

```
---
title: 'useAppConfig'
description: 'Access the reactive app config defined in the project.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/config.ts
    size: xs
---
```

Usage

```
``ts
const appConfig = useAppConfig()

console.log(appConfig)
``

:read-more{to="/docs/guide/directory-structure/app-config"}
```

```
---
title: 'Nuxt Guide'
titleTemplate: '%s'
description: 'Learn how Nuxt works with in-depth guides.'
navigation: false
surround: false
---
```

```
:::card-group{class="sm:grid-cols-1"}
:::card{icon="i-ph-medal-duotone" title="Key Concepts" to="/docs/guide/concepts"}
Discover the main concepts behind Nuxt, from auto-import, hybrid rendering to its TypeScript support.
::
:::card{icon="i-ph-folders-duotone" title="Directory Structure" to="/docs/guide/directory-structure"}
Learn about Nuxt directory structure and what benefits each directory or file offers.
::
:::card{icon="i-ph-star-duotone" title="Going Further" to="/docs/guide/going-further"}
Master Nuxt with advanced concepts like experimental features, hooks, modules, and more.
::
:::card{icon="i-ph-book-open-duotone" title="Recipes" to="/docs/guide/recipes"}
Find solutions to common problems and learn how to implement them in your Nuxt project.
::
::
```

```
---
title: ".output"
description: "Nuxt creates the .output/ directory when building your application for production."
head.title: ".output/"
navigation.icon: i-ph-folder-duotone
---

::important
This directory should be added to your [`.gitignore`](/docs/guide/directory-structure/gitignore) file to avoid pushing the
build output to your repository.
::

Use this directory to deploy your Nuxt application to production.

:read-more{to="/docs/getting-started/deployment"}

::warning
You should not touch any files inside since the whole directory will be re-created when running [`nuxt build`]
(/docs/api/commands/build).
::
```

```

---
title: '<ClientOnly>'
description: Render components only in client-side with the <ClientOnly> component.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/client-only.ts
    size: xs
---

```

The `<ClientOnly>` component is used for purposely rendering a component only on client side.

```

::note
The content of the default slot will be tree-shaken out of the server build. (This does mean that any CSS used by components
within it may not be inlined when rendering the initial HTML.)
::

```

Props

```

- `placeholderTag` | `fallbackTag`: specify a tag to be rendered server-side.
- `placeholder` | `fallback`: specify a content to be rendered server-side.

```

```

```vue
<template>
 <div>
 <Sidebar />
 <!-- The <Comment> component will only be rendered on client-side -->
 <ClientOnly fallback-tag="span" fallback="Loading comments...">
 <Comment />
 </ClientOnly>
 </div>
</template>
```

```

Slots

```

- `#fallback`: specify a content to be rendered on the server and displayed until `<ClientOnly>` is mounted in the browser.

```

```

```vue [pages/example.vue]
<template>
 <div>
 <Sidebar />
 <!-- This renders the "span" element on the server side -->
 <ClientOnly fallbackTag="span">
 <!-- this component will only be rendered on client side -->
 <Comments />
 <template #fallback>
 <!-- this will be rendered on server side -->
 <p>Loading comments...</p>
 </template>
 </ClientOnly>
 </div>
</template>
```

```

```
---
title: '<DevOnly>'
description: Render components only during development with the <DevOnly> component.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/dev-only.ts
    size: xs
---
```

Nuxt provides the ``<DevOnly>`` component to render a component only during development.

The content will not be included in production builds.

```
```vue [pages/example.vue]
<template>
 <div>
 <Sidebar />
 <DevOnly>
 <!-- this component will only be rendered during development -->
 <LazyDebugBar />

 <!-- if you ever require to have a replacement during production -->
 <!-- be sure to test these using `nuxt preview` -->
 <template #fallback>
 <div><!-- empty div for flex.justify-between --></div>
 </template>
 </DevOnly>
 </div>
</template>
```
```

Slots

- ``#fallback``: if you ever require to have a replacement during production.

```
```vue
<template>
 <div>
 <Sidebar />
 <DevOnly>
 <!-- this component will only be rendered during development -->
 <LazyDebugBar />
 <!-- be sure to test these using `nuxt preview` -->
 <template #fallback>
 <div><!-- empty div for flex.justify-between --></div>
 </template>
 </DevOnly>
 </div>
</template>
```
```

```
---
title: ".nuxt"
description: "Nuxt uses the .nuxt/ directory in development to generate your Vue application."
head.title: ".nuxt/"
navigation.icon: i-ph-folder-duotone
---
```

::important

This directory should be added to your [.gitignore](/docs/guide/directory-structure/gitignore) file to avoid pushing the dev build output to your repository.

::

This directory is interesting if you want to learn more about the files Nuxt generates based on your directory structure.

Nuxt also provides a Virtual File System (VFS) for modules to add templates to this directory without writing them to disk.

You can explore the generated files by opening the [Nuxt DevTools](https://devtools.nuxt.com) in development mode and navigating to the **Virtual Files** tab.

::warning

You should not touch any files inside since the whole directory will be re-created when running [nuxt dev] (/docs/api/commands/dev).

::

```

---
title: Introduction
description: Nuxt's goal is to make web development intuitive and performant with a great Developer Experience in mind.
navigation:
  icon: i-ph-info-duotone
---

Nuxt is a free and [open-source framework](https://github.com/nuxt/nuxt) with an intuitive and extendable way to create type-safe, performant and production-grade full-stack web applications and websites with [Vue.js](https://vuejs.org).

We made everything so you can start writing `.vue` files from the beginning while enjoying hot module replacement in development and a performant application in production with server-side rendering by default.

Nuxt has no vendor lock-in, allowing you to deploy your application everywhere, even on the edge(/blog/nuxt-on-the-edge).

::tip
If you want to play around with Nuxt in your browser, you can [try it out in one of our online sandboxes](/docs/getting-started/installation#play-online).
::

## Automation and Conventions

Nuxt uses conventions and an opinionated directory structure to automate repetitive tasks and allow developers to focus on pushing features. The configuration file can still customize and override its default behaviors.

- File-based routing: define routes based on the structure of your [pages/` directory](/docs/guide/directory-structure/pages). This can make it easier to organize your application and avoid the need for manual route configuration.
- Code splitting: Nuxt automatically splits your code into smaller chunks, which can help reduce the initial load time of your application.
- Server-side rendering out of the box: Nuxt comes with built-in SSR capabilities, so you don't have to set up a separate server yourself.
- Auto-imports: write Vue composables and components in their respective directories and use them without having to import them with the benefits of tree-shaking and optimized JS bundles.
- Data-fetching utilities: Nuxt provides composables to handle SSR-compatible data fetching as well as different strategies.
- Zero-config TypeScript support: write type-safe code without having to learn TypeScript with our auto-generated types and `tsconfig.json`
- Configured build tools: we use [Vite](https://vitejs.dev) by default to support hot module replacement (HMR) in development and bundling your code for production with best-practices baked-in.

Nuxt takes care of these and provides both frontend and backend functionality so you can focus on what matters: creating your web application.

## Server-Side Rendering

Nuxt comes with built-in server-side rendering (SSR) capabilities by default, without having to configure a server yourself, which has many benefits for web applications:

- Faster initial page load time: Nuxt sends a fully rendered HTML page to the browser, which can be displayed immediately. This can provide a faster perceived page load time and a better user experience (UX), especially on slower networks or devices.
- Improved SEO: search engines can better index SSR pages because the HTML content is available immediately, rather than requiring JavaScript to render the content on the client-side.
- Better performance on low-powered devices: it reduces the amount of JavaScript that needs to be downloaded and executed on the client-side, which can be beneficial for low-powered devices that may struggle with processing heavy JavaScript applications.
- Better accessibility: the content is immediately available on the initial page load, improving accessibility for users who rely on screen readers or other assistive technologies.
- Easier caching: pages can be cached on the server-side, which can further improve performance by reducing the amount of time it takes to generate and send the content to the client.

Overall, server-side rendering can provide a faster and more efficient user experience, as well as improve search engine optimization and accessibility.

As Nuxt is a versatile framework, it gives you the possibility to statically render your whole application to a static hosting with `nuxt generate`, disable SSR globally with the `ssr: false` option or leverage hybrid rendering by setting up the `routeRules` option.

:read-more{title="Nuxt rendering modes" to="/docs/guide/concepts/rendering"}

### Server engine

The Nuxt server engine [Nitro](https://nitro.unjs.io) unlocks new full-stack capabilities.

In development, it uses Rollup and Node.js workers for your server code and context isolation. It also generates your server API by reading files in `server/api/` and server middleware from `server/middleware/`.

In production, Nitro builds your app and server into one universal `.output` directory. This output is light: minified and removed from any Node.js modules (except polyfills). You can deploy this output on any system supporting JavaScript, from Node.js, Serverless, Workers, Edge-side rendering or purely static.

:read-more{title="Nuxt server engine" to="/docs/guide/concepts/server-engine"}

```

Production-ready

A Nuxt application can be deployed on a Node or Deno server, pre-rendered to be hosted in static environments, or deployed to serverless and edge providers.

:read-more{title="Deployment section" to="/docs/getting-started/deployment"}

Modular

A module system allows to extend Nuxt with custom features and integrations with third-party services.

:read-more{title="Nuxt Modules Concept" to="/docs/guide/concepts/modules"}

Architecture

Nuxt is composed of different [core packages](https://github.com/nuxt/nuxt/tree/main/packages):

- Core Engine: [nuxt](https://github.com/nuxt/nuxt/tree/main/packages/nuxt)
- Bundlers: [@nuxt/vite-builder](https://github.com/nuxt/nuxt/tree/main/packages/vite) and [@nuxt/webpack-builder](https://github.com/nuxt/nuxt/tree/main/packages/webpack)
- Command line interface: [nuxi](https://github.com/nuxt/nuxt/tree/main/packages/nuxi)
- Server engine: [nitro](https://github.com/unjs/nitro)
- Development kit: [@nuxt/kit](https://github.com/nuxt/nuxt/tree/main/packages/kit)

We recommend reading each concept to have a full vision of Nuxt capabilities and the scope of each package.


```
---
title: "onPrehydrate"
description: "Use onPrehydrate to run a callback on the client immediately before
Nuxt hydrates the page."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
    size: xs
---
```

```
::important
This composable is available in Nuxt v3.12+.
::
```

`onPrehydrate` is a composable lifecycle hook that allows you to run a callback on the client immediately before Nuxt hydrates the page.

```
::note
This is an advanced utility and should be used with care. For example, [nuxt-time](https://github.com/danielroe/nuxt-time/pull/251) and [@nuxtjs/color-mode](https://github.com/nuxt-modules/color-mode/blob/main/src/script.js) manipulate the DOM to avoid hydration mismatches.
::
```

Usage

`onPrehydrate` can be called directly in the setup function of a Vue component (for example, in `<script setup>`), or in a plugin. It will only have an effect when it is called on the server, and it will not be included in your client build.

Parameters

- `callback`: A function that will be stringified and inlined in the HTML. It should not have any external dependencies (such as auto-imports) or refer to variables defined outside the callback. The callback will run before Nuxt runtime initializes so it should not rely on the Nuxt or Vue context.

Example

```
```vue twoslash [app.vue]
<script setup lang="ts">
declare const window: Window
// ---cut---
// onPrehydrate is guaranteed to run before Nuxt hydrates
onPrehydrate(() => {
 console.log(window)
})

// As long as it only has one root node, you can access the element
onPrehydrate((el) => {
 console.log(el.outerHTML)
 // <div data-v-inspector="app.vue:15:3" data-prehydrate-id=":b3qlvSiBeH:"> Hi there </div>
})

// For _very_ advanced use cases (such as not having a single root node) you
// can access/set `data-prehydrate-id` yourself
const prehydrateId = onPrehydrate((el) => {})
</script>

<template>
 <div>
 Hi there
 </div>
</template>
```
```

```
---
title: 'Deployment'
description: Learn how to deploy your Nuxt application to any hosting provider.
navigation.icon: i-ph-cloud-duotone
---
```

A Nuxt application can be deployed on a Node.js server, pre-rendered for static hosting, or deployed to serverless or edge (CDN) environments.

```
::tip
If you are looking for a list of cloud providers that support Nuxt 3, see the [Hosting providers](/deploy) section.
::
```

Node.js Server

Discover the Node.js server preset with Nitro to deploy on any Node hosting.

- **Default output format** if none is specified or auto-detected

- Loads only the required chunks to render the request for optimal cold start timing

- Useful for deploying Nuxt apps to any Node.js hosting

Entry Point

When running `nuxt build` with the Node server preset, the result will be an entry point that launches a ready-to-run Node server.

```
```bash [Terminal]
node .output/server/index.mjs
```
```

This will launch your production Nuxt server that listens on port 3000 by default.

It respects the following runtime environment variables:

- `NITRO_PORT` or `PORT` (defaults to `3000`)
- `NITRO_HOST` or `HOST` (defaults to `'0.0.0.0'`)
- `NITRO_SSL_CERT` and `NITRO_SSL_KEY` - if both are present, this will launch the server in HTTPS mode. In the vast majority of cases, this should not be used other than for testing, and the Nitro server should be run behind a reverse proxy like nginx or Cloudflare which terminates SSL.

PM2

[PM2](https://pm2.keymetrics.io/) (Process Manager 2) is a fast and easy solution for hosting your Nuxt application on your server or VM.

To use `pm2`, use an `ecosystem.config.cjs`:

```
```ts [ecosystem.config.cjs]
module.exports = {
 apps: [
 {
 name: 'NuxtAppName',
 port: '3000',
 exec_mode: 'cluster',
 instances: 'max',
 script: './.output/server/index.mjs'
 }
]
}
```
```

Cluster Mode

You can use `NITRO_PRESET=node_cluster` in order to leverage multi-process performance using Node.js [cluster](https://nodejs.org/dist/latest/docs/api/cluster.html) module.

By default, the workload gets distributed to the workers with the round robin strategy.

Learn More

`:read-more{to="https://nitro.unjs.io/deploy/node" title="the Nitro documentation for node-server preset"}`

Static Hosting

There are two ways to deploy a Nuxt application to any static hosting services:

- Static site generation (SSG) with `ssr: true` pre-renders routes of your application at build time. (This is the default behavior when running `nuxi generate`.) It will also generate `/200.html` and `/404.html` single-page app fallback pages, which can render dynamic routes or 404 errors on the client (though you may need to configure this on your static host).
- Alternatively, you can prerender your site with `ssr: false` (static single-page app). This will produce HTML pages with an empty `

:read-more{title="Nuxt prerendering" to="/docs/getting-started/prerendering"}

Client-side Only Rendering

If you don't want to pre-render your routes, another way of using static hosting is to set the `ssr` property to `false` in the `nuxt.config` file. The `nuxi generate` command will then output an `output/public/index.html` entrypoint and JavaScript bundles like a classic client-side Vue.js application.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 ssr: false
})
```
```

Hosting Providers

Nuxt can be deployed to several cloud providers with a minimal amount of configuration:

:read-more{to="/deploy"}

Presets

In addition to Node.js servers and static hosting services, a Nuxt project can be deployed with several well-tested presets and minimal amount of configuration.

You can explicitly set the desired preset in the [nuxt.config.ts](/docs/guide/directory-structure/nuxt-config) file:

```
```:js twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 nitro: {
 preset: 'node-server'
 }
})
```
```

... or use the `NITRO_PRESET` environment variable when running `nuxt build`:

```
```:bash [Terminal]
NITRO_PRESET=node-server nuxt build
```
```

ðŸ”Ž Check [the Nitro deployment](https://nitro.unjs.io/deploy) for all possible deployment presets and providers.

CDN Proxy

In most cases, Nuxt can work with third-party content that is not generated or created by Nuxt itself. But sometimes such content can cause problems, especially Cloudflare's "Minification and Security Options".

Accordingly, you should make sure that the following options are unchecked / disabled in Cloudflare. Otherwise, unnecessary re-rendering or hydration errors could impact your production application.

1. Speed > Optimization > Content Optimization > Auto Minify: Uncheck JavaScript, CSS and HTML
2. Speed > Optimization > Content Optimization > Disable "Rocket Loader",
3. Speed > Optimization > Image Optimization > Disable "Mirage"
4. Scrape Shield > Disable "Email Address Obfuscation"

With these settings, you can be sure that Cloudflare won't inject scripts into your Nuxt application that may cause unwanted side effects.

::tip
Their location on the Cloudflare dashboard sometimes changes so don't hesitate to look around.
::

```
---
title: 'Vue.js Development'
description: "Nuxt uses Vue.js and adds features such as component auto-imports, file-based routing and composables for an SSR-friendly usage."
---
```

Nuxt integrates Vue 3, the new major release of Vue that enables new patterns for Nuxt users.

```
::note
While an in-depth knowledge of Vue is not required to use Nuxt, we recommend that you read the documentation and go through
some of the examples on [vuejs.org](https://vuejs.org).
::
```

Nuxt has always used Vue as a frontend framework.

We chose to build Nuxt on top of Vue for these reasons:

- The reactivity model of Vue, where a change in data automatically triggers a change in the interface.
- The component-based templating, while keeping HTML as the common language of the web, enables intuitive patterns to keep your interface consistent, yet powerful.
- From small projects to large web applications, Vue keeps performing well at scale to ensure that your application keeps delivering value to your users.

Vue with Nuxt

Single File Components

[Vue's single-file components](https://v3.vuejs.org/guide/single-file-component.html) (SFC or ``*.vue` files) encapsulate the markup (`<template>`), logic (`<script>`) and styling (`<style>`) of a Vue component. Nuxt provides a zero-config experience for SFCs with [Hot Module Replacement](https://vitejs.dev/guide/features.html#hot-module-replacement) that offers a seamless developer experience.

