misses critical features, such as the slider being keyboard navigable!

{% Aside %} In the previous example, it is the responsibility of the web developer building the website to make sure they use libraries that are fit for purpose. It is not the responsibility of the library author. This image slider is one particular example, but do consider other JavaScript library examples: {% endAside %}

- Does the responsive typography plugin support users who zoom in or out of the page?
- Does the file uploader plugin support file uploads from assistive devices?
- Does the animation library offer support for users who prefer reduced motion?
- Does the interactive maps plugin support keyboard-only usage?
- Does the audio player library offer an appropriate experience on screen readers?

It's reasonable to expect that some level of involvement is needed from you, the web developer, to meet such accessibility requirements. For example:

- For any missing features, you can implement such features within your codebase, even while continuing to use the library in question.
- With the support of your employer, you can contribute such a missing feature to the library, if the library author allows for such a contribution.
- You can open up a dialogue with the library author. For example, are these specific accessibility features on your roadmap? Do you agree they belong in the library?
- For popular use cases, you can explore alternative library options that are more accessible; they may exist but are harder to find.
- In the extreme case, you may need to ditch a library entirely and implement your features from scratch. This can happen when a library or framework has a degraded accessibility experience on initial use, and you need to undo a lot of what the library or framework is supposedly giving you for free.

Conventions

A software library that uses established coding conventions is easier to work with. If a library uses a coding convention that is unheard of, then it may be difficult for you and your team to work with such a library.

If a library does not follow common coding conventions (for example, a common style guide), there is not much you can do as an immediate fix. However, there are still a few options:

- Be sure to differentiate between the library source code and the API exposed to you, the library user. While the internal source code may use unfamiliar conventions, if the API (the part of the library you interact with) uses familiar conventions, then there may be nothing to worry about.
- If the library API does not follow common coding conventions, you can use a JavaScript design pattern, like the proxy pattern, to wrap and contain all interactions with the library to a single file in the codebase. Your proxy can then offer a more intuitive API to other parts of the code within your codebase.

Conventions play a big role with ease of use. A library that includes an intuitive API can save many hours', or even days' worth of people hours, when compared with a counter-intuitive API that needs a lot of experimentation to figure out.

Updates

As an example, for a fully working library that performs a few mathematical calculations, such a library may rarely need updates. In fact, a feature-complete library is a rare find in the ever-changing world of web development! However, there are times where you want the library author to be responsive and willing to make updates. New research and findings can reveal better ways of doing things, so the techniques used in libraries and frameworks are always subject to change.

When you pick a library or framework, pay attention to how updates are handled, and be aware that such decisions can affect you:

- Does the library have a sensible release schedule? For example, updates to the source code repository may happen frequently, but if such updates are not "published" or "released" accordingly, you'll find it can be a struggle to download such updates.
- Does the library release updates with a sensible software versioning scheme? A library should save you time. If you have to unexpectedly change your code every time you update the library version, this may defeat the purpose of using that library in the first place. Breaking changes are sometimes unavoidable, but in an ideal world, changes are infrequent and not forced upon library consumers.
- Does the library invest effort towards backward compatibility? Sometimes, software updates can come with breaking changes, but also provide a layer of backward compatibility. This allows the library consumer to use the latest library version with minimal changes to their code.

{% Aside %} Library authors and maintainers invest huge amounts of time supporting web developers. It's not always practical for a library author to provide updates in the ways previously discussed. For example, independent library authors who work on open source may not even receive the financial support they need to keep up with the demands of the project. {% endAside %}

Licensing

{% Aside 'warning' %} This is not legal advice. {% endAside %}

Software licensing is an important aspect of using third-party software libraries. A library author may assign a license to their library. If you are considering using the library, then their choice of license may affect you.

For example, a JavaScript library may have a software license that permits you to use it in a non-commercial environment. For a personal hobby project, this could be a great choice. If your project has a commercial element, then you may need to consider an enterprise license.

When in doubt, consider seeking professional legal advice or defer to the legal team within your company.

