Tooling

Try It Online

You don't need to install anything on your machine to try out Vue SFCs - there are online playgrounds that allow you to do so right in the browser:

- Vue SFC Playground
  - Always deployed from latest commit
  - Designed for inspecting component compilation results
- Vue + Vite on StackBlitz
  - IDE-like environment running actual Vite dev server in the browser
  - Closest to local setup

It is also recommended to use these online playgrounds to provide reproductions when reporting bugs.

Project Scaffolding

Vite

Vite is a lightweight and fast build tool with first-class Vue SFC support. It is created by Evan You, who is also the author of Vue!

To get started with Vite + Vue, simply run:

npm pnpm yarn bun

```sh
$ npm create vue@latest
```

This command will install and execute create-vue, the official Vue project scaffolding tool.

Vue compiler, check out the docs for @vitejs/plugin-vue.

Both online playgrounds mentioned above also support downloading files as a Vite project.

Vue CLI

Vue CLI is the official webpack-based toolchain for Vue. It is now in maintenance mode and we recommend starting new projects with Vite unless you rely on specific webpack-only features. Vite will provide superior developer experience in most cases.

For information on migrating from Vue CLI to Vite:

- Vue CLI -> Vite Migration Guide from VueSchool.io
- Tools / Plugins that help with auto migration

Note on In-Browser Template Compilation

When using Vue without a build step, component templates are written either directly in the page's HTML or as inlined JavaScript strings. In such cases, Vue needs to ship the template compiler to the browser in order to perform on-the-fly template compilation. On the other hand, the compiler would be unnecessary if we pre-compile the templates with a build step. To reduce client bundle size, Vue provides different "builds" optimized for different use cases.

- Build files that start with `vue.runtime.*` are **runtime-only builds**: they do not include the compiler. When using these builds, all templates must be pre-compiled via a build step.
- Build files that do not include `.runtime` are **full builds**: they include the compiler and support compiling templates directly in the browser. However, they will increase the payload by ~14kb.

Our default tooling setups use the runtime-only build since all templates in SFCs are pre-compiled. If, for some reason, you need in-browser template compilation even with a build step, you can do so by configuring the build tool to alias `vue` to `vue/dist/vue.esm-bundler.js` instead.

If you are looking for a lighter-weight alternative for no-build-step usage, check out petite-vue.

IDE Support

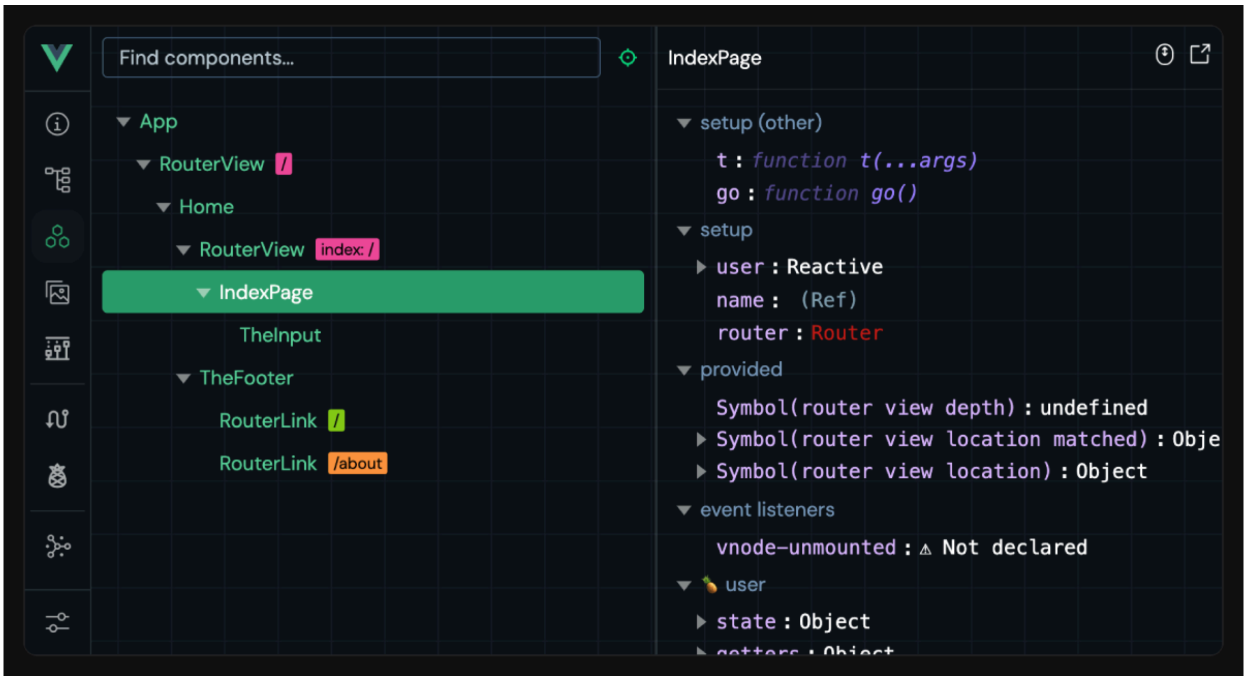
template expressions and component props.

