Client-side tooling overview

In this article we provide an overview of modern web tooling, what kinds of tools are available and where you'll meet them in the lifecycle of web app development, and how to find help with individual tools.

Prerequisites:	Familiarity with the core HTML, CSS, and JavaScript languages.
Objective:	To understand what types of client-side tooling there are, and how to find tools and get help with them.

Overview of modern tooling

Writing software for the web has become more sophisticated through the ages. Although it is still entirely reasonable to write HTML, CSS, and JavaScript "by hand" there is now a wealth of tools that developers can use to speed up the process of building a web site, or app.

There are some extremely well-established tools that have become common "household names" amongst the development community, and new tools are being written and released every day to solve specific problems. You might even find yourself writing a piece of software to aid your own development process, to solve a specific problem that existing tools don't already seem to handle.

It is easy to become overwhelmed by the sheer number of tools that can be included in a single project. Equally, a single configuration file for a tool like Webpack can be hundreds of lines long, most of which are magical incantations that seem to do the job but which only a master engineer will fully understand!

From time to time, even the most experienced of web developers get stuck on a tooling problem; it is possible to waste hours attempting to get a tooling pipeline working before even touching a single line of application code. If you have found yourself struggling in the past, then don't worry — you are not alone.

In these articles, we won't answer every question about web tooling, but we will provide you with a useful starting point of understanding the fundamentals, which you can then build from. As with any complex topic, it is good to start small, and gradually work your way up to more advanced uses.

The modern tooling ecosystem

Today's modern developer tooling ecosystem is huge, so it's useful to have a broad idea of what main problems the tools are solving. If you jump on your favorite search engine and look for "front-end developer tools" you're going to hit a huge spectrum of results ranging from text editors, to browsers, to the type of pens you can use to take notes.

Though your choice of code editor is certainly a tooling choice, this series of articles will go beyond that, focusing on developer tools that help you produce web code more efficiently.

From a high-level perspective, you can put client-side tools into the following three broad categories of problems to solve:

- **Safety net** — Tools that are useful during your code development.
- **Transformation** — Tools that transform code in some way, e.g. turning an intermediate language into JavaScript that a browser can understand.
- **Post-development** — Tools that are useful after you have written your code, such as testing and deployment tools.

Let's look at each one of these in more detail.

Safety net

These are tools that make the code you write a little better.

This part of the tooling should be specific to your own development environment, though it's not uncommon for companies to have some kind of policy or pre-baked configuration available to install so that all their developers are all using the same processes.

This includes anything that makes your development process easier with respect to generating stable and reliable code. Safety net tooling should also help you either prevent mistakes or correct mistakes automatically without having to build your code from scratch each time.

A few very common safety net tool types you will find being used by developers are as follows.

Linters

Linters are tools that check through your code and tell you about any errors that are present, what error types they are, and what code lines they are present on. Often linters can be configured to not only report errors, but also report any violations of a specified style guide that your team might be using (for example code that is using the wrong number of spaces for indentation, or using template literals rather than regular string literals).

ESLint is the industry standard JavaScript linter — a highly configurable tool for catching potential syntax errors and encouraging "best practices" throughout your code. Some companies and projects have also shared their ESLint configs.

You can also find linting tools for other languages, such as csslint.

Also well-worth looking at is webhint, a configurable, open-source linter for the web that surfaces best practices including approaches to accessibility, performance, cross-browser compatibility via MDN's browser compatibility data, security, testing for PWAs, and more. It is available as a Node.js command-line tool and a VS Code extension.

Source code control

Also known as **version control systems** (VCS), **source code control** is essential for backing work up and working in teams. A typical VCS involves having a local version of the code that you make changes to. You then "push" changes to a "master" version of the code inside a remote repository stored on a server somewhere. There is usually a way of controlling and coordinating what changes are made to the "master" copy of the code, and when, so a team of developers doesn't end up overwriting each other's work all the time.