Auto-imports

Every Vue component created in the [`components/`](/docs/guide/directory-structure/components) directory of a Nuxt project will be available in your project without having to import it. If a component is not used anywhere, your production's code will not include it.

```
:read-more{to="/docs/guide/concepts/auto-imports"}
```

Vue Router

Most applications need multiple pages and a way to navigate between them. This is called **routing**. Nuxt uses a [`pages/`](/docs/guide/directory-structure/pages) directory and naming conventions to directly create routes mapped to your files using the official [Vue Router library](https://router.vuejs.org).

```
:read-more{to="/docs/getting-started/routing"}
```

```
:link-example{to="/docs/examples/features/auto-imports"}
```

Differences with Nuxt 2 / Vue 2

Nuxt 3+ is based on Vue 3. The new major Vue version introduces several changes that Nuxt takes advantage of:

- Better performance
- Composition API
- TypeScript support

Faster Rendering

The Vue Virtual DOM (VDOM) has been rewritten from the ground up and allows for better rendering performance. On top of that, when working with compiled Single-File Components, the Vue compiler can further optimize them at build time by separating static and dynamic markup.

This results in faster first rendering (component creation) and updates, and less memory usage. In Nuxt 3, it enables faster server-side rendering as well.

Smaller Bundle

With Vue 3 and Nuxt 3, a focus has been put on bundle size reduction. With version 3, most of Vue's functionality, including template directives and built-in components, is tree-shakable. Your production bundle will not include them if you don't use them.

This way, a minimal Vue 3 application can be reduced to 12 kb gzipped.

Composition API

The only way to provide data and logic to components in Vue 2 was through the Options API, which allows you to return data and methods to a template with pre-defined properties like `data` and `methods`:

```
``vue twoslash
<script>
export default {
```

```

    data() {
      return {
        count: 0
      }
    },
    methods: {
      increment(){
        this.count++
      }
    }
  }
}
</script>
```

```

The [Composition API](https://vuejs.org/guide/extras/composition-api-faq.html) introduced in Vue 3 is not a replacement of the Options API, but it enables better logic reuse throughout an application, and is a more natural way to group code by concern in complex components.

Used with the `setup` keyword in the `

```

title: Auto-imports
description: "Nuxt auto-imports components, composables, helper functions and Vue APIs."

```

Nuxt auto-imports components, composables and [Vue.js APIs](https://vuejs.org/api) to use across your application without explicitly importing them.

```
``vue twoslash [app.vue]
<script setup lang="ts">
const count = ref(1) // ref is auto-imported
</script>
````
```

Thanks to its opinionated directory structure, Nuxt can auto-import your [`components/`](/docs/guide/directory-structure/components), [`composables/`](/docs/guide/directory-structure/composables) and [`utils/`](/docs/guide/directory-structure/utils).

Contrary to a classic global declaration, Nuxt preserves typings, IDEs completions and hints, and **only** includes what is used in your production code.

```
::note
In the docs, every function that is not explicitly imported is auto-imported by Nuxt and can be used as-is in your code. You can find a reference for auto-imported components, composables and utilities in the [API section](/docs/api).
::
```

```
::note
In the [server/](/docs/guide/directory-structure/server) directory, Nuxt auto-imports exported functions and variables from server/utils/.
::
```

```
::note
You can also auto-import functions exported from custom folders or third-party packages by configuring the [imports/](/docs/api/nuxt-config#imports) section of your nuxt.config file.
::
```

Built-in Auto-imports

Nuxt auto-imports functions and composables to perform [data fetching](/docs/getting-started/data-fetching), get access to the [app context](/docs/api/composables/use-nuxt-app) and [runtime config](/docs/guide/going-further/runtime-config), manage [state](/docs/getting-started/state-management) or define components and plugins.

```
``vue twoslash
<script setup lang="ts">
/* useAsyncData() and $fetch() are auto-imported */
const { data, refresh, status } = await useFetch('/api/hello')
</script>
````
```

Vue 3 exposes Reactivity APIs like `ref` or `computed`, as well as lifecycle hooks and helpers that are auto-imported by Nuxt.

```
``vue twoslash
<script setup lang="ts">
/* ref() and computed() are auto-imported */
const count = ref(1)
const double = computed(() => count.value * 2)
</script>
````
```

Vue and Nuxt composables

<!-- TODO: move to separate page with https://github.com/nuxt/nuxt/issues/14723 and add more information -->

When you are using the built-in Composition API composables provided by Vue and Nuxt, be aware that many of them rely on being called in the right `_context_`.

During a component lifecycle, Vue tracks the temporary instance of the current component (and similarly, Nuxt tracks a temporary instance of `nuxtApp`) via a global variable, and then unsets it in same tick. This is essential when server rendering, both to avoid cross-request state pollution (leaking a shared reference between two users) and to avoid leakage between different components.

That means that (with very few exceptions) you cannot use them outside a Nuxt plugin, Nuxt route middleware or Vue setup function. On top of that, you must use them synchronously - that is, you cannot use `await` before calling a composable, except within `<script setup>` blocks, within the setup function of a component declared with `defineNuxtComponent`, in `defineNuxtPlugin` or in `defineNuxtRouteMiddleware`, where we perform a transform to keep the synchronous context even after the `await`.

If you get an error message like `Nuxt instance is unavailable` then it probably means you are calling a Nuxt composable in the wrong place in the Vue or Nuxt lifecycle.

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=ofuKRZLt0dY" target="_blank"}
Watch a video from Alexander Lichter about handling async code in composables and fixing Nuxt instance is unavailable in
```

```

your app.
::

::read-more{to="/docs/guide/going-further/experimental-features#asynccontext" icon="i-ph-star-duotone"}
Checkout the `asyncContext` experimental feature to use Nuxt composables in async functions.
::

::read-more{to="https://github.com/nuxt/nuxt/issues/14269#issuecomment-1397352832" target="_blank"}
See the full explanation in this GitHub comment.
::

**Example of breaking code:**

``ts twoslash [composables/example.ts]
// trying to access runtime config outside a composable
const config = useRuntimeConfig()

export const useMyComposable = () => {
  // accessing runtime config here
}
...

**Example of working code:**

``ts twoslash [composables/example.ts]
export const useMyComposable = () => {
  // Because your composable is called in the right place in the lifecycle,
  // useRuntimeConfig will work here
  const config = useRuntimeConfig()

  // ...
}
...

## Directory-based Auto-imports

Nuxt directly auto-imports files created in defined directories:

- `components/` for [Vue components](/docs/guide/directory-structure/components).
- `composables/` for [Vue composables](/docs/guide/directory-structure/composables).
- `utils/` for helper functions and other utilities.

:link-example{to="/docs/examples/features/auto-imports"}

::warning
**Auto-imported `ref` and `computed` won't be unwrapped in a component ``.** :br
This is due to how Vue works with refs that aren't top-level to the template. You can read more about it [in the Vue documentation](https://vuejs.org/guide/essentials/reactivity-fundamentals.html#caveat-when-unwrapping-in-templates).
::

### Explicit Imports

Nuxt exposes every auto-import with the `#imports` alias that can be used to make the import explicit if needed:

<!-- TODO:twoslash: Twoslash does not support tsconfig paths yet -->

``vue
<script setup lang="ts">
import { ref, computed } from '#imports'

const count = ref(1)
const double = computed(() => count.value * 2)
</script>
...

### Disabling Auto-imports

If you want to disable auto-importing composables and utilities, you can set `imports.autoImport` to `false` in the `nuxt.config` file.

``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  imports: {
    autoImport: false
  }
})
...

This will disable auto-imports completely but it's still possible to use [explicit imports](#explicit-imports) from `#imports`.

## Auto-imported Components

```

Nuxt also automatically imports components from your `~/components` directory, although this is configured separately from auto-importing composables and utility functions.

```
:read-more{to="/docs/guide/directory-structure/components"}
```

To disable auto-importing components from your own `~/components` directory, you can set `components.dirs` to an empty array (though note that this will not affect components added by modules).

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 components: {
 dirs: []
 }
})
```:
```

Auto-import from third-party packages

Nuxt also allows auto-importing from third-party packages.

::tip
If you are using the Nuxt module for that package, it is likely that the module has already configured auto-imports for that package.
::

For example, you could enable the auto-import of the `useI18n` composable from the `vue-i18n` package like this:

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 imports: {
 presets: [
 {
 from: 'vue-i18n',
 imports: ['useI18n']
 }
]
 }
})
```:
```

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=FT2LQJ2NvVI" target="_blank"}
Watch a video from Alexander Lichter on how to easily set up custom auto imports.
::


```

---
title: "$fetch"
description: Nuxt uses ofetch to expose globally the $fetch helper for making HTTP requests.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/entry.ts
    size: xs
---

```

Nuxt uses [ofetch](https://github.com/unjs/ofetch) to expose globally the ` \$fetch ` helper for making HTTP requests within your Vue app or API routes.

```

::tip{icon="i-ph-rocket-launch-duotone" color="gray"}
During server-side rendering, calling ` $fetch ` to fetch your internal [API routes](/docs/guide/directory-structure/server) will directly call the relevant function (emulating the request), saving an additional API call.
::

```

```

::note{color="blue" icon="i-ph-info-duotone"}
Using ` $fetch ` in components without wrapping it with [ `useAsyncData` ](/docs/api/composables/use-async-data) causes fetching the data twice: initially on the server, then again on the client-side during hydration, because ` $fetch ` does not transfer state from the server to the client. Thus, the fetch will be executed on both sides because the client has to get the data again.
::

```

We recommend to use [`useFetch`](/docs/api/composables/use-fetch) or [`useAsyncData`](/docs/api/composables/use-async-data) + ` \$fetch ` to prevent double data fetching when fetching the component data.

```

```vue [app.vue]
<script setup lang="ts">
// During SSR data is fetched twice, once on the server and once on the client.
const dataTwice = await $fetch('/api/item')

// During SSR data is fetched only on the server side and transferred to the client.
const { data } = await useAsyncData('item', () => $fetch('/api/item'))

// You can also useFetch as shortcut of useAsyncData + $fetch
const { data } = await useFetch('/api/item')
</script>
```

```

[:read-more{to="/docs/getting-started/data-fetching"}](/docs/getting-started/data-fetching)

You can use ` \$fetch ` in any methods that are executed only on client-side.

```

```vue [pages/contact.vue]
<script setup lang="ts">
function contactForm() {
 $fetch('/api/contact', {
 method: 'POST',
 body: { hello: 'world ' }
 })
}
</script>

<template>
 <button @click="contactForm">Contact</button>
</template>
```

```

```

::tip
` $fetch ` is the preferred way to make HTTP calls in Nuxt instead of [ @nuxt/http ](https://github.com/nuxt/http) and [ @nuxtjs/axios ](https://github.com/nuxt-community/axios-module) that are made for Nuxt 2.
::

```

```

::note
If you use ` $fetch ` to call an (external) HTTPS URL with a self-signed certificate in development, you will need to set ` NODE_TLS_REJECT_UNAUTHORIZED=0 ` in your environment.
::

```

```

---
title: "<NuxtClientFallback>"
description: "Nuxt provides the <NuxtClientFallback> component to render its content on the client if any of its children trigger an error in SSR"
links:
  - label: Source (client)
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/client-fallback.client.ts
    size: xs
  - label: Source (server)
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/client-fallback.server.ts
    size: xs
---

```

Nuxt provides the `<NuxtClientFallback>` component to render its content on the client if any of its children trigger an error in SSR.

```

:::note{to="/docs/guide/going-further/experimental-features#clientfallback"}
This component is experimental and in order to use it you must enable the `experimental.clientFallback` option in your `nuxt.config`.
::

```

```

```vue [pages/example.vue]
<template>
 <div>
 <Sidebar />
 <!-- this component will be rendered on client-side -->
 <NuxtClientFallback fallback-tag="span">
 <Comments />
 <BrokeInSSR />
 </NuxtClientFallback>
 </div>
</template>
```

```

Events

- `@ssr-error`: Event emitted when a child triggers an error in SSR. Note that this will only be triggered on the server.

```

```vue
<template>
 <NuxtClientFallback @ssr-error="logSomeError">
 <!-- ... -->
 </NuxtClientFallback>
</template>
```

```

Props

- `placeholderTag` | `fallbackTag`: Specify a fallback tag to be rendered if the slot fails to render on the server.
 - **type**: `string`
 - **default**: `div`
- `placeholder` | `fallback`: Specify fallback content to be rendered if the slot fails to render.
 - **type**: `string`
- `keepFallback`: Keep the fallback content if it failed to render server-side.
 - **type**: `boolean`
 - **default**: `false`

```

```vue
<template>
 <!-- render Hello world server-side if the default slot fails to render -->
 <NuxtClientFallback fallback-tag="span" fallback="Hello world">
 <BrokeInSsr />
 </NuxtClientFallback>
</template>
```

```

Slots

- `#fallback`: specify content to be displayed server-side if the slot fails to render.

```

```vue
<template>
 <NuxtClientFallback>
 <!-- ... -->
 <template #fallback>
 <!-- this will be rendered on server side if the default slot fails to render in ssr -->
 <p>Hello world</p>
 </template>
 </NuxtClientFallback>
</template>
```

```



```
---
title: "assets"
description: "The assets/ directory is used to add all the website's assets that the build tool will process."
head.title: "assets/"
navigation.icon: i-ph-folder-duotone
---
```

The directory usually contains the following types of files:

- Stylesheets (CSS, SASS, etc.)
- Fonts
- Images that won't be served from the [`public/`](/docs/guide/directory-structure/public) directory.

If you want to serve assets from the server, we recommend taking a look at the [`public/`](/docs/guide/directory-structure/public) directory.

```
:read-more{to="/docs/getting-started/assets"}
```

```
---
title: 'abortNavigation'
description: 'abortNavigation is a helper function that prevents navigation from taking place and throws an error if one is set as a parameter.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
    size: xs
---
```

```
:::warning
`abortNavigation` is only usable inside a [route middleware handler](/docs/guide/directory-structure/middleware).
:::
```

Type

```
```ts
abortNavigation(err?: Error | string): false
```
```

Parameters

```
### `err`

- **Type**: [Error](https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Global_Objects/Error) | string

Optional error to be thrown by `abortNavigation`.
```

Examples

The example below shows how you can use `abortNavigation` in a route middleware to prevent unauthorized route access:

```
```ts [middleware/auth.ts]
export default defineNuxtRouteMiddleware((to, from) => {
 const user = useState('user')

 if (!user.value.isAuthenticated) {
 return abortNavigation()
 }

 if (to.path !== '/edit-post') {
 return navigateTo('/edit-post')
 }
})
```
```

`err` as a String

You can pass the error as a string:

```
```ts [middleware/auth.ts]
export default defineNuxtRouteMiddleware((to, from) => {
 const user = useState('user')

 if (!user.value.isAuthenticated) {
 return abortNavigation('Insufficient permissions.')
 }
})
```
```

`err` as an Error Object

You can pass the error as an [`Error`](https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Global_Objects/Error) object, e.g. caught by the `catch`-block:

```
```ts [middleware/auth.ts]
export default defineNuxtRouteMiddleware((to, from) => {
 try {
 /* code that might throw an error */
 } catch (err) {
 return abortNavigation(err)
 }
})
```
```

```

---
title: 'addRouteMiddleware'
description: 'addRouteMiddleware() is a helper function to dynamically add middleware in your application.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
    size: xs
---

::note
Route middleware are navigation guards stored in the [middleware/](/docs/guide/directory-structure/middleware) directory of your Nuxt application (unless [set otherwise](/docs/api/nuxt-config#middleware)).
::

## Type

```ts
addRouteMiddleware (name: string | RouteMiddleware, middleware?: RouteMiddleware, options: AddRouteMiddlewareOptions = {})
```

## Parameters

### `name`

- **Type:** `string` | `RouteMiddleware`

Can be either a string or a function of type `RouteMiddleware`. Function takes the next route `to` as the first argument and the current route `from` as the second argument, both of which are Vue route objects.

Learn more about available properties of [route objects](/docs/api/composables/use-route).

### `middleware`

- **Type:** `RouteMiddleware`

The second argument is a function of type `RouteMiddleware`. Same as above, it provides `to` and `from` route objects. It becomes optional if the first argument in `addRouteMiddleware()` is already passed as a function.

### `options`

- **Type:** `AddRouteMiddlewareOptions`

An optional `options` argument lets you set the value of `global` to `true` to indicate whether the router middleware is global or not (set to `false` by default).

## Examples

### Anonymous Route Middleware

Anonymous route middleware does not have a name. It takes a function as the first argument, making the second `middleware` argument redundant:

```ts [plugins/my-plugin.ts]
export default defineNuxtPlugin(() => {
 addRouteMiddleware((to, from) => {
 if (to.path === '/forbidden') {
 return false
 }
 })
})
```

### Named Route Middleware

Named route middleware takes a string as the first argument and a function as the second.