> ⓘ TIP
>
> Vue - Official replaces Vetur, our previous official VS Code extension for Vue 2. If you have Vetur currently installed, make sure to disable it in Vue 3 projects.

- WebStorm also provides great built-in support for Vue SFCs.
- Other IDEs that support the Language Service Protocol (LSP) can also leverage Volar's core functionalities via LSP:
  - Sublime Text support via LSP-Volar.
  - vim / Neovim support via coc-volar.
  - emacs support via lsp-mode

Browser Devtools

The Vue browser devtools extension allows you to explore a Vue app's component tree, inspect the state of individual components, track state management events, and profile performance.



```
Find components...                         IndexPage
▾ App                                      ▾ setup (other)
  ▾ RouterView /                               t : function t(...args)
    ▾ Home                                     go : function go()
      ▾ RouterView index: /                ▾ setup
        ▾ IndexPage                          ▸ user : Reactive
            TheInput                           name :  (Ref)
        ▾ TheFooter                            router : Router
            RouterLink /                   ▾ provided
            RouterLink /about                  Symbol(router view depth) : undefined
                                             ▸ Symbol(router view location matched) : Obje
                                             ▸ Symbol(router view location) : Object
                                           ▾ event listeners
                                               vnode-unmounted : ⚠ Not declared
                                           ▾ user
                                             ▸ state : Object
```

- Documentation
- Chrome Extension
- Vite Plugin

TypeScript

Main article: Using Vue with TypeScript.

- Vue - Official extension provides type checking for SFCs using `<script lang="ts">` blocks, including template expressions and cross-component props validation.
- Use `vue-tsc` for performing the same type checking from the command line, or for generating `d.ts` files for SFCs.

Testing

Main article: Testing Guide.

- Cypress is recommended for E2E tests. It can also be used for component testing for Vue SFCs via the Cypress Component Test Runner.
- Vitest is a test runner created by Vue / Vite team members that focuses on speed. It is specifically designed for Vite-based applications to provide the same instant feedback loop for unit / component testing.
- Jest can be made to work with Vite via vite-jest. However, this is only recommended if you have existing Jest-based test suites that you need to migrate over to a Vite-based setup, as Vitest provides similar functionalities with a much more efficient integration.

Linting

The Vue team maintains eslint-plugin-vue, an ESLint plugin that supports SFC-specific linting rules.

Users previously using Vue CLI may be used to having linters configured via webpack loaders. However when using a Vite-based build setup, our general recommendation is:

1. `npm install -D eslint eslint-plugin-vue`, then follow `eslint-plugin-vue`'s configuration guide.

starting the dev server.

3. Run ESLint as part of the production build command, so you get full linter feedback before shipping to production.

4. (Optional) Setup tools like lint-staged to automatically lint modified files on git commit.

Formatting

- The Vue - Official VS Code extension provides formatting for Vue SFCs out of the box.
- Alternatively, Prettier provides built-in Vue SFC formatting support.

SFC Custom Block Integrations

Custom blocks are compiled into imports to the same Vue file with different request queries. It is up to the underlying build tool to handle these import requests.

- If using Vite, a custom Vite plugin should be used to transform matched custom blocks into executable JavaScript. Example
- If using Vue CLI or plain webpack, a webpack loader should be configured to transform the matched blocks. Example

---

Lower-Level Packages

`@vue/compiler-sfc`

- Docs

This package is part of the Vue core monorepo and is always published with the same version as the main `vue` package. It is included as a dependency of the main `vue` package and proxied under `vue/compiler-sfc` so you don't need to install it individually.

The package itself provides lower-level utilities for processing Vue SFCs and is only meant for tooling authors that need to support Vue SFCs in custom tools.

Always prefer using this package via the `vue/compiler-sfc` deep import since this ensures its version is in sync with the Vue runtime.

`@vitejs/plugin-vue`

- Docs

Official plugin that provides Vue SFC support in Vite.

`vue-loader`

- Docs

The official loader that provides Vue SFC support in webpack. If you are using Vue CLI, also see docs on modifying `vue-loader` options in Vue CLI.

Other Online Playgrounds

- VueUse Playground
- Vue + Vite on Repl.it
- Vue on CodeSandbox
- Vue on Codepen
- Vue on WebComponents.dev

Edit this page on GitHub

Previous
Single-File Components

Next
Routing