Git is the source code control system that most people use these days. It is primarily accessed via the command line but can be accessed via friendly user interfaces. With your code in a git repository, you can push it to your own server instance, or use a hosted source control website such as GitHub , GitLab , or BitBucket .

We'll be using GitHub in this module. You can find more information about it at Git and GitHub.

Code formatters

Code formatters are somewhat related to linters, except that rather than point out errors in your code, they usually tend to make sure your code is formatted correctly, according to your style rules, ideally automatically fixing errors that they find.

Prettier is a very popular example of a code formatter, which we'll make use of later on in the module.

Bundlers/packagers

These are tools that get your code ready for production, for example by "tree-shaking" to make sure only the parts of your code libraries that you are actually using are put into your final production code, or "minifying" to remove all the whitespace in your production code, making it as small as possible before it is uploaded to a server.

Parcel is a particularly clever tool that fits into this category — it can do the above tasks, but also helps to package assets like HTML, CSS, and image files into convenient bundles that you can then go on to deploy, and also adds dependencies for you automatically whenever you try to use them. It can even handle some code transformation duties for you.

Webpack is another very popular packaging tool that does similar things.

Transformation

This stage of your web app lifecycle typically allows you to code in either "future code" (such as the latest CSS or JavaScript features that might not have native support in browsers yet) or code using another language entirely, such as TypeScript . Transformation tools will then generate browser-compatible code for you, to be used in production.

Generally web development is thought of as three languages: HTML, CSS, and JavaScript, and there are transformation tools for all of these languages. Transformation offers two main benefits (amongst others):

1. The ability to write code using the latest language features and have that transformed into code that works on everyday devices. For example, you might want to write JavaScript using cutting-edge new language features, but still have your final production code work on older browsers that don't support those features. Good examples here include:
   - Babel : A JavaScript compiler that allows developers to write their code using cutting-edge JavaScript, which Babel then takes and converts into old-fashioned JavaScript that more browsers can understand. Developers can also write and publish plugins for Babel .
   - PostCSS : Does the same kind of thing as Babel, but for cutting-edge CSS features. If there isn't an equivalent way to do something using older CSS features, PostCSS will install a JavaScript polyfill to emulate the CSS effect you want.
2. The option to write your code in an entirely different language and have it transformed into a web-compatible language. For example:
   - Sass/SCSS : This CSS extension allows you to use variables, nested rules, mixins, functions, and many other features, some of which are available in native CSS (such

as variables), and some of which aren't.

- TypeScript : TypeScript is a superset of JavaScript that offers a bunch of additional features. The TypeScript compiler converts TypeScript code to JavaScript when building for production.
- Frameworks such as React , Ember , and Vue : Frameworks provide a lot of functionality for free and allow you to use it via custom syntax built on top of vanilla JavaScript. In the background, the framework's JavaScript code works hard to interpret this custom syntax and render it as a final web app.

Post development

Post-development tooling ensures that your software makes it to the web and continues to run. This includes the deployment processes, testing frameworks, auditing tools, and more.

This stage of the development process is one that you want the least amount of active interaction with so that once it is configured, it runs mostly automatically, only popping up to say hello if something has gone wrong.

Testing tools

These generally take the form of a tool that will automatically run tests against your code to make sure it is correct before you go any further (for example, when you attempt to push changes to a GitHub repo). This can include linting, but also more sophisticated procedures like unit tests, where you run part of your code, making sure they behave as they should.

Deployment tools

Deployment systems allow you to get your website published, are available for both static and dynamic sites, and commonly tend to work alongside testing systems. For example, a typical toolchain will wait for you to push changes to a remote repo, run some tests to see if the changes are OK, and then if the tests pass, automatically deploy your app to a production site.

Netlify is one of the most popular deployment tools right now, but others include Vercel and GitHub Pages .