When defined in a plugin, it overrides any existing middleware of the same name located in the `middleware/` directory:

```ts [plugins/my-plugin.ts]
export default defineNuxtPlugin(() => {
 addRouteMiddleware('named-middleware', () => {
 console.log('named middleware added in Nuxt plugin')
 })
})
```

### Global Route Middleware

Set an optional, third argument `{ global: true }` to indicate whether the route middleware is global:

```ts [plugins/my-plugin.ts]
export default defineNuxtPlugin(() => {

```

```
addRouteMiddleware('global-middleware', (to, from) => {
 console.log('global middleware that runs on every route change')
},
{ global: true }
)
})
`
```

```

title: 'useAsyncData'
description: useAsyncData provides access to data that resolves asynchronously in an SSR-friendly composable.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
 size: xs

```

Within your pages, components, and plugins you can use useAsyncData to get access to data that resolves asynchronously.

```

::note
[`useAsyncData`](/docs/api/composables/use-async-data) is a composable meant to be called directly in the [Nuxt context]
(/docs/guide/going-further/nuxt-app#the-nuxt-context). It returns reactive composables and handles adding responses to the
Nuxt payload so they can be passed from server to client without re-fetching the data on client side when the page
hydrates.
::

```

## ## Usage

```

```vue [pages/index.vue]
<script setup lang="ts">
const { data, status, error, refresh, clear } = await useAsyncData(
  'mountains',
  () => $fetch('https://api.nuxtjs.dev/mountains')
)
</script>
```

```

```

::warning
If you're using a custom useAsyncData wrapper, do not await it in the composable, as that can cause unexpected behavior.
Please follow [this recipe](/docs/guide/recipes/custom-usefetch#custom-usefetch) for more information on how to make a custom
async data fetcher.
::

```

```

::note
`data`, `status` and `error` are Vue refs and they should be accessed with `.value` when used within the `
```



By default, this is: ``key => nuxt.isHydrating ? nuxt.payload.data[key] : nuxt.static.data[key]``, which only caches data when ``payloadExtraction`` is enabled.

- ``pick``: only pick specified keys in this array from the ``handler`` function result
- ``watch``: watch reactive sources to auto-refresh
- ``deep``: return data in a deep ref object (it is ``true`` by default). It can be set to ``false`` to return data in a shallow ref object, which can improve performance if your data does not need to be deeply reactive.
- ``dedupe``: avoid fetching same key more than once at a time (defaults to ``cancel``). Possible options:
  - ``cancel`` - cancels existing requests when a new one is made
  - ``defer`` - does not make new requests at all if there is a pending request

:::note

Under the hood, ``lazy: false`` uses `<Suspense>` to block the loading of the route before the data has been fetched. Consider using ``lazy: true`` and implementing a loading state instead for a snappier user experience.

::

:::read-more{to="/docs/api/composables/use-lazy-async-data"}

You can use ``useLazyAsyncData`` to have the same behavior as ``lazy: true`` with ``useAsyncData``.

::

:::tip{icon="i-simple-icons-youtube" color="gray" to="https://www.youtube.com/watch?v=aQPR0xn-MMk" target="\_blank"}

Learn how to use ``transform`` and ``getCachedData`` to avoid superfluous calls to an API and cache data for visitors on the client.

::

## ## Return Values

- ``data``: the result of the asynchronous function that is passed in.
- ``refresh`/`execute``: a function that can be used to refresh the data returned by the ``handler`` function.
- ``error``: an error object if the data fetching failed.
- ``status``: a string indicating the status of the data request (``"idle"``, ``"pending"``, ``"success"``, ``"error"``).
- ``clear``: a function which will set ``data`` to ``undefined``, set ``error`` to ``null``, set ``status`` to ``"idle"``, and mark any currently pending requests as cancelled.

By default, Nuxt waits until a ``refresh`` is finished before it can be executed again.

:::note

If you have not fetched data on the server (for example, with ``server: false``), then the data will not be fetched until hydration completes. This means even if you await `[`useAsyncData`](/docs/api/composables/use-async-data)` on the client side, ``data`` will remain ``null`` within `<script setup>`.

::

## ## Type

``ts [Signature]

```
function useAsyncData<DataT, DataE>(
 handler: (nuxtApp?: NuxtApp) => Promise<DataT>,
 options?: AsyncDataOptions<DataT>
) : AsyncData<DataT, DataE>
```

```
function useAsyncData<DataT, DataE>(
 key: string,
 handler: (nuxtApp?: NuxtApp) => Promise<DataT>,
 options?: AsyncDataOptions<DataT>
) : Promise<AsyncData<DataT, DataE>
```

```
type AsyncDataOptions<DataT> = {
 server?: boolean
 lazy?: boolean
 immediate?: boolean
 deep?: boolean
 dedupe?: 'cancel' | 'defer'
 default?: () => DataT | Ref<DataT> | null
 transform?: (input: DataT) => DataT | Promise<DataT>
 pick?: string[]
 watch?: WatchSource[]
 getCachedData?: (key: string, nuxtApp: NuxtApp) => DataT
}
```

```
type AsyncData<DataT, ErrorT> = {
 data: Ref<DataT | null>
 refresh: (opts?: AsyncDataExecuteOptions) => Promise<void>
 execute: (opts?: AsyncDataExecuteOptions) => Promise<void>
 clear: () => void
 error: Ref<ErrorT | null>
 status: Ref<AsyncDataRequestStatus>
};
```

```
interface AsyncDataExecuteOptions {
 dedupe?: 'cancel' | 'defer'
}
```

```
type AsyncDataRequestStatus = 'idle' | 'pending' | 'success' | 'error'
```

``

:read-more{to="/docs/getting-started/data-fetching"}

```

title: 'Rendering Modes'
description: 'Learn about the different rendering modes available in Nuxt.'

```

Nuxt supports different rendering modes, [universal rendering](#universal-rendering), [client-side rendering](#client-side-rendering) but also offers [hybrid-rendering](#hybrid-rendering) and the possibility to render your application on [CDN Edge Servers](#edge-side-rendering).

Both the browser and server can interpret JavaScript code to turn Vue.js components into HTML elements. This step is called **rendering**. Nuxt supports both **universal** and **client-side** rendering. The two approaches have benefits and downsides that we will cover.

By default, Nuxt uses **universal rendering** to provide better user experience, performance and to optimize search engine indexing, but you can switch rendering modes in [one line of configuration](/docs/api/nuxt-config#ssr).

## ## Universal Rendering

When the browser requests a URL with universal (server-side + client-side) rendering enabled, the server returns a fully rendered HTML page to the browser. Whether the page has been generated in advance and cached or is rendered on the fly, at some point, Nuxt has run the JavaScript (Vue.js) code in a server environment, producing an HTML document. Users immediately get the content of our application, contrary to client-side rendering. This step is similar to traditional **server-side rendering** performed by PHP or Ruby applications.

To not lose the benefits of the client-side rendering method, such as dynamic interfaces and pages transitions, the Client (browser) loads the JavaScript code that runs on the Server in the background once the HTML document has been downloaded. The browser interprets it again (hence **Universal rendering**) and Vue.js takes control of the document and enables interactivity.

Making a static page interactive in the browser is called "Hydration".

Universal rendering allows a Nuxt application to provide quick page load times while preserving the benefits of client-side rendering. Furthermore, as the content is already present in the HTML document, crawlers can index it without overhead.

[Users can access the static content when the HTML document is loaded. Hydration then allows page's interactivity](/assets/docs/concepts/rendering/ssr.svg)

### **\*\*Benefits of server-side rendering:\*\***

- **\*\*Performance\*\***: Users can get immediate access to the page's content because browsers can display static content much faster than JavaScript-generated content. At the same time, Nuxt preserves the interactivity of a web application when the hydration process happens.
- **\*\*Search Engine Optimization\*\***: Universal rendering delivers the entire HTML content of the page to the browser as a classic server application. Web crawlers can directly index the page's content, which makes Universal rendering a great choice for any content that you want to index quickly.

### **\*\*Downsides of server-side rendering:\*\***

- **\*\*Development constraints\*\***: Server and browser environments don't provide the same APIs, and it can be tricky to write code that can run on both sides seamlessly. Fortunately, Nuxt provides guidelines and specific variables to help you determine where a piece of code is executed.
- **\*\*Cost\*\***: A server needs to be running in order to render pages on the fly. This adds a monthly cost like any traditional server. However, the server calls are highly reduced thanks to universal rendering with the browser taking over on client-side navigation. A cost reduction is possible by leveraging [edge-side-rendering](#edge-side-rendering).

Universal rendering is very versatile and can fit almost any use case, and is especially appropriate for any content-oriented websites: **blogs, marketing websites, portfolios, e-commerce sites, and marketplaces.**

### **::tip**

For more examples about writing Vue code without hydration mismatch, see [the Vue docs](https://vuejs.org/guide/scaling-up/ssr.html#hydration-mismatch).

::

### **::important**

When importing a library that relies on browser APIs and has side effects, make sure the component importing it is only called client-side. Bundlers do not treeshake imports of modules containing side effects.

::

## ## Client-Side Rendering

Out of the box, a traditional Vue.js application is rendered in the browser (or **client**). Then, Vue.js generates HTML elements after the browser downloads and parses all the JavaScript code containing the instructions to create the current interface.

[Users have to wait for the browser to download, parse and execute the JavaScript before seeing the page's content](/assets/docs/concepts/rendering/csr.svg)

### **\*\*Benefits of client-side rendering:\*\***

- **\*\*Development speed\*\***: When working entirely on the client-side, we don't have to worry about the server compatibility of the code, for example, by using browser-only APIs like the `window` object.
- **\*\*Cheaper\*\***: Running a server adds a cost of infrastructure as you would need to run on a platform that supports JavaScript. We can host Client-only applications on any static server with HTML, CSS, and JavaScript files.
- **\*\*Offline\*\***: Because code entirely runs in the browser, it can nicely keep working while the internet is unavailable.

### **\*\*Downsides of client-side rendering:\*\***

- **Performance**: The user has to wait for the browser to download, parse and run JavaScript files. Depending on the network for the download part and the user's device for the parsing and execution, this can take some time and impact the user's experience.

- **Search Engine Optimization**: Indexing and updating the content delivered via client-side rendering takes more time than with a server-rendered HTML document. This is related to the performance drawback we discussed, as search engine crawlers won't wait for the interface to be fully rendered on their first try to index the page. Your content will take more time to show and update in search results pages with pure client-side rendering.

Client-side rendering is a good choice for heavily interactive **web applications** that don't need indexing or whose users visit frequently. It can leverage browser caching to skip the download phase on subsequent visits, such as **SaaS**, back-office applications, or online games.

You can enable client-side only rendering with Nuxt in your `nuxt.config.ts`:

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 ssr: false
})
``
```

**note**

If you do use `ssr: false`, you should also place an HTML file in `~/app/spa-loading-template.html` with some HTML you would like to use to render a loading screen that will be rendered until your app is hydrated.

`read-more{title="SPA Loading Template" to="/docs/api/configuration/nuxt-config#spaladingtemplate"}`

**tip**

`to="https://www.youtube.com/watch?v=7Lr0QTP1Ro8" icon="i-logos-youtube-icon" target="_blank"`

Watch a video from Alexander Lichter about **Building a plain SPA with Nuxt!**.

**tip**

### Deploying a Static Client-Rendered App

If you deploy your app to [static hosting](/docs/getting-started/deployment#static-hosting) with the `nuxi generate` or `nuxi build --prerender` commands, then by default, Nuxt will render every page as a separate static HTML file.

If you are using purely client-side rendering, then this might be unnecessary. You might only need a single `index.html` file, plus `200.html` and `404.html` fallbacks, which you can tell your static web host to serve up for all requests.

In order to achieve this we can change how the routes are prerendered. Just add this to [your hooks](/docs/api/advanced/hooks#nuxt-hooks-build-time) in your `nuxt.config.ts`:

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 hooks: {
 'prerender:routes' ({ routes }) {
 routes.clear() // Do not generate any routes (except the defaults)
 }
 },
})
``
```

This will produce three files:

```
- `index.html`
- `200.html`
- `404.html`
```

The `200.html` and `404.html` might be useful for the hosting provider you are using.

### Hybrid Rendering

Hybrid rendering allows different caching rules per route using **Route Rules** and decides how the server should respond to a new request on a given URL.

Previously every route/page of a Nuxt application and server must use the same rendering mode, universal or client-side. In various cases, some pages could be generated at build time, while others should be client-side rendered. For example, think of a content website with an admin section. Every content page should be primarily static and generated once, but the admin section requires registration and behaves more like a dynamic application.

Nuxt includes route rules and hybrid rendering support. Using route rules you can define rules for a group of nuxt routes, change rendering mode or assign a cache strategy based on route!

Nuxt server will automatically register corresponding middleware and wrap routes with cache handlers using [Nitro caching layer](https://nitro.unjs.io/guide/cache).

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 routeRules: {
 // Homepage pre-rendered at build time
 '/': { prerender: true },
 // Products page generated on demand, revalidates in background, cached until API response changes
 '/products': { swr: true },
 // Product page generated on demand, revalidates in background, cached for 1 hour (3600 seconds)
 }
})
``
```

```

'/products/**': { swr: 3600 },
// Blog posts page generated on demand, revalidates in background, cached on CDN for 1 hour (3600 seconds)
'/blog': { isr: 3600 },
// Blog post page generated on demand once until next deployment, cached on CDN
'/blog/**': { isr: true },
// Admin dashboard renders only on client-side
'/admin/**': { ssr: false },
// Add cors headers on API routes
'/api/**': { cors: true },
// Redirects legacy urls
'/old-page': { redirect: '/new-page' }
}
})
```

```

Route Rules

The different properties you can use are the following:

- ``redirect: string`{lang=ts}` - Define server-side redirects.
- ``ssr: boolean`{lang=ts}` - Disables server-side rendering for sections of your app and make them SPA-only with ``ssr: false``
- ``cors: boolean`{lang=ts}` - Automatically adds cors headers with ``cors: true`` - you can customize the output by overriding with ``headers``
- ``headers: object`{lang=ts}` - Add specific headers to sections of your site - for example, your assets
- ``swr: number | boolean`{lang=ts}` - Add cache headers to the server response and cache it on the server or reverse proxy for a configurable TTL (time to live). The ``node-server`` preset of Nitro is able to cache the full response. When the TTL expired, the cached response will be sent while the page will be regenerated in the background. If true is used, a ``stale-while-revalidate`` header is added without a `MaxAge`.
- ``isr: number | boolean`{lang=ts}` - The behavior is the same as ``swr`` except that we are able to add the response to the CDN cache on platforms that support this (currently Netlify or Vercel). If ``true`` is used, the content persists until the next deploy inside the CDN.
- ``prerender: boolean`{lang=ts}` - Prerenders routes at build time and includes them in your build as static assets
- ``experimentalNoScripts: boolean`{lang=ts}` - Disables rendering of Nuxt scripts and JS resource hints for sections of your site.
- ``appMiddleware: string | string[] | Record<string, boolean>`{lang=ts}` - Allows you to define middleware that should or should not run for page paths within the Vue app part of your application (that is, not your Nitro routes)

Whenever possible, route rules will be automatically applied to the deployment platform's native rules for optimal performances (Netlify and Vercel are currently supported).

::important

Note that Hybrid Rendering is not available when using `[`nuxt generate`](/docs/api/commands/generate)`.

::

****Examples:****

```

::card-group
  ::card
    ---
    icon: i-simple-icons-github
    title: Nuxt Vercel ISR
    to: https://github.com/danielroe/nuxt-vercel-isr
    target: _blank
    ui.icon.base: text-black dark:text-white
    ---
    Example of a Nuxt application with hybrid rendering deployed on Vercel.
  ::
::

```

Edge-Side Rendering

Edge-Side Rendering (ESR) is a powerful feature introduced in Nuxt that allows the rendering of your Nuxt application closer to your users via edge servers of a Content Delivery Network (CDN). By leveraging ESR, you can ensure improved performance and reduced latency, thereby providing an enhanced user experience.

With ESR, the rendering process is pushed to the 'edge' of the network - the CDN's edge servers. Note that ESR is more a deployment target than an actual rendering mode.

When a request for a page is made, instead of going all the way to the original server, it's intercepted by the nearest edge server. This server generates the HTML for the page and sends it back to the user. This process minimizes the physical distance the data has to travel, ****reducing latency and loading the page faster****.

Edge-side rendering is possible thanks to [Nitro](https://nitro.unjs.io), the [server engine](/docs/guide/concepts/server-engine) that powers Nuxt 3. It offers cross-platform support for Node.js, Deno, Cloudflare Workers, and more.

The current platforms where you can leverage ESR are:

- [Cloudflare Pages](https://pages.cloudflare.com) with zero configuration using the git integration and the ``nuxt build`` command
- [Vercel Edge Functions](https://vercel.com/features/edge-functions) using the ``nuxt build`` command and ``NITRO_PRESET=vercel-edge`` environment variable
- [Netlify Edge Functions](https://www.netlify.com/products/#netlify-edge-functions) using the ``nuxt build`` command and ``NITRO_PRESET=netlify-edge`` environment variable

Note that ****Hybrid Rendering**** can be used when using Edge-Side Rendering with route rules.

You can explore open source examples deployed on some of the platform mentioned above:

```
::card-group
::card
---
icon: i-simple-icons-github
title: Nuxt Todos Edge
to: https://github.com/atinux/nuxt-todos-edge
target: _blank
ui.icon.base: text-black dark:text-white
---
A todos application with user authentication, SSR and SQLite.
::
::card
---
icon: i-simple-icons-github
title: Atinotes
to: https://github.com/atinux/atinotes
target: _blank
ui.icon.base: text-black dark:text-white
---
An editable website with universal rendering based on CloudFlare KV.
::
::
<!-- TODO: link to templates with ESR category for examples -->
```

```

---
title: Testing
description: How to test your Nuxt application.
navigation.icon: i-ph-check-circle-duotone
---

::tip
If you are a module author, you can find more specific information in the [Module Author's guide](/docs/guide/going-further/modules#testing).
::

Nuxt offers first-class support for end-to-end and unit testing of your Nuxt application via `@nuxt/test-utils`, a library of test utilities and configuration that currently powers the [tests we use on Nuxt itself](https://github.com/nuxt/nuxt/tree/main/test) and tests throughout the module ecosystem.

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=yGzwk9xi9gU" target="_blank"}
Watch a video from Alexander Lichter about getting started with the `@nuxt/test-utils`.
::

## Installation

In order to allow you to manage your other testing dependencies, `@nuxt/test-utils` ships with various optional peer dependencies. For example:

- you can choose between `happy-dom` and `jsdom` for a runtime Nuxt environment
- you can choose between `vitest`, `cucumber`, `jest` and `playwright` for end-to-end test runners
- `playwright-core` is only required if you wish to use the built-in browser testing utilities (and are not using `@playwright/test` as your test runner)

::code-group
```bash [yarn]
yarn add --dev @nuxt/test-utils vitest @vue/test-utils happy-dom playwright-core
```
```bash [npm]
npm i --save-dev @nuxt/test-utils vitest @vue/test-utils happy-dom playwright-core
```
```bash [pnpm]
pnpm add -D @nuxt/test-utils vitest @vue/test-utils happy-dom playwright-core
```
```bash [bun]
bun add --dev @nuxt/test-utils vitest @vue/test-utils happy-dom playwright-core
```
::

## Unit Testing

We currently ship an environment for unit testing code that needs a [Nuxt](https://nuxt.com) runtime environment. It currently _only_ has support for `vitest` (although contribution to add other runtimes would be welcome).

### Setup

1. Add `@nuxt/test-utils/module` to your `nuxt.config` file (optional). It adds a Vitest integration to your Nuxt DevTools which supports running your unit tests in development.

