# Single-File Components

## Introduction

Vue Single-File Components (a.k.a. `*.vue` files, abbreviated as **SFC**) is a special file format that allows us to encapsulate the template, logic, **and** styling of a Vue component in a single file. Here's an example SFC:

```vue
<script setup>
import { ref } from 'vue'
const greeting = ref('Hello World!')
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

As we can see, Vue SFC is a natural extension of the classic trio of HTML, CSS and JavaScript. The `<template>`, `<script>`, and `<style>` blocks encapsulate and colocate the view, logic and styling of a component in the same file. The full syntax is defined in the SFC Syntax Specification.

## Why SFC

While SFCs require a build step, there are numerous benefits in return:

- Author modularized components using familiar HTML, CSS and JavaScript syntax

- Component-scoped CSS
- More ergonomic syntax when working with Composition API
- More compile-time optimizations by cross-analyzing template and script
- IDE support with auto-completion and type-checking for template expressions
- Out-of-the-box Hot-Module Replacement (HMR) support

SFC is a defining feature of Vue as a framework, and is the recommended approach for using Vue in the following scenarios:

- Single-Page Applications (SPA)
- Static Site Generation (SSG)
- Any non-trivial frontend where a build step can be justified for better development experience (DX).

That said, we do realize there are scenarios where SFCs can feel like overkill. This is why Vue can still be used via plain JavaScript without a build step. If you are just looking for enhancing largely static HTML with light interactions, you can also check out petite-vue, a 6 kB subset of Vue optimized for progressive enhancement.

---

## How It Works

Vue SFC is a framework-specific file format and must be pre-compiled by @vue/compiler-sfc into standard JavaScript and CSS. A compiled SFC is a standard JavaScript (ES) module - which means with proper build setup you can import an SFC like a module:

```js
import MyComponent from './MyComponent.vue'

export default {
  components: {
    MyComponent
  }
}
```

`<style>` tags inside SFCs are typically injected as native `<style>` tags during development to support hot updates. For production they can be extracted and merged into a single CSS file.

You can play with SFCs and explore how they are compiled in the Vue SFC Playground.

In actual projects, we typically integrate the SFC compiler with a build tool such as Vite or Vue CLI (which is based on webpack), and Vue provides official scaffolding tools to get you started with SFCs as fast as possible. Check out more details in the SFC Tooling section.

## What About Separation of Concerns?

Some users coming from a traditional web development background may have the concern that SFCs are mixing different concerns in the same place - which HTML/CSS/JS were supposed to separate!

To answer this question, it is important for us to agree that **separation of concerns is not equal to the separation of file types**. The ultimate goal of engineering principles is to improve the maintainability of codebases. Separation of concerns, when applied dogmatically as separation of file types, does not help us reach that goal in the context of increasingly complex frontend applications.

In modern UI development, we have found that instead of dividing the codebase into three huge layers that interweave with one another, it makes much more sense to divide them into loosely-coupled components and compose them. Inside a component, its template, logic, and styles are inherently coupled, and colocating them actually makes the component more cohesive and maintainable.

Note even if you don't like the idea of Single-File Components, you can still leverage its hot-reloading and pre-compilation features by separating your JavaScript and CSS into separate files using Src Imports.

Edit this page on GitHub

Previous
Suspense

Next
Tooling