Others

There are a number of other tool types available to use in the post-development stage, including Code Climate for gathering code quality metrics, the webhint browser extension for performing runtime analysis of cross-browser compatibility and other checks, GitHub bots for providing more powerful GitHub functionality, Updown for providing app uptime monitoring, and so many more!

Some thoughts about tooling types

There's certainly an order in which the different tooling types apply in the development lifecycle, but rest assured that you don't *have* to have all of these in place to release a website. In fact, you don't need any of these. However, including some of these tools in your process will improve your own development experience and likely improve the overall quality of your code.

It often takes some time for new developer tools to settle down in their complexity. One of the best known tools, Webpack, has a reputation for being overly complicated to work with, but in the latest major release there was a huge push to simplify common usage so the configuration required is reduced down to an absolute minimum.

There's definitely no silver bullet that will guarantee success with tools, but as your experience increases you'll find workflows that work *for you* or for your team and their projects. Once all the kinks in the process are flattened out, your tool chain should be something you can forget about and it *should* just work.

How to choose and get help with a particular tool

Most tools tend to get written and released in isolation, so although there's almost certainly help available it's never in the same place or format. It can therefore be hard to find help with using a tool, or even to choose what tool to use. The knowledge about which are the best tools to use is a bit tribal, meaning that if you aren't already in the web community, it is hard to find out exactly which ones to go for! This is one reason we wrote this series of articles, to hopefully provide that first step that is hard to find otherwise.

You'll probably need a combination of the following things:

- Experienced teachers, mentors, fellow students, or colleagues that have some experience, have solved such problems before, and can give advice.
- A useful specific place to search. General web searches for front-end developer tools are generally useless unless you already know the name of the tool you are searching for.
  - If you are using the npm package manager to manage your dependencies for example, it is a good idea to go to the npm homepage and search for the type of tool you are looking for, for example try searching for "date" if you want a date formatting utility, or "formatter" if you are searching for a general code formatter. Pay attention to the popularity, quality, and maintenance scores, and how recently the package was last updated. Also click through to the tool pages to find out how many monthly downloads a package has, and whether it has good documentation that you can use to figure out whether it does what you need it to do. Based on these criteria, the date-fns library looks like a good date formatting tool to use. You'll see this tool in action and learn more about package managers in general in Chapter 3 of this module.
  - If you are looking for a plugin to integrate tool functionality into your code editor, look at the code editor's plugins/extensions page — see Atom packages and VSCode extensions , for example. Have a look at the featured extensions on the front page, and again, try searching for the kind of extension you want (or the tool name, for example search for "ESLint" on the VSCode extensions page). When you get results, have a look at information such as how many stars or downloads the extension has, as an indicator of its quality.
- Development-related forums to ask questions on about what tools to use, for example MDN Learn Discourse , or Stack Overflow .

When you've chosen a tool to use, the first port of call should be the tool project homepage. This could be a full blown website or it might be a single readme document in a code repository. The date-fns docs for example are pretty good, complete, and easy to follow. Some documentation however can be rather technical and academic and not a good fit for your learning needs.

Instead, you might want to find some dedicated tutorials on getting started with particular types of tools. A great starting place is to search web sites like CSS Tricks , Dev , freeCodeCamp , and Smashing Magazine , as they're tailored to the web development industry.

Again, you'll probably go through several different tools as you search for the right ones for you, trying them out to see if they make sense, are well-supported, and do what you want them to do. This is fine — it is all good for learning, and the road will get smoother as you get more experience.

Summary

That rounds off our gentle introduction to the topic of client-side web tooling, from a high level. Next up we provide you with a crash course on the command line, which is where a lot of tooling is invoked from. We'll take a look at what the command line can do and then try installing and using our first tool.

In this module

- **Client-side tooling overview**
- Command line crash course
- Package management basics
- Introducing a complete toolchain
- Deploying our app

Last modified: Apr 27, 2022, by MDN contributors