```ts twoslash
export default defineNuxtConfig({
 modules: [
 '@nuxt/test-utils/module'
]
})
```

2. Create a `vitest.config.ts` with the following content:

```ts twoslash
import { defineVitestConfig } from '@nuxt/test-utils/config'

export default defineVitestConfig({
 // any custom Vitest config you require
})
```

::tip
When importing `@nuxt/test-utils` in your vitest config, It is necessary to have `"type": "module"` specified in your `package.json` or rename your vitest config file appropriately.
> ie. `vitest.config.m{ts,js}`.
::

### Using a Nuxt Runtime Environment

By default, `@nuxt/test-utils` will not change your default Vitest environment, so you can do fine-grained opt-in and run Nuxt tests together with other unit tests.

```

You can opt in to a Nuxt environment by adding ``.nuxt.`` to the test file's name (for example, `my-file.nuxt.test.ts`` or `my-file.nuxt.spec.ts``) or by adding `@vitest-environment nuxt`` as a comment directly in the test file.

```
``ts twoslash
// @vitest-environment nuxt
import { test } from 'vitest'

test('my test', () => {
  // ... test with Nuxt environment!
})
``
```

You can alternatively set `environment: 'nuxt`` in your Vitest configuration to enable the Nuxt environment for **all tests**.

```
``ts twoslash
// vitest.config.ts
import { fileURLToPath } from 'node:url'
import { defineVitestConfig } from '@nuxt/test-utils/config'

export default defineVitestConfig({
  test: {
    environment: 'nuxt',
    // you can optionally set Nuxt-specific environment options
    // environmentOptions: {
    //   nuxt: {
    //     rootDir: fileURLToPath(new URL('./playground', import.meta.url)),
    //     domEnvironment: 'happy-dom', // 'happy-dom' (default) or 'jsdom'
    //     overrides: {
    //       // other Nuxt config you want to pass
    //     }
    //   }
    // }
  }
})
``
```

If you have set `environment: 'nuxt`` by default, you can then opt `_out_` of the [default environment] (<https://vitest.dev/guide/environment.html#test-environment>) per test file as needed.

```
``ts twoslash
// @vitest-environment node
import { test } from 'vitest'

test('my test', () => {
  // ... test without Nuxt environment!
})
``
```

`::warning`
When you run your tests within the Nuxt environment, they will be running in a [`happy-dom``] (<https://github.com/capricorn86/happy-dom>) or [`jsdom``] (<https://github.com/jsdom/jsdom>) environment. Before your tests run, a global Nuxt app will be initialized (including, for example, running any plugins or code you've defined in your `app.vue``).

This means you should take particular care not to mutate the global state in your tests (or, if you need to, to reset it afterwards).
`::`

Built-In Mocks

`@nuxt/test-utils`` provides some built-in mocks for the DOM environment.

`IntersectionObserver`

Default `true``, creates a dummy class without any functionality for the IntersectionObserver API

`indexedDB`

Default `false``, uses [`fake-indexeddb``] (<https://github.com/dumbmatter/fakeIndexedDB>) to create a functional mock of the IndexedDB API

These can be configured in the `environmentOptions`` section of your `vitest.config.ts`` file:

```
``ts twoslash
import { defineVitestConfig } from '@nuxt/test-utils/config'

export default defineVitestConfig({
  test: {
    environmentOptions: {
      nuxt: {
        mock: {
          intersectionObserver: true,
          indexedDb: true,

```



```

    }
  }
}
})
...

```

ÖŸ> i, Helpers

`@nuxt/test-utils` provides a number of helpers to make testing Nuxt apps easier.

`mountSuspended`

`mountSuspended` allows you to mount any Vue component within the Nuxt environment, allowing async setup and access to injections from your Nuxt plugins.

```
:::alert{type=info}
```

Under the hood, `mountSuspended` wraps `mount` from `@vue/test-utils`, so you can check out [the Vue Test Utils documentation](https://test-utils.vuejs.org/guide/) for more on the options you can pass, and how to use this utility.

For example:

```

```:ts twoslash
import { it, expect } from 'vitest'
import type { Component } from 'vue'
declare module '#components' {
 export const SomeComponent: Component
}
// ---cut---
// tests/components/SomeComponents.nuxt.spec.ts
import { mountSuspended } from '@nuxt/test-utils/runtime'
import { SomeComponent } from '#components'

it('can mount some component', async () => {
 const component = await mountSuspended(SomeComponent)
 expect(component.text()).toMatchInlineSnapshot(
 '"This is an auto-imported component"'
)
})
...

```

```

```:ts twoslash
import { it, expect } from 'vitest'
// ---cut---
// tests/components/SomeComponents.nuxt.spec.ts
import { mountSuspended } from '@nuxt/test-utils/runtime'
import App from '~/app.vue'

// tests/App.nuxt.spec.ts
it('can also mount an app', async () => {
  const component = await mountSuspended(App, { route: '/test' })
  expect(component.html()).toMatchInlineSnapshot(`
    "<div>This is an auto-imported component</div>
    <div> I am a global component </div>
    <div></div>
    <a href="/test"> Test link </a>"
  `)
})
...

```

`renderSuspended`

`renderSuspended` allows you to render any Vue component within the Nuxt environment using `@testing-library/vue`, allowing async setup and access to injections from your Nuxt plugins.

This should be used together with utilities from Testing Library, e.g. `screen` and `fireEvent`. Install [testing-library/vue](https://testing-library.com/docs/vue-testing-library/intro) in your project to use these.

Additionally, Testing Library also relies on testing globals for cleanup. You should turn these on in your [Vitest config](https://vitest.dev/config/#globals).

The passed in component will be rendered inside a `

Examples:

```

```:ts twoslash
import { it, expect } from 'vitest'
import type { Component } from 'vue'
declare module '#components' {
 export const SomeComponent: Component
}

```

```
// ---cut---
// tests/components/SomeComponents.nuxt.spec.ts
import { renderSuspended } from '@nuxt/test-utils/runtime'
import { SomeComponent } from '#components'
import { screen } from '@testing-library/vue'

it('can render some component', async () => {
 await renderSuspended(SomeComponent)
 expect(screen.getByText('This is an auto-imported component')).toBeDefined()
})
````
```

```
````ts twoslash
import { it, expect } from 'vitest'
// ---cut---
// tests/App.nuxt.spec.ts
import { renderSuspended } from '@nuxt/test-utils/runtime'
import App from '~/app.vue'

it('can also render an app', async () => {
 const html = await renderSuspended(App, { route: '/test' })
 expect(html).toMatchInlineSnapshot(`
 <div id="test-wrapper">
 <div>This is an auto-imported component</div>
 <div> I am a global component </div>
 <div>Index page</div> Test link
 </div>`
)
})
````
```

`mockNuxtImport`

`mockNuxtImport` allows you to mock Nuxt's auto import functionality. For example, to mock `useStorage`, you can do so like this:

```
````ts twoslash
import { mockNuxtImport } from '@nuxt/test-utils/runtime'

mockNuxtImport('useStorage', () => {
 return () => {
 return { value: 'mocked storage' }
 }
})

// your tests here
````
```

```
::alert{type=info}
`mockNuxtImport` can only be used once per mocked import per test file. It is actually a macro that gets transformed to
`vi.mock` and `vi.mock` is hoisted, as described [here](https://vitest.dev/api/vi.html#vi-mock).
::
```

If you need to mock a Nuxt import and provide different implementations between tests, you can do it by creating and exposing your mocks using [`vi.hoisted`](https://vitest.dev/api/vi.html#vi-hoisted), and then use those mocks in `mockNuxtImport`. You then have access to the mocked imports, and can change the implementation between tests. Be careful to [restore mocks](https://vitest.dev/api/mock.html#mockrestore) before or after each test to undo mock state changes between runs.

```
````ts twoslash
import { vi } from 'vitest'
import { mockNuxtImport } from '@nuxt/test-utils/runtime'

const { useStorageMock } = vi.hoisted(() => {
 return {
 useStorageMock: vi.fn().mockImplementation(() => {
 return { value: 'mocked storage' }
 })
 }
})

mockNuxtImport('useStorage', () => {
 return useStorageMock
})

// Then, inside a test
useStorageMock.mockImplementation(() => {
 return { value: 'something else' }
})
````
```

`mockComponent`

`mockComponent` allows you to mock Nuxt's component.

The first argument can be the component name in PascalCase, or the relative path of the component. The second argument is a factory function that returns the mocked component.

For example, to mock `MyComponent`, you can:

```
```:ts twoslash
import { mockComponent } from '@nuxt/test-utils/runtime'

mockComponent('MyComponent', {
 props: {
 value: String
 },
 setup(props) {
 // ...
 }
})

// relative path or alias also works
mockComponent '~/components/my-component.vue', async () => {
 // or a factory function
 return defineComponent({
 setup(props) {
 // ...
 }
 })
})

// or you can use SFC for redirecting to a mock component
mockComponent('MyComponent', () => import('./MockComponent.vue'))

// your tests here
```:
```

> **Note**: You can't reference local variables in the factory function since they are hoisted. If you need to access Vue APIs or other variables, you need to import them in your factory function.

```
```:ts twoslash
import { mockComponent } from '@nuxt/test-utils/runtime'

mockComponent('MyComponent', async () => {
 const { ref, h } = await import('vue')

 return defineComponent({
 setup(props) {
 const counter = ref(0)
 return () => h('div', null, counter.value)
 }
 })
})
```:
```

`registerEndpoint`

`registerEndpoint` allows you create Nitro endpoint that returns mocked data. It can come in handy if you want to test a component that makes requests to API to display some data.

The first argument is the endpoint name (e.g. `/test/`). The second argument is a factory function that returns the mocked data.

For example, to mock `/test/` endpoint, you can do:

```
```:ts twoslash
import { registerEndpoint } from '@nuxt/test-utils/runtime'

registerEndpoint('/test/', () => ({
 test: 'test-field'
}))
```:
```

By default, your request will be made using the `GET` method. You may use another method by setting an object as the second argument instead of a function.

```
```:ts twoslash
import { registerEndpoint } from '@nuxt/test-utils/runtime'

registerEndpoint('/test/', {
 method: 'POST',
 handler: () => ({ test: 'test-field' })
})
```:
```

> **Note**: If your requests in a component go to an external API, you can use `baseUrl` and then make it empty using [Nuxt Environment Override Config](/docs/getting-started/configuration#environment-overrides) (`\$test`) so all your requests will

go to Nitro server.

Conflict with End-To-End Testing

`@nuxt/test-utils/runtime` and `@nuxt/test-utils/e2e` need to run in different testing environments and so can't be used in the same file.

If you would like to use both the end-to-end and unit testing functionality of `@nuxt/test-utils`, you can split your tests into separate files. You then either specify a test environment per-file with the special `// @vitest-environment nuxt` comment, or name your runtime unit test files with the `.nuxt.spec.ts` extension.

`app.nuxt.spec.ts`

```
````ts twoslash
import { mockNuxtImport } from '@nuxt/test-utils/runtime'

mockNuxtImport('useStorage', () => {
 return () => {
 return { value: 'mocked storage' }
 }
})
...

```

`app.e2e.spec.ts`

```
````ts twoslash
import { setup, $fetch } from '@nuxt/test-utils/e2e'

await setup({
  setupTimeout: 10000,
})

// ...
...

```

Using `@vue/test-utils`

If you prefer to use `@vue/test-utils` on its own for unit testing in Nuxt, and you are only testing components which do not rely on Nuxt composables, auto-imports or context, you can follow these steps to set it up.

1. Install the needed dependencies

```
::code-group
````bash [yarn]
yarn add --dev vitest @vue/test-utils happy-dom @vitejs/plugin-vue
...
````bash [npm]
npm i --save-dev vitest @vue/test-utils happy-dom @vitejs/plugin-vue
...
````bash [pnpm]
pnpm add -D vitest @vue/test-utils happy-dom @vitejs/plugin-vue
...
````bash [bun]
bun add --dev vitest @vue/test-utils happy-dom @vitejs/plugin-vue
...
::

```

2. Create a `vitest.config.ts` with the following content:

```
````ts twoslash
import { defineConfig } from 'vitest/config'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
 plugins: [vue()],
 test: {
 environment: 'happy-dom',
 },
});
...

```

3. Add a new command for test in your `package.json`

```
````json
"scripts": {
  "build": "nuxt build",
  "dev": "nuxt dev",
  ...
  "test": "vitest"
},
...

```

4. Create a simple ``<HelloWorld>`` component `components/HelloWorld.vue` with the following content:

```
``vue
<template>
  <p>Hello world</p>
</template>
``
```

5. Create a simple unit test for this newly created component `~/components/HelloWorld.spec.ts`

```
``ts twoslash
import { describe, it, expect } from 'vitest'
import { mount } from '@vue/test-utils'

import HelloWorld from './HelloWorld.vue'

describe('HelloWorld', () => {
  it('component renders Hello world properly', () => {
    const wrapper = mount(HelloWorld)
    expect(wrapper.text()).toContain('Hello world')
  })
})
``
```

6. Run vitest command

```
::code-group
``bash [yarn]
yarn test
``
``bash [npm]
npm run test
``
``bash [pnpm]
pnpm run test
``
``bash [bun]
bun run test
``
::
```

Congratulations, you're all set to start unit testing with `@vue/test-utils` in Nuxt! Happy testing!

End-To-End Testing

For end-to-end testing, we support [Vitest](https://github.com/vitest-dev/vitest), [Jest](https://jestjs.io), [Cucumber](https://cucumber.io/) and [Playwright](https://playwright.dev/) as test runners.

Setup

In each `describe` block where you are taking advantage of the `@nuxt/test-utils/e2e` helper methods, you will need to set up the test context before beginning.

```
``ts twoslash [test/my-test.spec.ts]
import { describe, test } from 'vitest'
import { setup, $fetch } from '@nuxt/test-utils/e2e'

describe('My test', async () => {
  await setup({
    // test context options
  })

  test('my test', () => {
    // ...
  })
})
``
```

Behind the scenes, `setup` performs a number of tasks in `beforeAll`, `beforeEach`, `afterEach` and `afterAll` to set up the Nuxt test environment correctly.

Please use the options below for the `setup` method.

Nuxt Config

- `rootDir`: Path to a directory with a Nuxt app to be put under test.
- Type: `string`
- Default: ``.``
- `configFile`: Name of the configuration file.
- Type: `string`
- Default: `'nuxt.config'`

<!--

- ``config``: Object with configuration overrides.
 - Type: ``NuxtConfig``
 - Default: ``{}`` -->

Timings

- ``setupTimeout``: The amount of time (in milliseconds) to allow for ``setupTest`` to complete its work (which could include building or generating files for a Nuxt application, depending on the options that are passed).
 - Type: ``number``
 - Default: ``60000``

Features

- ``build``: Whether to run a separate build step.
 - Type: ``boolean``
 - Default: ``true`` (``false`` if ``browser`` or ``server`` is disabled, or if a ``host`` is provided)
- ``server``: Whether to launch a server to respond to requests in the test suite.
 - Type: ``boolean``
 - Default: ``true`` (``false`` if a ``host`` is provided)
- ``port``: If provided, set the launched test server port to the value.
 - Type: ``number | undefined``
 - Default: ``undefined``
- ``host``: If provided, a URL to use as the test target instead of building and running a new server. Useful for running "real" end-to-end tests against a deployed version of your application, or against an already running local server (which may provide a significant reduction in test execution timings). See the [target host end-to-end example below](#target-host-end-to-end-example).
 - Type: ``string``
 - Default: ``undefined``
- ``browser``: Under the hood, Nuxt test utils uses [playwright](https://playwright.dev) to carry out browser testing. If this option is set, a browser will be launched and can be controlled in the subsequent test suite.
 - Type: ``boolean``
 - Default: ``false``
- ``browserOptions``
 - Type: ``object`` with the following properties
 - ``type``: The type of browser to launch - either ``chromium``, ``firefox`` or ``webkit``
 - ``launch``: ``object`` of options that will be passed to playwright when launching the browser. See [full API reference](https://playwright.dev/docs/api/class-browsertype#browser-type-launch).
- ``runner``: Specify the runner for the test suite. Currently, [Vitest](https://vitest.dev) is recommended.
 - Type: ``'vitest' | 'jest' | 'cucumber'``
 - Default: ``'vitest'``

Target ``host`` end-to-end example

A common use-case for end-to-end testing is running the tests against a deployed application running in the same environment typically used for Production.

For local development or automated deploy pipelines, testing against a separate local server can be more efficient and is typically faster than allowing the test framework to rebuild between tests.

