Quick Start

Try Vue Online

To quickly get a taste of Vue, you can try it directly in our Playground.

If you prefer a plain HTML setup without any build steps, you can use this JSFiddle as your starting point.

If you are already familiar with Node.js and the concept of build tools, you can also try a complete build setup right within your browser on StackBlitz.

Creating a Vue Application

ⓘ Prerequisites

Familiarity with the command line
Install Node.js version 16.0 or higher

In this section we will introduce how to scaffold a Vue Single Page Application on your local machine. The created project will be using a build setup based on Vite and allow us to use Vue Single-File Components (SFCs).

Make sure you have an up-to-date version of Node.js installed and your current working directory is the one where you intend to create a project. Run the following command in your command line (without the > sign):

```
> npm create vue@latest
```

This command will install and execute create-vue, the official Vue project scaffolding tool. You will be presented with prompts for several optional features such as TypeScript and testing support:

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add an End-to-End Testing Solution? … No / Cypress / Playwright
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

If you are unsure about an option, simply choose `No` by hitting enter for now. Once the project is created, follow the instructions to install dependencies and start the dev server:

```
> cd <your-project-name>
> npm install
> npm run dev
```

You should now have your first Vue project running! Note that the example components in the generated project are written using the Composition API and `<script setup>`, rather than the Options API. Here are some additional tips:

> The recommended IDE setup is Visual Studio Code + Volar extension. If you use other editors, check out the IDE support section.
> More tooling details, including integration with backend frameworks, are discussed in the Tooling Guide.
> To learn more about the underlying build tool Vite, check out the Vite docs.
> If you choose to use TypeScript, check out the TypeScript Usage Guide.

When you are ready to ship your app to production, run the following:

```
> npm run build
```

This will create a production-ready build of your app in the project's `./dist` directory. Check out the Production Deployment Guide to learn more about shipping your app to

production.

Next Steps >

Using Vue from CDN

You can use Vue directly from a CDN via a script tag:

```html
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
```

Here we are using unpkg, but you can also use any CDN that serves npm packages, for example jsdelivr or cdnjs. Of course, you can also download this file and serve it yourself.

When using Vue from a CDN, there is no "build step" involved. This makes the setup a lot simpler, and is suitable for enhancing static HTML or integrating with a backend framework. However, you won't be able to use the Single-File Component (SFC) syntax.

Using the Global Build

The above link loads the *global build* of Vue, where all top-level APIs are exposed as properties on the global `Vue` object. Here is a full example using the global build:

```html
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

<div id="app">{{ message }}</div>

<script>
  const { createApp, ref } = Vue

  createApp({
    setup() {
      const message = ref('Hello vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

Codepen demo

> ⓘ TIP
>
> Many of the examples for Composition API throughout the guide will be using the `<script setup>` syntax, which requires build tools. If you intend to use Composition API without a build step, consult the usage of the `setup()` option.

Using the ES Module Build

Throughout the rest of the documentation, we will be primarily using ES modules syntax. Most modern browsers now support ES modules natively, so we can use Vue from a CDN via native ES modules like this:

```html
<div id="app">{{ message }}</div>

<script type="module">
  import { createApp, ref } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    setup() {
      const message = ref('Hello Vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

Notice that we are using `<script type="module">`, and the imported CDN URL is pointing to the **ES modules build** of Vue instead.

Codepen demo

Enabling Import maps

In the above example, we are importing from the full CDN URL, but in the rest of the documentation you will see code like this:

```js
import { createApp } from 'vue'
```

We can teach the browser where to locate the `vue` import by using Import Maps:

```html
<script type="importmap">
  {
    "imports": {
      "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
    }
  }
</script>

<div id="app">{{ message }}</div>

<script type="module">
  import { createApp, ref } from 'vue'

  createApp({
    setup() {
      const message = ref('Hello Vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

Codepen demo

You can also add entries for other dependencies to the import map - but make sure they point to the ES modules version of the library you intend to use.

> ⓘ **Import Maps Browser Support**
>
> Import Maps is a relatively new browser feature. Make sure to use a browser within its support range. In particular, it is only supported in Safari 16.4+.

> ⚠ **Notes on Production Use**
>
> The examples so far are using the development build of Vue - if you intend to use Vue from a CDN in production, make sure to check out the Production Deployment Guide.

Splitting Up the Modules

As we dive deeper into the guide, we may need to split our code into separate JavaScript files so that they are easier to manage. For example:

```html
<!-- index.html -->
<div id="app"></div>

<script type="module">
  import { createApp } from 'vue'
  import MyComponent from './my-component.js'

  createApp(MyComponent).mount('#app')
</script>
```

```js
// my-component.js
import { ref } from 'vue'
export default {
  setup() {
    const count = ref(0)
    return { count }
  },
  template: `<div>count is {{ count }}</div>`
}
```

If you directly open the above `index.html` in your browser, you will find that it throws an error because ES modules cannot work over the `file://` protocol, which is the protocol the browser uses when you open a local file.

Due to security reasons, ES modules can only work over the `http://` protocol, which is what the browsers use when opening pages on the web. In order for ES modules to work on our local machine, we need to serve the `index.html` over the `http://` protocol, with a local HTTP server.

To start a local HTTP server, first make sure you have Node.js installed, then run `npx serve` from the command line in the same directory where your HTML file is. You can also use any other HTTP server that can serve static files with the correct MIME types.

You may have noticed that the imported component's template is inlined as a JavaScript string. If you are using VSCode, you can install the es6-string-html extension and prefix the strings with a `/*html*/` comment to get syntax highlighting for them.

Next Steps

If you skipped the Introduction, we strongly recommend reading it before moving on to the rest of the documentation.

Continue with the Guide

The guide walks you through every aspect of the framework in full detail.

Try the Tutorial

For those who prefer learning things hands-on.

Check out the Examples

Explore examples of core features and common UI tasks.

Edit this page on GitHub