Community

A library or framework that has a large community of users/contributors can be beneficial, but this isn't a guarantee. In general, the more users a library or framework has, the more likely it is to benefit. Consider the following pros and cons to participation in a development community:

Pros:

- A large user base could mean a greater chance of bugs being caught early and often.
- A large active community could mean more tutorials, guides, videos, and even courses, on the library or framework in question.
- A large active community could mean more support on forums and question and answer websites, increasing the likelihood that support questions are answered.
- An engaged community can mean more external contributors to the library or framework. They can help deliver features that are otherwise not on the author's roadmap.
- When a library or framework is popular within a community, there's an increased likelihood that your peers and colleagues will have heard of, or even be familiar with, such a library or framework.

Cons:

- A project with a large and diverse user base can become bloated from constant feature additions. Bloated libraries can harm web performance.
- A project with an active and engaged community can be stressful for the authors and maintainers, and may require heavy community moderation.
- A project that grows rapidly, but does not have the appropriate support in place, can begin to exhibit signs of having a toxic community. As an example, beginner or junior web developers can be made to feel unwelcome in a certain community due to gatekeeping.

{% Aside %} The web development community is, by far, wonderful! Every community has its struggles, but you should not feel dissuaded from engaging with a particular community based on what you read here. It can be especially challenging for a new web developer to feel part of a community, especially those communities with polarizing views. You should know that this is a common challenge within the web community, and is an area in which more open conversation is needed. {% endAside %}

Documentation

No matter how simple or complex a JavaScript library or framework might be, software documentation can always help. Even very experienced developers use documentation, rather than figuring out the code themselves. Documentation clarifies the API you should use, and how you should use it.

Documentation can even provide sample code, making it easier for you to get started quickly. When you evaluate a library or framework, you can ask some of these questions:

- Does the library include documentation? If it does not, you'll need to be comfortable with figuring things out on your own.
- Is the documentation clear, easy to understand, and free of ambiguity? Many developers spend large amounts of time on documentation. It may seem small, but clarity within textual documentation can have a big effect on your productivity.
- Is the documentation completely automatically generated? Such documentation can be harder to digest and does not always provide clear guidance on how to use an API.
- Is the documentation up to date? Documentation maintenance is sometimes treated as an afterthought. If the library is updated but the documentation is not, this can lead to wasted development time.
- Is the documentation comprehensive and available in multiple formats? User guides, sample code, reference documentation, live demos, and tutorials are all valuable documentation formats that can help you be successful in using a library or framework.

Documentation cannot always be complete, and that's ok. You'll need to assess the needs of your organization, your project requirements, and the complexity of your software, and use that to determine the level of documentation you need.

Conclusion

It's normal to feel overwhelmed when picking a library or framework for the first time. Just like everything else, the more you learn and practice a task, the better you become. It might be helpful to refer to this post when you next choose a library or framework to use. You can use the headings within this post as a checklist. For example: Is this library performant? Does this library meet my business standards for web accessibility?

There are other aspects of libraries and frameworks that you may wish to consider, and that have not been heavily discussed this post:

- **Extendibility:** how easy is it to extend the library with custom logic and/or behavior?
- **Tooling:** if applicable, does the library have tooling such as code editor plugins, debugging tools, and build system plugins?
- **Architecture:** clean code is important, but is the overall architecture of the library sensible?
- **Tests:** does the project have a test suite? Does the project website use badges or indicators which the test suite is passing against the latest commit?
- **Compatibility:** does the library work well with other libraries and/or frameworks you are currently using?
- **Cost:** what is the cost of a framework? Is it open-source or available for purchase?
- **Vanity metrics:** this should be low down in your criteria list, or even ignored entirely, but you may wish to consider project "votes", social media accounts that represent the project, and/or how many open bugs/issues there are on the project page.

□□□ #2
□ □□□ □□□ 2026-10-31 23:16:15 UTC
□ □□□ □□□ 2026-10-31 23:16:17 UTC