To utilize a separate target host for end-to-end tests, simply provide the ``host`` property of the ``setup`` function with the desired URL.

```

```ts
import { setup, createPage } from '@nuxt/test-utils/e2e'
import { describe, it, expect } from 'vitest'

describe('login page', async () => {
 await setup({
 host: 'http://localhost:8787',
 })

 it('displays the email and password fields', async () => {
 const page = await createPage('/login')
 expect(await page.getByTestId('email').isVisible()).toBe(true)
 expect(await page.getByTestId('password').isVisible()).toBe(true)
 })
})
```

```

APIs

``$fetch(url)``

Get the HTML of a server-rendered page.

```

```ts twoslash
import { $fetch } from '@nuxt/test-utils/e2e'

const html = await $fetch('/')

```

```
...
```

```
`fetch(url)`
```

Get the response of a server-rendered page.

```
````ts twoslash
import { fetch } from '@nuxt/test-utils/e2e'

const res = await fetch('/')
const { body, headers } = res
...`
```

```
#### `url(path)`
```

Get the full URL for a given page (including the port the test server is running on.)

```
````ts twoslash
import { url } from '@nuxt/test-utils/e2e'

const pageUrl = url('/page')
// 'http://localhost:6840/page'
...`
```

```
Testing in a Browser
```

We provide built-in support using Playwright within `@nuxt/test-utils`, either programmatically or via the Playwright test runner.

```
`createPage(url)`
```

Within `vitest`, `jest` or `cucumber`, you can create a configured Playwright browser instance with `createPage`, and (optionally) point it at a path from the running server. You can find out more about the API methods available from [in the Playwright documentation](https://playwright.dev/docs/api/class-page).

```
````ts twoslash
import { createPage } from '@nuxt/test-utils/e2e'

const page = await createPage('/page')
// you can access all the Playwright APIs from the `page` variable
...`
```

```
#### Testing with Playwright Test Runner
```

We also provide first-class support for testing Nuxt within [the Playwright test runner](https://playwright.dev/docs/intro).

```
::code-group
````bash [yarn]
yarn add --dev @playwright/test @nuxt/test-utils
...
````bash [npm]
npm i --save-dev @playwright/test @nuxt/test-utils
...
````bash [pnpm]
pnpm add -D @playwright/test @nuxt/test-utils
...
````bash [bun]
bun add --dev @playwright/test @nuxt/test-utils
...
::`
```

You can provide global Nuxt configuration, with the same configuration details as the `setup()` function mentioned earlier in this section.

```
````ts [playwright.config.ts]
import { fileURLToPath } from 'node:url'
import { defineConfig, devices } from '@playwright/test'
import type { ConfigOptions } from '@nuxt/test-utils/playwright'

export default defineConfig<ConfigOptions>({
 use: {
 nuxt: {
 rootDir: fileURLToPath(new URL('.', import.meta.url))
 }
 },
 // ...
})
...`
```

```
::read-more{title="See full example config" to="https://github.com/nuxt/test-utils/blob/main/examples/app-playwright/playwright.config.ts" target="_blank"}
::`
```

Your test file should then use `expect` and `test` directly from `@nuxt/test-utils/playwright`:

```
``ts [tests/example.test.ts]
import { expect, test } from '@nuxt/test-utils/playwright'

test('test', async ({ page, goto }) => {
 await goto('/', { waitUntil: 'hydration' })
 await expect(page.getByRole('heading')).toHaveText('Welcome to Playwright!')
})
``
```

You can alternatively configure your Nuxt server directly within your test file:

```
``ts [tests/example.test.ts]
import { expect, test } from '@nuxt/test-utils/playwright'

test.use({
 nuxt: {
 rootDir: fileURLToPath(new URL('.', import.meta.url))
 }
})

test('test', async ({ page, goto }) => {
 await goto('/', { waitUntil: 'hydration' })
 await expect(page.getByRole('heading')).toHaveText('Welcome to Playwright!')
})
``
```



```

title: "Features"
description: "Enable or disable optional Nuxt features to unlock new possibilities."

```

Some features of Nuxt are available on an opt-in basis, or can be disabled based on your needs.

## `features`

### inlineStyles

Inline styles when rendering HTML. This is currently available only when using Vite.

You can also pass a function that receives the path of a Vue component and returns a boolean indicating whether to inline the styles for that component.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 features: {
 inlineStyles: true // or a function to determine inlining
 }
})
``
```

### noScripts

Disables rendering of Nuxt scripts and JS resource hints. Can also be configured granularly within `routeRules`.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 features: {
 noScripts: true
 }
})
``
```

## `future`

There is also a `future` namespace for early opting-in to new features that will become default in a future (possibly major) version of the framework.

### compatibilityVersion

```
::important
This configuration option is available in Nuxt v3.12+.
::
```

This enables early access to Nuxt features or flags.

Setting `compatibilityVersion` to `4` changes defaults throughout your Nuxt configuration to opt-in to Nuxt v4 behaviour, but you can granularly re-enable Nuxt v3 behaviour when testing (see example). Please file issues if so, so that we can address in Nuxt or in the ecosystem.

```
``ts
export default defineNuxtConfig({
 future: {
 compatibilityVersion: 4,
 },
 // To re-enable _all_ Nuxt v3 behaviour, set the following options:
 srcDir: '.',
 dir: {
 app: 'app'
 },
 experimental: {
 compileTemplate: true,
 templateUtils: true,
 relativeWatchPaths: true,
 defaults: {
 useAsyncData: {
 deep: true
 }
 }
 },
 unhead: {
 renderSSRHeadOptions: {
 omitLineBreaks: false
 }
 }
})
``
```

### typescriptBundlerResolution

This enables 'Bundler' module resolution mode for TypeScript, which is the recommended setting for frameworks like Nuxt and [Vite](https://vitejs.dev/guide/performance.html#reduce-resolve-operations).

It improves type support when using modern libraries with `exports`.

See [the original TypeScript pull request](https://github.com/microsoft/TypeScript/pull/51669).

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 future: {
 typescriptBundlerResolution: true
 }
})
``
```

```

title: "How Nuxt Works?"
description: "Nuxt is a minimal but highly customizable framework to build web applications."

```

This guide helps you better understand Nuxt internals to develop new solutions and module integrations on top of Nuxt.

## ## The Nuxt Interface

When you start Nuxt in development mode with [`nuxi dev`](/docs/api/commands/dev) or building a production application with [`nuxi build`](/docs/api/commands/build), a common context will be created, referred to as `nuxt` internally. It holds normalized options merged with `nuxt.config` file, some internal state, and a powerful [hooking system](/docs/api/advanced/hooks) powered by [unjs/hookable](https://github.com/unjs/hookable) allowing different components to communicate with each other. You can think of it as **Builder Core**.

This context is globally available to be used with [Nuxt Kit](/docs/guide/going-further/kit) composables. Therefore only one instance of Nuxt is allowed to run per process.

To extend the Nuxt interface and hook into different stages of the build process, we can use [Nuxt Modules](/docs/guide/going-further/modules).

For more details, check out [the source code](https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/core/nuxt.ts).

## ## The NuxtApp Interface

When rendering a page in the browser or on the server, a shared context will be created, referred to as `nuxtApp`. This context keeps vue instance, runtime hooks, and internal states like `ssrContext` and `payload` for hydration. You can think of it as **Runtime Core**.

This context can be accessed using [`useNuxtApp()`](/docs/api/composables/use-nuxt-app) composable within Nuxt plugins and `<script setup>` and vue composables. Global usage is possible for the browser but not on the server, to avoid sharing context between users.

Since [`useNuxtApp()`](/docs/api/composables/use-nuxt-app) throws an exception if context is currently unavailable, if your composable does not always require `nuxtApp`, you can use [`tryUseNuxtApp()`](/docs/api/composables/use-nuxt-app#tryusenuxtapp) instead, which will return `null` instead of throwing an exception.

To extend the `nuxtApp` interface and hook into different stages or access contexts, we can use [Nuxt Plugins](/docs/guide/directory-structure/plugins).

Check [Nuxt App](/docs/api/composables/use-nuxt-app) for more information about this interface.

`nuxtApp` has the following properties:

```
```js
const nuxtApp = {
  vueApp, // the global Vue application: https://vuejs.org/api/application.html#application-api

  versions, // an object containing Nuxt and Vue versions

  // These let you call and add runtime NuxtApp hooks
  // https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts#L18
  hooks,
  hook,
  callHook,

  // Only accessible on server-side
  ssrContext: {
    url,
    req,
    res,
    runtimeConfig,
    noSSR,
  },

  // This will be stringified and passed from server to client
  payload: {
    serverRendered: true,
    data: {},
    state: {}
  }

  provide: (name: string, value: any) => void
}
```
```

For more details, check out [the source code](https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts).

## ## Runtime Context vs. Build Context

Nuxt builds and bundles project using Node.js but also has a runtime side.

While both areas can be extended, that runtime context is isolated from build-time. Therefore, they are not supposed to share state, code, or context other than runtime configuration!

``nuxt.config`` and [\[Nuxt Modules\]\(/docs/guide/going-further/modules\)](#) can be used to extend the build context, and [\[Nuxt Plugins\]\(/docs/guide/directory-structure/plugins\)](#) can be used to extend runtime.

When building an application for production, ``nuxi build`` will generate a standalone build in the ``.output`` directory, independent of ``nuxt.config`` and [\[Nuxt modules\]\(/docs/guide/going-further/modules\)](#).

```

title: 'useCookie'
description: useCookie is an SSR-friendly composable to read and write cookies.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/cookie.ts
 size: xs

```

Within your pages, components and plugins you can use `useCookie`, an SSR-friendly composable to read and write cookies.

```

````ts
const cookie = useCookie(name, options)
````

::note
`useCookie` only works in the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context).
::

::tip
`useCookie` ref will automatically serialize and deserialize cookie value to JSON.
::

```

## ## Example

The example below creates a cookie called `counter`. If the cookie doesn't exist, it is initially set to a random value. Whenever we update the `counter` variable, the cookie will be updated accordingly.

```

````vue [app.vue]
<script setup lang="ts">
const counter = useCookie('counter')

counter.value = counter.value || Math.round(Math.random() * 1000)
</script>

<template>
  <div>
    <h1>Counter: {{ counter || '-' }}</h1>
    <button @click="counter = null">reset</button>
    <button @click="counter--">-</button>
    <button @click="counter++">+</button>
  </div>
</template>
````

```

```
:link-example{to="/docs/examples/advanced/use-cookie"}
```

```

::note
Refresh `useCookie` values manually when a cookie has changed with [refreshCookie](/docs/api/utils/refresh-cookie).
::

```

## ## Options

Cookie composable accepts several options which let you modify the behavior of cookies.

Most of the options will be directly passed to the [cookie](https://github.com/jshttp/cookie) package.

### ### `maxAge` / `expires`

Use these options to set the expiration of the cookie.

`maxAge`: Specifies the `number` (in seconds) to be the value for the [`Max-Age` `Set-Cookie` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.2). The given number will be converted to an integer by rounding down. By default, no maximum age is set.

`expires`: Specifies the `Date` object to be the value for the [`Expires` `Set-Cookie` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.1). By default, no expiration is set. Most clients will consider this a "non-persistent cookie" and will delete it on a condition like exiting a web browser application.

```

::note
The [cookie storage model specification](https://tools.ietf.org/html/rfc6265#section-5.3) states that if both `expires` and `maxAge` is set, then `maxAge` takes precedence, but not all clients may obey this, so if both are set, they should point to the same date and time!
::

```

```

::note
If neither of `expires` and `maxAge` is set, the cookie will be session-only and removed when the user closes their browser.
::

```

### ### `httpOnly`

Specifies the ``boolean`` value for the [``HttpOnly`` ``Set-Cookie`` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.6). When truthy, the ``HttpOnly`` attribute is set; otherwise it is not. By default, the ``HttpOnly`` attribute is not set.

```
::warning
Be careful when setting this to `true`, as compliant clients will not allow client-side JavaScript to see the cookie in
`document.cookie`.
::
```

### ``secure``

Specifies the ``boolean`` value for the [``Secure`` ``Set-Cookie`` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.5). When truthy, the ``Secure`` attribute is set; otherwise it is not. By default, the ``Secure`` attribute is not set.

```
::warning
Be careful when setting this to `true`, as compliant clients will not send the cookie back to the server in the future if the
browser does not have an HTTPS connection. This can lead to hydration errors.
::
```

### ``partitioned``

Specifies the ``boolean`` value for the [``Partitioned`` ``Set-Cookie``](https://datatracker.ietf.org/doc/html/draft-cutler-httpbis-partitioned-cookies#section-2.1) attribute. When truthy, the ``Partitioned`` attribute is set, otherwise it is not. By default, the ``Partitioned`` attribute is not set.

```
::note
This is an attribute that has not yet been fully standardized, and may change in the future.
This also means many clients may ignore this attribute until they understand it.
```

More information can be found in the [proposal](https://github.com/privacycg/CHIPS).

```
::
```

### ``domain``

Specifies the value for the [``Domain`` ``Set-Cookie`` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.3). By default, no domain is set, and most clients will consider applying the cookie only to the current domain.

### ``path``

Specifies the value for the [``Path`` ``Set-Cookie`` attribute](https://tools.ietf.org/html/rfc6265#section-5.2.4). By default, the path is considered the ["default path"](<https://tools.ietf.org/html/rfc6265#section-5.1.4>).

### ``sameSite``

Specifies the ``boolean`` or ``string`` value for the [``SameSite`` ``Set-Cookie`` attribute](https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-03#section-4.1.2.7).

- ``true`` will set the ``SameSite`` attribute to ``Strict`` for strict same-site enforcement.
- ``false`` will not set the ``SameSite`` attribute.
- ``'lax'`` will set the ``SameSite`` attribute to ``Lax`` for lax same-site enforcement.
- ``'none'`` will set the ``SameSite`` attribute to ``None`` for an explicit cross-site cookie.
- ``'strict'`` will set the ``SameSite`` attribute to ``Strict`` for strict same-site enforcement.

More information about the different enforcement levels can be found in [the specification](<https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-03#section-4.1.2.7>).

### ``encode``

Specifies a function that will be used to encode a cookie's value. Since the value of a cookie has a limited character set (and must be a simple string), this function can be used to encode a value into a string suited for a cookie's value.

The default encoder is the ``JSON.stringify`` + ``encodeURIComponent``.

### ``decode``

Specifies a function that will be used to decode a cookie's value. Since the value of a cookie has a limited character set (and must be a simple string), this function can be used to decode a previously encoded cookie value into a JavaScript string or other object.

The default decoder is ``decodeURIComponent`` + [``destr``](https://github.com/unjs/destr).

```
::note
If an error is thrown from this function, the original, non-decoded cookie value will be returned as the cookie's value.
::
```

### ``default``

Specifies a function that returns the cookie's default value. The function can also return a ``Ref``.

### ``readonly``

Allows ``_accessing`` a cookie value without the ability to set it.

### ``watch``

Specifies the `boolean` or `string` value for [watch](https://vuejs.org/api/reactivity-core.html#watch) cookie ref data.

- `true` - Will watch cookie ref data changes and its nested properties (default).
- `shallow` - Will watch cookie ref data changes for only top level properties
- `false` - Will not watch cookie ref data changes.

:::note

Refresh `useCookie` values manually when a cookie has changed with [`refreshCookie`](/docs/api/utils/refresh-cookie).

:::

**\*\*Example 1:\*\***

```
```\nvue\n<script setup lang="ts">\nconst user = useCookie(\n  'userInfo',\n  {\n    default: () => ({ score: -1 }),\n    watch: false\n  }\n)\n\nif (user.value && user.value !== null) {\n  user.value.score++; // userInfo cookie not update with this change\n}\n</script>\n\n<template>\n  <div>User score: {{ user?.score }}</div>\n</template>\n```\n
```

****Example 2:****

```
```\nvue\n<script setup lang="ts">\nconst list = useCookie(\n  'list',\n  {\n    default: () => [],\n    watch: 'shallow'\n  }\n)\n\nfunction add() {\n  list.value?.push(Math.round(Math.random() * 1000))\n  // list cookie not update with this change\n}\n\nfunction save() {\n  if (list.value && list.value !== null) {\n    list.value = [...list.value]\n    // list cookie update with this change\n  }\n}\n</script>\n\n<template>\n  <div>\n    <h1>List</h1>\n    <pre>{{ list }}</pre>\n    <button @click="add">Add</button>\n    <button @click="save">Save</button>\n  </div>\n</template>\n```\n
```

## ## Cookies in API Routes

You can use `getCookie` and `setCookie` from [`h3`](https://github.com/unjs/h3) package to set cookies in server API routes.

```
```\nts [server/api/counter.ts]\nexport default defineEventHandler(event => {\n  // Read counter cookie\n  let counter = getCookie(event, 'counter') || 0\n\n  // Increase counter cookie by 1\n  setCookie(event, 'counter', ++counter)\n\n  // Send JSON response\n  return { counter }\n})\n
```

```
...  
  
:link-example{to="/docs/examples/advanced/use-cookie"}
```



```
---
title: '<Teleport>'
description: The <Teleport> component teleports a component to a different location in the DOM.
---

::warning
The `to` target of [<Teleport>](https://vuejs.org/guide/built-ins/teleport.html) expects a CSS selector string or an actual DOM node. Nuxt currently has SSR support for teleports to `#teleports` only, with client-side support for other targets using a <ClientOnly> wrapper.
::
```

Body Teleport

```
```vue
<template>
 <button @click="open = true">
 Open Modal
 </button>
 <Teleport to="#teleports">
 <div v-if="open" class="modal">
 <p>Hello from the modal!</p>
 <button @click="open = false">
 Close
 </button>
 </div>
 </Teleport>
</template>
```
```

Client-side Teleport

```
```vue
<template>
 <ClientOnly>
 <Teleport to="#some-selector">
 <!-- content -->
 </Teleport>
 </ClientOnly>
</template>
```
```

```
:link-example{to="/docs/examples/advanced/teleport"}
```

```
---
title: "<NuxtPicture>"
description: "Nuxt provides a <NuxtPicture> component to handle automatic image optimization."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/image/blob/main/src/runtime/components/nuxt-picture.ts
    size: xs
---
```

`<NuxtPicture>` is a drop-in replacement for the native `<picture>` tag.

Usage of `<NuxtPicture>` is almost identical to [`<NuxtImg>`](/docs/api/components/nuxt-img) but it also allows serving modern formats like `webp` when possible.

Learn more about the [`<picture>` tag on MDN](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/picture).

Setup

In order to use `<NuxtPicture>` you should install and enable the Nuxt Image module:

```
```bash [Terminal]
npx nuxi@latest module add image
```
```

::read-more{to="https://image.nuxt.com/usage/nuxt-picture" target="_blank"}
Read more about the `<NuxtPicture>` component.
::

```
---
title: "useError"
description: useError composable returns the global Nuxt error that is being handled.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/error.ts
    size: xs
---
```

The composable returns the global Nuxt error that is being handled and it is available on both client and server.

```
```ts
const error = useError()
```
```

`useError` sets an error in the state and creates a reactive as well as SSR-friendly global Nuxt error across components.

Nuxt errors have the following properties:

```
```ts
interface {
 // HTTP response status code
 statusCode: number
 // HTTP response status message
 statusMessage: string
 // Error message
 message: string
}
```

```
:read-more{to="/docs/getting-started/error-handling"}
```

```

title: 'composables'
head.title: 'composables/'
description: Use the composables/ directory to auto-import your Vue composables into your application.
navigation.icon: i-ph-folder-duotone

```

## ## Usage

**\*\*Method 1:\*\*** Using named export

```
```js [composables/useFoo.ts]
export const useFoo = () => {
  return useState('foo', () => 'bar')
}
```
```

**\*\*Method 2:\*\*** Using default export

```
```js [composables/use-foo.ts or composables/useFoo.ts]
// It will be available as useFoo() (camelCase of file name without extension)
export default function () {
  return useState('foo', () => 'bar')
}
```
```

**\*\*Usage:\*\*** You can now use auto imported composable in ``.js``, ``.ts`` and ``.vue`` files

```
```vue [app.vue]
<script setup lang="ts">
const foo = useFoo()
</script>
```

```
<template>
  <div>
    {{ foo }}
  </div>
</template>
```
```

`::alert{type=info}`

The `composables/` directory in Nuxt does not provide any additional reactivity capabilities to your code. Instead, any reactivity within composables is achieved using Vue's Composition API mechanisms, such as `ref` and `reactive`. Note that reactive code is also not limited to the boundaries of the `composables/` directory. You are free to employ reactivity features wherever they're needed in your application.

`::`

`:read-more{to="/docs/guide/concepts/auto-imports"}`

`:link-example{to="/docs/examples/features/auto-imports"}`

## ## Types

Under the hood, Nuxt auto generates the file ``.nuxt/imports.d.ts`` to declare the types.

Be aware that you have to run `[`nuxi prepare`](/docs/api/commands/prepare)`, `[`nuxi dev`](/docs/api/commands/dev)` or `[`nuxi build`](/docs/api/commands/build)` in order to let Nuxt generate the types.

`::note`

If you create a composable without having the dev server running, TypeScript will throw an error, such as ``Cannot find name 'useBar'.`

`::`

## ## Examples

### ### Nested Composables

You can use a composable within another composable using auto imports:

```
```js [composables/test.ts]
export const useFoo = () => {
  const nuxtApp = useNuxtApp()
  const bar = useBar()
}
```
```

### ### Access plugin injections

You can access `[plugin injections](/docs/guide/directory-structure/plugins#automatically-providing-helpers)` from composables:

```
```js [composables/test.ts]
export const useHello = () => {
  const nuxtApp = useNuxtApp()
```

```

    return nuxtApp.$hello
  }
  ...

```

How Files Are Scanned

Nuxt only scans files at the top level of the [`composables/`` directory](/docs/guide/directory-structure/composables), e.g.:

```

```bash [Directory Structure]
| composables/
---| index.ts // scanned
---| useFoo.ts // scanned
----| nested/
-----| utils.ts // not scanned
```

```

Only `composables/index.ts`` and `composables/useFoo.ts`` would be searched for imports.

To get auto imports working for nested modules, you could either re-export them (recommended) or configure the scanner to include nested directories:

Example: Re-export the composables you need from the `composables/index.ts`` file:

```

```ts [composables/index.ts]
// Enables auto import for this export
export { utils } from './nested/utils.ts'
```

```

Example: Scan nested directories inside the `composables/`` folder:

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 imports: {
 dirs: [
 // Scan top-level modules
 'composables',
 // ... or scan modules nested one level deep with a specific name and file extension
 'composables/*/index.{ts,js,mjs,mts}',
 // ... or scan all modules within given directory
 'composables/**'
]
 }
})
```

```

```
---
title: "Experimental Features"
description: "Enable Nuxt experimental features to unlock new possibilities."
---
```

The Nuxt experimental features can be enabled in the Nuxt configuration file.

Internally, Nuxt uses `@nuxt/schema` to define these experimental features. You can refer to the [API documentation] (/docs/api/configuration/nuxt-config#experimental) or the [source code] (https://github.com/nuxt/nuxt/blob/main/packages/schema/src/config/experimental.ts) for more information.

```
::note
Note that these features are experimental and could be removed or modified in the future.
::
```

asyncContext

Enable native async context to be accessible for nested composables in Nuxt and in Nitro. This opens the possibility to use composables inside async composables and reduce the chance to get the ``Nuxt instance is unavailable`` error.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 asyncContext: true
 }
})
```:
```

```
::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/pull/20918" target="_blank"}
See full explanation on the GitHub pull-request.
::
```

asyncEntry

Enables generation of an async entry point for the Vue bundle, aiding module federation support.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 asyncEntry: true
 }
})
```:
```

externalVue

Externalizes `vue`, `@vue/*` and `vue-router` when building.

Enabled by default.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 externalVue: true
 }
})
```:
```

```
::warning
This feature will likely be removed in a near future.
::
```

emitRouteChunkError

Emits `app:chunkError` hook when there is an error loading vite/webpack chunks. Default behavior is to perform a hard reload of the new route when a chunk fails to load.

You can disable automatic handling by setting this to `false`, or handle chunk errors manually by setting it to `manual`.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 emitRouteChunkError: 'automatic' // or 'manual' or false
 }
})
```:
```

restoreState

Allows Nuxt app state to be restored from `sessionStorage` when reloading the page after a chunk error or manual `[`reloadNuxtApp()`](/docs/api/utils/reload-nuxt-app)` call.

To avoid hydration errors, it will be applied only after the Vue app has been mounted, meaning there may be a flicker on

initial load.

::important

Consider carefully before enabling this as it can cause unexpected behavior, and consider providing explicit keys to [`useState`](/docs/api/composables/use-state) as auto-generated keys may not match across builds.

::

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 restoreState: true
 }
})
```
```

inlineRouteRules

Define route rules at the page level using [`defineRouteRules`](/docs/api/utils/define-route-rules).

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 inlineRouteRules: true
 }
})
```
```

Matching route rules will be created, based on the page's `path`.

[::read-more{to="/docs/api/utils/define-route-rules" icon="i-ph-function-duotone"}](/docs/api/utils/define-route-rules)
Read more in `defineRouteRules` utility.

::

[:read-more{to="/docs/guide/concepts/rendering#hybrid-rendering" icon="i-ph-medal-duotone"}](/docs/guide/concepts/rendering#hybrid-rendering)

renderJsonPayloads

Allows rendering of JSON payloads with support for revivifying complex types.

Enabled by default.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 renderJsonPayloads: true
 }
})
```
```

noVueServer

Disables Vue server renderer endpoint within Nitro.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 noVueServer: true
 }
})
```
```

payloadExtraction

Enables extraction of payloads of pages generated with `nuxt generate`.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 payloadExtraction: true
 }
})
```
```

clientFallback

Enables the experimental [`<NuxtClientFallback>`](/docs/api/components/nuxt-client-fallback) component for rendering content on the client if there's an error in SSR.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 clientFallback: true
 }
})
```
```

```
}
})
``,`
```

crossOriginPrefetch

Enables cross-origin prefetch using the Speculation Rules API.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 crossOriginPrefetch: true
 }
})
``,`
```

[::read-more{icon="i-simple-icons-w3c" color="gray" to="https://wicg.github.io/nav-speculation/prefetch.html" target="\\_blank"}](https://wicg.github.io/nav-speculation/prefetch.html)  
Read more about the **Speculation Rules API**.  
::

### ## viewTransition

Enables View Transition API integration with client-side router.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  experimental: {
    viewTransition: true
  }
})
``,`
```

[:link-example{to="https://stackblitz.com/edit/nuxt-view-transitions?file=app.vue" target="_blank"}](https://stackblitz.com/edit/nuxt-view-transitions?file=app.vue)

[::read-more{icon="i-simple-icons-mdnwebdocs" color="gray" to="https://developer.mozilla.org/en-US/docs/Web/API/View_Transitions_API" target="_blank"}](https://developer.mozilla.org/en-US/docs/Web/API/View_Transitions_API)
Read more about the **View Transition API**.
::

writeEarlyHints

Enables writing of early hints when using node server.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 writeEarlyHints: true
 }
})
``,`
```

### ## componentIslands

Enables experimental component islands support with [`<NuxtIsland>`](/docs/api/components/nuxt-island) and `.island.vue` files.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  experimental: {
    componentIslands: true // false or 'local+remote'
  }
})
``,`
```

[:read-more{to="/docs/guide/directory-structure/components#server-components"}](/docs/guide/directory-structure/components#server-components)

[::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/issues/19772" target="_blank"}](https://github.com/nuxt/nuxt/issues/19772)
You can follow the server components roadmap on GitHub.
::

localLayerAliases

Resolve `~``, `~~``, `@`` and `@@`` aliases located within layers with respect to their layer source and root directories.

Enabled by default.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 localLayerAliases: true
 }
})
``,`
```



## ## typedPages

Enable the new experimental typed router using [`unplugin-vue-router`](https://github.com/posva/unplugin-vue-router).

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 typedPages: true
 }
})
``
```

Out of the box, this will enable typed usage of [`navigateTo`](/docs/api/utils/navigate-to), [`<NuxtLink>`](/docs/api/components/nuxt-link), [`router.push()`](/docs/api/composables/use-router) and more.

You can even get typed params within a page by using `const route = useRoute('route-name')`.

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=SXk-L19gTZk" target="_blank"}
Watch a video from Daniel Roe explaining type-safe routing in Nuxt.
::
```

## ## watcher

Set an alternative watcher that will be used as the watching service for Nuxt.

Nuxt uses `chokidar-granular` by default, which will ignore top-level directories (like `node_modules` and `.git`) that are excluded from watching.

You can set this instead to `parcel` to use `@parcel/watcher`, which may improve performance in large projects or on Windows platforms.

You can also set this to `chokidar` to watch all files in your source directory.

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 watcher: 'chokidar-granular' // 'chokidar' or 'parcel' are also options
 }
})
``
```

## ## sharedPrerenderData

Enabling this feature automatically shares payload *data* between pages that are prerendered. This can result in a significant performance improvement when prerendering sites that use `useAsyncData` or `useFetch` and fetch the same data in different pages.

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 sharedPrerenderData: true
 }
})
``
```

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=1jUupYHVvrU" target="_blank"}
Watch a video from Alexander Lichter about the experimental `sharedPrerenderData` setting.
::
```

It is particularly important when enabling this feature to make sure that any unique key of your data is always resolvable to the same data. For example, if you are using `useAsyncData` to fetch data related to a particular page, you should provide a key that uniquely matches that data. (`useFetch` should do this automatically for you.)

```
``ts
// This would be unsafe in a dynamic page (e.g. `[slug].vue`) because the route slug makes a difference
// to the data fetched, but Nuxt can't know that because it's not reflected in the key.
const route = useRoute()
const { data } = await useAsyncData(async () => {
 return await $fetch(`/api/my-page/${route.params.slug}`)
})
// Instead, you should use a key that uniquely identifies the data fetched.
const { data } = await useAsyncData(route.params.slug, async () => {
 return await $fetch(`/api/my-page/${route.params.slug}`)
})
``
```

## ## clientNodeCompat

With this feature, Nuxt will automatically polyfill Node.js imports in the client build using [`unenv`](https://github.com/unjs/unenv).

```
::alert{type=info}
```

To make globals like ``Buffer`` work in the browser, you need to manually inject them.

```
``ts
import { Buffer } from 'node:buffer'

globalThis.Buffer = globalThis.Buffer || Buffer
``
::
```

```
scanPageMeta
```

This option allows exposing some route metadata defined in ``definePageMeta`` at build-time to modules (specifically ``alias``, ``name``, ``path``, ``redirect``).

This only works with static or strings/arrays rather than variables or conditional assignment. See [original issue] (<https://github.com/nuxt/nuxt/issues/24770>) for more information and context.

You can disable this feature if it causes issues in your project.

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 scanPageMeta: false
 }
})
``
```

```
cookieStore
```

Enables CookieStore support to listen for cookie updates (if supported by the browser) and refresh ``useCookie`` ref values.

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 experimental: {
 cookieStore: true
 }
})
``
```

```
::read-more{icon="i-simple-icons-mdnwebdocs" color="gray" to="https://developer.mozilla.org/en-US/docs/Web/API/CookieStore"
target="_blank"}
```

Read more about the **CookieStore**.

```
::
```

```

title: Server Engine
description: 'Nuxt is powered by a new server engine: Nitro.'

```

While building Nuxt 3, we created a new server engine: [Nitro](https://nitro.unjs.io).

It is shipped with many features:

- Cross-platform support for Node.js, Browsers, service-workers and more.
- Serverless support out-of-the-box.
- API routes support.
- Automatic code-splitting and async-loaded chunks.
- Hybrid mode for static + serverless sites.
- Development server with hot module reloading.

## ## API Layer

Server [API endpoints](/docs/guide/directory-structure/server#api-routes) and [Middleware](/docs/guide/directory-structure/server#server-middleware) are added by Nitro that internally uses [h3](https://github.com/unjs/h3).

Key features include:

- Handlers can directly return objects/arrays for an automatically-handled JSON response
- Handlers can return promises, which will be awaited (`res.end()` and `next()` are also supported)
- Helper functions for body parsing, cookie handling, redirects, headers and more

Check out [the h3 docs](https://github.com/unjs/h3) for more information.

```
::read-more{to="/docs/guide/directory-structure/server#server-routes"}
Learn more about the API layer in the `server/` directory.
::
```

## ## Direct API Calls

Nitro allows 'direct' calling of routes via the globally-available [`$fetch`](/docs/api/utils/dollarfetch) helper. This will make an API call to the server if run on the browser, but will directly call the relevant function if run on the server, **saving an additional API call**.

[`$fetch`](/docs/api/utils/dollarfetch) API is using [ofetch](https://github.com/unjs/ofetch), with key features including:

- Automatic parsing of JSON responses (with access to raw response if needed)
- Request body and params are automatically handled, with correct `Content-Type` headers

For more information on `$fetch` features, check out [ofetch](https://github.com/unjs/ofetch).

## ## Typed API Routes

When using API routes (or middleware), Nitro will generate typings for these routes as long as you are returning a value instead of using `res.end()` to send a response.

You can access these types when using [`$fetch()`](/docs/api/utils/dollarfetch) or [`useFetch()`](/docs/api/composables/use-fetch).

## ## Standalone Server

Nitro produces a standalone server dist that is independent of `node_modules`.

The server in Nuxt 2 is not standalone and requires part of Nuxt core to be involved by running `nuxt start` (with the [`nuxt-start`](https://www.npmjs.com/package/nuxt-start) or [`nuxt`](https://www.npmjs.com/package/nuxt) distributions) or custom programmatic usage, which is fragile and prone to breakage and not suitable for serverless and service-worker environments.

Nuxt generates this dist when running `nuxt build` into a [`.output`](/docs/guide/directory-structure/output) directory.

The output contains runtime code to run your Nuxt server in any environment (including experimental browser service workers!) and serve your static files, making it a true hybrid framework for the JAMstack. In addition, Nuxt implements a native storage layer, supporting multi-source drivers and local assets.

```
::read-more{color="gray" icon="i-simple-icons-github" to="https://github.com/unjs/nitro" target="_blank"}
Read more about Nitro engine on GitHub.
::
```

```

title: 'Installation'
description: 'Get started with Nuxt quickly with our online starters or start locally with your terminal.'
navigation.icon: i-ph-play-duotone

```

### ## Play Online

If you just want to play around with Nuxt in your browser without setting up a project, you can use one of our online sandboxes:

```
::card-group
 :card{title="Open on StackBlitz" icon="i-simple-icons-stackblitz" to="https://nuxt.new/s/v3" target="_blank"}
 :card{title="Open on CodeSandbox" icon="i-simple-icons-codesandbox" to="https://nuxt.new/c/v3" target="_blank"}
::
```

Or follow the steps below to set up a new Nuxt project on your computer.

### ## New Project

```
<!-- TODO: need to fix upstream in nuxt/nuxt.com -->
<!-- markdownlint-disable-next-line MD001 -->
```

#### #### Prerequisites

- **Node.js** - [`v18.0.0`](https://nodejs.org/en) or newer
- **Text editor** - We recommend [Visual Studio Code](https://code.visualstudio.com/) with the [official Vue extension](https://marketplace.visualstudio.com/items?itemName=Vue.volar) (previously known as Volar)
- **Terminal** - In order to run Nuxt commands

```
::note
 ::details
 :summary[Additional notes for an optimal setup:]
 - Node.js: Make sure to use an even numbered version (18, 20, etc)
 - Nuxt: Install the community-developed [Nuxt extension](https://marketplace.visualstudio.com/items?itemName=Nuxt.nuxt-vscode)
 ::
::
```

Open a terminal (if you're using [Visual Studio Code](https://code.visualstudio.com), you can open an [integrated terminal](https://code.visualstudio.com/docs/editor/integrated-terminal)) and use the following command to create a new starter project:

```
::code-group

```bash [npm]
npx nuxi@latest init <project-name>
```

```bash [yarn]
yarn dlx nuxi@latest init <project-name>
```

```bash [pnpm]
pnpm dlx nuxi@latest init <project-name>
```

```bash [bun]
bun x nuxi@latest init <project-name>
```

::
```

```
::tip
Alternatively, you can find other starters or themes by opening nuxt.new and following the instructions there.
::
```

Open your project folder in Visual Studio Code:

```
```bash [Terminal]
code <project-name>
```
```

Or change directory into your new project from your terminal:

```
```bash
cd <project-name>
```
```

### ## Development Server

Now you'll be able to start your Nuxt app in development mode:

```
::code-group
```

```
```bash [yarn]
yarn dev --open
```
```

```
```bash [npm]
npm run dev -- -o
```
```

```
```bash [pnpm]
pnpm dev -o
```
```

```
```bash [bun]
bun run dev -o
```
```

```
::
```

```
::tip{icon="i-ph-check-circle-duotone"}
```

Well done! A browser window should automatically open for <<http://localhost:3000>>.

```
::
```

## ## Next Steps

Now that you've created your Nuxt project, you are ready to start building your application.

```
:read-more{title="Nuxt Concepts" to="/docs/guide/concepts"}
```

```

title: 'Modules'
description: "Nuxt provides a module system to extend the framework core and simplify integrations."

```

## ## Exploring Nuxt Modules

When developing production-grade applications with Nuxt you might find that the framework's core functionality is not enough. Nuxt can be extended with configuration options and plugins, but maintaining these customizations across multiple projects can be tedious, repetitive and time-consuming. On the other hand, supporting every project's needs out of the box would make Nuxt very complex and hard to use.

This is one of the reasons why Nuxt provides a module system that makes it possible to extend the core. Nuxt modules are async functions that sequentially run when starting Nuxt in development mode using [`nuxi dev`](/docs/api/commands/dev) or building a project for production with [`nuxi build`](/docs/api/commands/build). They can override templates, configure webpack loaders, add CSS libraries, and perform many other useful tasks.

Best of all, Nuxt modules can be distributed in npm packages. This makes it possible for them to be reused across projects and shared with the community, helping create an ecosystem of high-quality add-ons.

```
::read-more{to="/modules"}
Explore Nuxt Modules
::
```

## ## Add Nuxt Modules

Once you have installed the modules you can add them to your [`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config) file under the `modules` property. Module developers usually provide additional steps and details for usage.

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 modules: [
 // Using package name (recommended usage)
 '@nuxtjs/example',

 // Load a local module
 './modules/example',

 // Add module with inline-options
 ['./modules/example', { token: '123' }],

 // Inline module definition
 async (inlineOptions, nuxt) => { }
]
})
``
```

```
::warning
Nuxt modules are now build-time-only, and the `buildModules` property used in Nuxt 2 is deprecated in favor of `modules`.
::
```

## ## Create a Nuxt Module

Everyone has the opportunity to develop modules and we cannot wait to see what you will build.

```
::read-more{to="/docs/guide/going-further/modules" title="Module Author Guide"}
```

```

title: "callOnce"
description: "Run a given function or block of code once during SSR or CSR."
navigation:
 badge: New
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/once.ts
 size: xs

```

```
::important
This utility is available since [Nuxt v3.9](/blog/v3-9).
::
```

### ## Purpose

The `callOnce` function is designed to execute a given function or block of code only once during:

- server-side rendering but not hydration
- client-side navigation

This is useful for code that should be executed only once, such as logging an event or setting up a global state.

### ## Usage

```
```vue [app.vue]
<script setup lang="ts">
const websiteConfig = useState('config')
```

```
await callOnce(async () => {
  console.log('This will only be logged once')
  websiteConfig.value = await $fetch('https://my-cms.com/api/website-config')
})
</script>
```
```

```
::tip{to="/docs/getting-started/state-management#usage-with-pinia"}
`callOnce` is useful in combination with the [Pinia module](/modules/pinia) to call store actions.
::
```

```
:read-more{to="/docs/getting-started/state-management"}
```

```
::warning
Note that `callOnce` doesn't return anything. You should use [useAsyncData](/docs/api/composables/use-async-data) or [useFetch](/docs/api/composables/use-fetch) if you want to do data fetching during SSR.
::
```

```
::note
`callOnce` is a composable meant to be called directly in a setup function, plugin, or route middleware, because it needs to add data to the Nuxt payload to avoid re-calling the function on the client when the page hydrates.
::
```

### ## Type

```
```ts
callOnce(fn?: () => any | Promise<any>): Promise<void>
callOnce(key: string, fn?: () => any | Promise<any>): Promise<void>
```
```

- `key`: A unique key ensuring that the code is run once. If you do not provide a key, then a key that is unique to the file and line number of the instance of `callOnce` will be generated for you.
- `fn`: The function to run once. This function can also return a `Promise` and a value.

```

title: 'clearNuxtData'
description: Delete cached data, error status and pending promises of useAsyncData and useFetch.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
 size: xs

```

```
:::note
This method is useful if you want to invalidate the data fetching for another page.
:::
```

## ## Type

```
```:ts
clearNuxtData (keys?: string | string[] | ((key: string) => boolean)): void
```:
```

## ## Parameters

\* `keys`: One or an array of keys that are used in [`useAsyncData`](/docs/api/composables/use-async-data) to delete their cached data. If no keys are provided, **all data** will be invalidated.



```

title: 'content'
head.title: 'content/'
description: Use the content/ directory to create a file-based CMS for your application.
navigation.icon: i-ph-folder-duotone

```

[Nuxt Content](https://content.nuxt.com) reads the [`content/` directory](/docs/guide/directory-structure/content) in your project and parses `.md`, `.yaml`, `.csv` and `.json` files to create a file-based CMS for your application.

- Render your content with built-in components.
- Query your content with a MongoDB-like API.
- Use your Vue components in Markdown files with the MDC syntax.
- Automatically generate your navigation.

```
::read-more{to="https://content.nuxt.com" target="_blank"}
Learn more in Nuxt Content documentation.
::
```

## ## Enable Nuxt Content

Install the `@nuxt/content` module in your project as well as adding it to your `nuxt.config.ts` with one command:

```
```bash [Terminal]
npx nuxi module add content
```
```

## ## Create Content

Place your markdown files inside the `content/` directory:

```
```md [content/index.md]
# Hello Content
```
```

The module automatically loads and parses them.

## ## Render Content

To render content pages, add a [catch-all route](/docs/guide/directory-structure/pages/#catch-all-route) using the [`<ContentDoc>`](https://content.nuxt.com/components/content-doc) component:

```
```vue [pages/[...slug\].vue]
<template>
  <main>
    <!-- ContentDoc returns content for `$route.path` by default or you can pass a `path` prop -->
    <ContentDoc />
  </main>
</template>
```
```

## ## Documentation

```
::tip{ icon="i-ph-book" }
Head over to <https://content.nuxt.com> to learn more about the Content module features, such as how to build queries and use
Vue components in your Markdown files with the MDC syntax.
::
```

```

title: '<NuxtRouteAnnouncer>'
description: 'The <NuxtRouteAnnouncer> component adds a hidden element with the page title to announce route changes to assistive technologies.'
navigation:
 badge: New
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-route-announcer.ts
 size: xs

```

```
::important
This component is available in Nuxt v3.12+.
::
```

### ## Usage

Add `<NuxtRouteAnnouncer/>` in your `[`app.vue`](/docs/guide/directory-structure/app)` or `[`layouts/`](/docs/guide/directory-structure/layouts)` to enhance accessibility by informing assistive technologies about page title changes. This ensures that navigational changes are announced to users relying on screen readers.

```
``vue [app.vue]
<template>
 <NuxtRouteAnnouncer />
 <NuxtLayout>
 <NuxtPage />
 </NuxtLayout>
</template>
``
```

### ## Slots

You can pass custom HTML or components through the route announcer's default slot.

```
``vue
<template>
 <NuxtRouteAnnouncer>
 <template #default="{ message }">
 <p>{{ message }} was loaded.</p>
 </template>
 </NuxtRouteAnnouncer>
</template>
``
```

### ## Props

- ``atomic``: Controls if screen readers only announce changes or the entire content. Set to true for full content readouts on updates, false for changes only. (default ``false``)
- ``politeness``: Sets the urgency for screen reader announcements: ``off`` (disable the announcement), ``polite`` (waits for silence), or ``assertive`` (interrupts immediately). (default ``polite``)

```
::callout
This component is optional. :br
To achieve full customization, you can implement your own one based on [its source code]
(https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-route-announcer.ts).
::
```

```
::callout
You can hook into the underlying announcer instance using [the `useRouteAnnouncer` composable](/docs/api/composables/use-route-announcer), which allows you to set a custom announcement message.
::
```

```

title: "clearError"
description: "The clearError composable clears all handled errors."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/error.ts
 size: xs

```

Within your pages, components, and plugins, you can use `clearError` to clear all errors and redirect the user.

**Parameters:**

- `options?: { redirect?: string }`

You can provide an optional path to redirect to (for example, if you want to navigate to a 'safe' page).

```
```js
// Without redirect
clearError()

// With redirect
clearError({ redirect: '/homepage' })
```
```

Errors are set in state using [`useError()`](/docs/api/composables/use-error). The `clearError` composable will reset this state and calls the `app:error:cleared` hook with the provided options.

`:read-more{to="/docs/getting-started/error-handling"}`

```

title: "components"
head.title: "components/"
description: "The components/ directory is where you put all your Vue components."
navigation.icon: i-ph-folder-duotone

```

Nuxt automatically imports any components in this directory (along with components that are registered by any modules you may be using).

```

```bash [Directory Structure]
| components/
--| AppHeader.vue
--| AppFooter.vue
```

```

```

```html [app.vue]
<template>
  <div>
    <AppHeader />
    <NuxtPage />
    <AppFooter />
  </div>
</template>
```

```

## ## Component Names

If you have a component in nested directories such as:

```

```bash [Directory Structure]
| components/
--| base/
----| foo/
-----| Button.vue
```

```

... then the component's name will be based on its own path directory and filename, with duplicate segments being removed. Therefore, the component's name will be:

```

```html
<BaseFooButton />
```

```

:::note

For clarity, we recommend that the component's filename matches its name. So, in the example above, you could rename `Button.vue` to be `BaseFooButton.vue`.

::

If you want to auto-import components based only on its name, not path, then you need to set `pathPrefix` option to `false` using extended form of the configuration object:

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  components: [
    {
      path: '~/components',
      pathPrefix: false, // [!code ++]
    },
  ],
});
```

```

This registers the components using the same strategy as used in Nuxt 2. For example, `~/components/Some/MyComponent.vue` will be usable as `` and not ``.

## ## Dynamic Components

If you want to use the Vue `

For example:

```

```vue [pages/index.vue]
<script setup lang="ts">
import { SomeComponent } from '#components'

const MyButton = resolveComponent('MyButton')
</script>

<template>
  <component :is="clickable ? MyButton : 'div'" />
  <component :is="SomeComponent" />
</template>
```

```

```
</template>
````
```

```
::important
```

If you are using `resolveComponent` to handle dynamic components, make sure not to insert anything but the name of the component, which must be a string and not a variable.

```
::
```

Alternatively, though not recommended, you can register all your components globally, which will create async chunks for all your components and make them available throughout your application.

```
````diff
 export default defineNuxtConfig({
 components: {
+ global: true,
+ dirs: ['~/components']
 },
 })
````
```

You can also selectively register some components globally by placing them in a `~/components/global` directory, or by using a `.global.vue` suffix in the filename. As noted above, each global component is rendered in a separate chunk, so be careful not to overuse this feature.

```
::note
```

The `global` option can also be set per component directory.

```
::
```

Dynamic Imports

To dynamically import a component (also known as lazy-loading a component) all you need to do is add the `Lazy` prefix to the component's name. This is particularly useful if the component is not always needed.

By using the `Lazy` prefix you can delay loading the component code until the right moment, which can be helpful for optimizing your JavaScript bundle size.

```
````vue [pages/index.vue]
<script setup lang="ts">
const show = ref(false)
</script>

<template>
 <div>
 <h1>Mountains</h1>
 <LazyMountainsList v-if="show" />
 <button v-if="!show" @click="show = true">Show List</button>
 </div>
</template>
````
```

Direct Imports

You can also explicitly import components from `#components` if you want or need to bypass Nuxt's auto-importing functionality.

```
````vue [pages/index.vue]
<script setup lang="ts">
import { NuxtLink, LazyMountainsList } from '#components'

const show = ref(false)
</script>

<template>
 <div>
 <h1>Mountains</h1>
 <LazyMountainsList v-if="show" />
 <button v-if="!show" @click="show = true">Show List</button>
 <NuxtLink to="/">Home</NuxtLink>
 </div>
</template>
````
```

Custom Directories

By default, only the `~/components` directory is scanned. If you want to add other directories, or change how the components are scanned within a subfolder of this directory, you can add additional directories to the configuration:

```
````ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 components: [
 // ~/calendar-module/components/event/Update.vue => <EventUpdate />
 { path: '~/calendar-module/components' },
],
})
```

```
// ~/user-module/components/account/UserDeleteDialog.vue => <UserDeleteDialog />
{ path: '~/user-module/components', pathPrefix: false },

// ~/components/special-components/Btn.vue => <SpecialBtn />
{ path: '~/components/special-components', prefix: 'Special' },

// It's important that this comes last if you have overrides you wish to apply
// to sub-directories of `~/components`.
//
// ~/components/Btn.vue => <Btn />
// ~/components/base/Btn.vue => <BaseBtn />
'~/components'
]
})
...
```

## ## npm Packages

If you want to auto-import components from an npm package, you can use [``addComponent``](/docs/api/kit/components#addcomponent) in a [local module](/docs/guide/directory-structure/modules) to register them.

```
::code-group
```

```
```ts twoslash [~/modules/register-component.ts]
import { addComponent, defineNuxtModule } from '@nuxt/kit'
```

```
export default defineNuxtModule({
  setup() {
    // import { MyComponent as MyAutoImportedComponent } from 'my-npm-package'
    addComponent({
      name: 'MyAutoImportedComponent',
      export: 'MyComponent',
      filePath: 'my-npm-package',
    })
  },
})
...
```

```
```vue [app.vue]
<template>
 <div>
 <!-- the component uses the name we specified and is auto-imported -->
 <MyAutoImportedComponent />
 </div>
</template>
...
```

```
::
```

```
::note
```

Any nested directories need to be added first as they are scanned in order.

```
::
```

## ## Component Extensions

By default, any file with an extension specified in the [extensions key of ``nuxt.config.ts``](/docs/api/nuxt-config#extensions) is treated as a component.

If you need to restrict the file extensions that should be registered as components, you can use the extended form of the components directory declaration and its ``extensions`` key:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  components: [
    {
      path: '~/components',
      extensions: ['.vue'], // [!code ++]
    }
  ]
})
...
```

Client Components

If a component is meant to be rendered only client-side, you can add the ``.client`` suffix to your component.

```
```bash [Directory Structure]
| components/
--| Comments.client.vue
...
```

```
```vue [pages/example.vue]
<template>
  <div>
```

```

    <!-- this component will only be rendered on client side -->
    <Comments />
  </div>
</template>
```

```

:::note

This feature only works with Nuxt auto-imports and `#components` imports. Explicitly importing these components from their real paths does not convert them into client-only components.

::

:::important

`client` components are rendered only after being mounted. To access the rendered template using `onMounted()`, add `await nextTick()` in the callback of the `onMounted()` hook.

::

:::read-more{to="/docs/api/components/client-only"}

You can also achieve a similar result with the `` component.

::

## ## Server Components

Server components allow server-rendering individual components within your client-side apps. It's possible to use server components within Nuxt, even if you are generating a static site. That makes it possible to build complex sites that mix dynamic components, server-rendered HTML and even static chunks of markup.

Server components can either be used on their own or paired with a [client component](#paired-with-a-client-component).

:::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=u1yyXe86xJM" target="\_blank"}

Watch Learn Vue video about Nuxt Server Components.

::

:::tip{icon="i-ph-article-duotone" to="https://roe.dev/blog/nuxt-server-components" target="\_blank"}

Read Daniel Roe's guide to Nuxt Server Components.

::

## ### Standalone server components

Standalone server components will always be rendered on the server, also known as Islands components.

When their props update, this will result in a network request that will update the rendered HTML in-place.

Server components are currently experimental and in order to use them, you need to enable the 'component islands' feature in your nuxt.config:

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  experimental: {
    componentIslands: true
  }
})
```

```

Now you can register server-only components with the `.server` suffix and use them anywhere in your application automatically.

```

```bash [Directory Structure]
| components/
--| HighlightedMarkdown.server.vue
```

```

```

```vue [pages/example.vue]

```

```

<template>
  <div>
    <!--
      this will automatically be rendered on the server, meaning your markdown parsing + highlighting
      libraries are not included in your client bundle.
    -->
    <HighlightedMarkdown markdown="# Headline" />
  </div>
</template>
```

```

Server-only components use [`<NuxtIsland>`](/docs/api/components/nuxt-island) under the hood, meaning that `lazy` prop and `#fallback` slot are both passed down to it.

:::alert{type=warning}

Server components (and islands) must have a single root element. (HTML comments are considered elements as well.)

::

:::alert{type=warning}

Be careful when nesting islands within other islands as each island adds some extra overhead.

::

```
:::alert{type=warning}
Most features for server-only components and island components, such as slots and client components, are only available for single file components.
::
```

#### #### Client components within server components

```
:::alert{type=info}
This feature needs `experimental.componentIslands.selectiveClient` within your configuration to be true.
::
```

You can partially hydrate a component by setting a `nuxt-client` attribute on the component you wish to be loaded client-side.

```
```vue [components/ServerWithClient.vue]
<template>
  <div>
    <HighlightedMarkdown markdown="# Headline" />
    <!-- Counter will be loaded and hydrated client-side -->
    <Counter nuxt-client :count="5" />
  </div>
</template>
```
```

```
:::alert{type=info}
This only works within a server component. Slots for client components are working only with
`experimental.componentIsland.selectiveClient` set to `deep` and since they are rendered server-side, they are not interactive once client-side.
::
```

#### #### Server Component Context

When rendering a server-only or island component, `` makes a fetch request which comes back with a ``. (This is an internal request if rendered on the server, or a request that you can see in the network tab if it's rendering on client-side navigation.)

This means:

- A new Vue app will be created server-side to create the ``.
- A new 'island context' will be created while rendering the component.
- You can't access the 'island context' from the rest of your app and you can't access the context of the rest of your app from the island component. In other words, the server component or island is isolated from the rest of your app.
- Your plugins will run again when rendering the island, unless they have `env: { islands: false }` set (which you can do in an object-syntax plugin).

Within an island component, you can access its island context through `nuxtApp.ssrContext.islandContext`. Note that while island components are still marked as experimental, the format of this context may change.

```
:::note
Slots can be interactive and are wrapped within a `

` with `display: contents;`
::


```

#### ### Paired with a Client component

In this case, the `.server` + `.client` components are two 'halves' of a component and can be used in advanced use cases for separate implementations of a component on server and client side.

```
```bash [Directory Structure]
| components/
--| Comments.client.vue
--| Comments.server.vue
```

```vue [pages/example.vue]
<template>
  <div>
    <!-- this component will render Comments.server on the server then Comments.client once mounted in the browser -->
    <Comments />
  </div>
</template>
```
```

#### ## Built-In Nuxt Components

There are a number of components that Nuxt provides, including `` and ``. You can read more about them in the API documentation.

```
:::read-more{to="/docs/api"}
::
```

#### ## Library Authors



Making Vue component libraries with automatic tree-shaking and component registration is super easy. ðŸ™ƒ

You can use the ``components:dirs`` hook to extend the directory list without requiring user configuration in your Nuxt module.

Imagine a directory structure like this:

```
``bash [Directory Structure]
| node_modules/
---| awesome-ui/
-----| components/
-----| Alert.vue
-----| Button.vue
-----| nuxt.js
| pages/
---| index.vue
| nuxt.config.js
``
```

Then in ``awesome-ui/nuxt.js`` you can use the ``components:dirs`` hook:

```
``ts twoslash
import { defineNuxtModule, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
 hooks: {
 'components:dirs': (dirs) => {
 const { resolve } = createResolver(import.meta.url)
 // Add ./components dir to the list
 dirs.push({
 path: resolve('./components'),
 prefix: 'awesome'
 })
 }
 }
})
``
```

That's it! Now in your project, you can import your UI library as a Nuxt module in your ``nuxt.config`` file:

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 modules: ['awesome-ui/nuxt']
})
``
```

... and directly use the module components (prefixed with ``awesome-``) in our ``pages/index.vue``:

```
``vue
<template>
 <div>
 My <AwesomeButton>UI button</AwesomeButton>!
 <awesome-alert>Here's an alert!</awesome-alert>
 </div>
</template>
``
```

It will automatically import the components only if used and also support HMR when updating your components in ``node_modules/awesome-ui/components/``.

`:link-example{to="/docs/examples/features/auto-imports"}`

```

title: 'useFetch'
description: 'Fetch data from an API endpoint with an SSR-friendly composable.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/fetch.ts
 size: xs

```

This composable provides a convenient wrapper around [`useAsyncData`](/docs/api/composables/use-async-data) and [`$fetch`](/docs/api/utils/dollarfetch). It automatically generates a key based on URL and fetch options, provides type hints for request url based on server routes, and infers API response type.

```

::note
`useFetch` is a composable meant to be called directly in a setup function, plugin, or route middleware. It returns reactive composables and handles adding responses to the Nuxt payload so they can be passed from server to client without re-fetching the data on client side when the page hydrates.
::

```

## ## Usage

```

```vue [pages/modules.vue]
<script setup lang="ts">
const { data, status, error, refresh, clear } = await useFetch('/api/modules', {
  pick: ['title']
})
</script>
```

```

```

::warning
If you're using a custom useFetch wrapper, do not await it in the composable, as that can cause unexpected behavior. Please follow [this recipe](/docs/guide/recipes/custom-usefetch#custom-usefetch) for more information on how to make a custom async data fetcher.
::

```

```

::note
`data`, `status` and `error` are Vue refs and they should be accessed with `.value` when used within the `<script setup>`, while `refresh`, `execute` and `clear` are plain functions..
::

```

Using the `query` option, you can add search parameters to your query. This option is extended from [unjs/ofetch](https://github.com/unjs/ofetch) and is using [unjs/ufo](https://github.com/unjs/ufo) to create the URL. Objects are automatically stringified.

```

```ts
const param1 = ref('value1')
const { data, status, error, refresh } = await useFetch('/api/modules', {
  query: { param1, param2: 'value2' }
})
```

```

The above example results in `https://api.nuxt.com/modules?param1=value1&param2=value2`.

You can also use [interceptors](https://github.com/unjs/ofetch#%EF%B8%8F-interceptors):

```

```ts
const { data, status, error, refresh, clear } = await useFetch('/api/auth/login', {
  onRequest({ request, options }) {
    // Set the request headers
    options.headers = options.headers || {}
    options.headers.authorization = '...'
  },
  onRequestError({ request, options, error }) {
    // Handle the request errors
  },
  onResponse({ request, response, options }) {
    // Process the response data
    localStorage.setItem('token', response._data.token)
  },
  onResponseError({ request, response, options }) {
    // Handle the response errors
  }
})
```

```

```

::warning
`useFetch` is a reserved function name transformed by the compiler, so you should not name your own function `useFetch`.
::

```

```

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=njsGVmcWviY" target="_blank"}
Watch the video from Alexander Lichter to avoid using `useFetch` the wrong way!

```

```
::

:link-example{to="/docs/examples/advanced/use-custom-fetch-composable"}

:read-more{to="/docs/getting-started/data-fetching"}

:link-example{to="/docs/examples/features/data-fetching"}
```

## ## Params

- ``URL``: The URL to fetch.
- ``Options`` (extends `[unjs/ofetch](https://github.com/unjs/ofetch)` options & `[AsyncDataOptions](/docs/api/composables/use-async-data#params)`):
  - ``method``: Request method.
  - ``query``: Adds query search params to URL using `[ufo](https://github.com/unjs/ufo)`
  - ``params``: Alias for ``query``
  - ``body``: Request body - automatically stringified (if an object is passed).
  - ``headers``: Request headers.
  - ``baseUrl``: Base URL for the request.
  - ``timeout``: Milliseconds to automatically abort request
  - ``cache``: Handles cache control according to `[Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/fetch#cache)`
    - You can pass boolean to disable the cache or you can pass one of the following values: ``default``, ``no-store``, ``reload``, ``no-cache``, ``force-cache``, and ``only-if-cached``.

:::note

All fetch options can be given a ``computed`` or ``ref`` value. These will be watched and new requests made automatically with any new values if they are updated.

::

- ``Options`` (from `[`useAsyncData`](/docs/api/composables/use-async-data)`):
  - ``key``: a unique key to ensure that data fetching can be properly de-duplicated across requests, if not provided, it will be automatically generated based on URL and fetch options
  - ``server``: whether to fetch the data on the server (defaults to ``true``)
  - ``lazy``: whether to resolve the async function after loading the route, instead of blocking client-side navigation (defaults to ``false``)
    - ``immediate``: when set to ``false``, will prevent the request from firing immediately. (defaults to ``true``)
    - ``default``: a factory function to set the default value of the ``data``, before the async function resolves - useful with the ``lazy: true`` or ``immediate: false`` option
    - ``transform``: a function that can be used to alter ``handler`` function result after resolving
    - ``getCachedData``: Provide a function which returns cached data. A `_null_` or `_undefined_` return value will trigger a fetch. By default, this is: ``key => nuxt.isHydrating ? nuxt.payload.data[key] : nuxt.static.data[key]``, which only caches data when ``payloadExtraction`` is enabled.
    - ``pick``: only pick specified keys in this array from the ``handler`` function result
    - ``watch``: watch an array of reactive sources and auto-refresh the fetch result when they change. Fetch options and URL are watched by default. You can completely ignore reactive sources by using ``watch: false``. Together with ``immediate: false``, this allows for a fully-manual ``useFetch``. (You can [see an example here](/docs/getting-started/data-fetching#watch) of using ``watch``.)
    - ``deep``: return data in a deep ref object (it is ``true`` by default). It can be set to ``false`` to return data in a shallow ref object, which can improve performance if your data does not need to be deeply reactive.
    - ``dedupe``: avoid fetching same key more than once at a time (defaults to ``cancel``). Possible options:
      - ``cancel`` - cancels existing requests when a new one is made
      - ``defer`` - does not make new requests at all if there is a pending request

:::note

If you provide a function or ref as the ``url`` parameter, or if you provide functions as arguments to the ``options`` parameter, then the ``useFetch`` call will not match other ``useFetch`` calls elsewhere in your codebase, even if the options seem to be identical. If you wish to force a match, you may provide your own key in ``options``.

::

:::note

If you use ``useFetch`` to call an (external) HTTPS URL with a self-signed certificate in development, you will need to set ``NODE_TLS_REJECT_UNAUTHORIZED=0`` in your environment.

::

:::tip{icon="i-simple-icons-youtube" color="gray" to="https://www.youtube.com/watch?v=aQPR0xn-MMk" target="\_blank"}  
Learn how to use ``transform`` and ``getCachedData`` to avoid superfluous calls to an API and cache data for visitors on the client.  
::

## ## Return Values

- ``data``: the result of the asynchronous function that is passed in.
- ``refresh`/`execute``: a function that can be used to refresh the data returned by the ``handler`` function.
- ``error``: an error object if the data fetching failed.
- ``status``: a string indicating the status of the data request (``idle``, ``pending``, ``success``, ``error``).
- ``clear``: a function which will set ``data`` to ``undefined``, set ``error`` to ``null``, set ``status`` to ``idle``, and mark any currently pending requests as cancelled.

By default, Nuxt waits until a ``refresh`` is finished before it can be executed again.

:::note

If you have not fetched data on the server (for example, with ``server: false``), then the data will not be fetched until hydration completes. This means even if you await ``useFetch`` on client-side, ``data`` will remain null within ``<script setup>``.

```

::

Type

```ts [Signature]
function useFetch<DataT, ErrorT>(
  url: string | Request | Ref<string | Request> | () => string | Request,
  options?: UseFetchOptions<DataT>
): Promise<AsyncData<DataT, ErrorT>>

type UseFetchOptions<DataT> = {
  key?: string
  method?: string
  query?: SearchParams
  params?: SearchParams
  body?: RequestInit['body'] | Record<string, any>
  headers?: Record<string, string> | [key: string, value: string][] | Headers
  baseURL?: string
  server?: boolean
  lazy?: boolean
  immediate?: boolean
  getCachedData?: (key: string, nuxtApp: NuxtApp) => DataT
  deep?: boolean
  dedupe?: 'cancel' | 'defer'
  default?: () => DataT
  transform?: (input: DataT) => DataT | Promise<DataT>
  pick?: string[]
  watch?: WatchSource[] | false
}

type AsyncData<DataT, ErrorT> = {
  data: Ref<DataT | null>
  refresh: (opts?: AsyncDataExecuteOptions) => Promise<void>
  execute: (opts?: AsyncDataExecuteOptions) => Promise<void>
  clear: () => void
  error: Ref<ErrorT | null>
  status: Ref<AsyncDataRequestStatus>
}

interface AsyncDataExecuteOptions {
  dedupe?: 'cancel' | 'defer'
}

type AsyncDataRequestStatus = 'idle' | 'pending' | 'success' | 'error'
```

```

```

title: 'ES Modules'
description: "Nuxt uses native ES modules."

```

This guide helps explain what ES Modules are and how to make a Nuxt app (or upstream library) compatible with ESM.

## ## Background

### ### CommonJS Modules

CommonJS (CJS) is a format introduced by Node.js that allows sharing functionality between isolated JavaScript modules ([read more](https://nodejs.org/api/modules.html)).

You might be already familiar with this syntax:

```
```js
const a = require('./a')

module.exports.a = a
```
```

Bundlers like webpack and Rollup support this syntax and allow you to use modules written in CommonJS in the browser.

### ### ESM Syntax

Most of the time, when people talk about ESM vs CJS, they are talking about a different syntax for writing [modules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules).

```
```js
import a from './a'

export { a }
```

Before ECMAScript Modules (ESM) became a standard (it took more than 10 years!), tooling like [webpack](https://webpack.js.org/guides/ecma-script-modules) and even languages like TypeScript started supporting so-called **ESM syntax**.

However, there are some key differences with actual spec; here's [a helpful explainer](https://hacks.mozilla.org/2018/03/es-modules-a-cartoon-deep-dive).

What is 'Native' ESM?

You may have been writing your app using ESM syntax for a long time. After all, it's natively supported by the browser, and in Nuxt 2 we compiled all the code you wrote to the appropriate format (CJS for server, ESM for browser).

When adding modules to your package, things were a little different. A sample library might expose both CJS and ESM versions, and let us pick which one we wanted:

```
```json
{
 "name": "sample-library",
 "main": "dist/sample-library.cjs.js",
 "module": "dist/sample-library.esm.js"
}
```

So in Nuxt 2, the bundler (webpack) would pull in the CJS file ('main') for the server build and use the ESM file ('module') for the client build.

However, in recent Node.js LTS releases, it is now possible to [use native ESM module](https://nodejs.org/api/esm.html) within Node.js. That means that Node.js itself can process JavaScript using ESM syntax, although it doesn't do it by default. The two most common ways to enable ESM syntax are:

- set `"type": "module"` within your `package.json` and keep using `.js` extension
- use the `.mjs` file extensions (recommended)

This is what we do for Nuxt Nitro; we output a `.output/server/index.mjs` file. That tells Node.js to treat this file as a native ES module.

### ### What Are Valid Imports in a Node.js Context?

When you `import` a module rather than `require` it, Node.js resolves it differently. For example, when you `import 'sample-library'`, Node.js will look not for the `main` but for the `exports` or `module` entry in that library's `package.json`.

This is also true of dynamic imports, like `const b = await import('sample-library')`.

Node supports the following kinds of imports (see [docs](https://nodejs.org/api/packages.html#determining-module-system)):

1. files ending in `.mjs` - these are expected to use ESM syntax
1. files ending in `.cjs` - these are expected to use CJS syntax
1. files ending in `.js` - these are expected to use CJS syntax unless their `package.json` has `"type": "module"`

### ### What Kinds of Problems Can There Be?

For a long time module authors have been producing ESM-syntax builds but using conventions like ``.esm.js`` or ``.es.js``, which they have added to the `module`` field in their `package.json``. This hasn't been a problem until now because they have only been used by bundlers like webpack, which don't especially care about the file extension.

However, if you try to import a package with an ``.esm.js`` file in a Node.js ESM context, it won't work, and you'll get an error like:

```
```bash [Terminal]
(node:22145) Warning: To load an ES module, set "type": "module" in the package.json or use the .mjs extension.
/path/to/index.js:1
```

```
export default {}
^^^^^^
```

```
SyntaxError: Unexpected token 'export'
    at wrapSafe (internal/modules/cjs/loader.js:1001:16)
    at Module._compile (internal/modules/cjs/loader.js:1049:27)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    ....
    at async Object.loadESM (internal/process/esm_loader.js:68:5)
...`
```

You might also get this error if you have a named import from an ESM-syntax build that Node.js thinks is CJS:

```
```bash [Terminal]
file:///path/to/index.mjs:5
import { named } from 'sample-library'
 ^^^^^
```

```
SyntaxError: Named export 'named' not found. The requested module 'sample-library' is a CommonJS module, which may not support all module.exports as named exports.
```

CommonJS modules can always be imported via the default export, for example using:

```
import pkg from 'sample-library';
const { named } = pkg;

 at ModuleJob._instantiate (internal/modules/esm/module_job.js:120:21)
 at async ModuleJob.run (internal/modules/esm/module_job.js:165:5)
 at async Loader.import (internal/modules/esm/loader.js:177:24)
 at async Object.loadESM (internal/process/esm_loader.js:68:5)
...`
```

## ## Troubleshooting ESM Issues

If you encounter these errors, the issue is almost certainly with the upstream library. They need to [fix their library] (#library-author-guide) to support being imported by Node.

## ### Transpiling Libraries

In the meantime, you can tell Nuxt not to try to import these libraries by adding them to `build.transpile``:

```
```ts twoslash
export default defineNuxtConfig({
  build: {
    transpile: ['sample-library']
  }
})
...`
```

You may find that you also need to add other packages that are being imported by these libraries.

Aliasing Libraries

In some cases, you may also need to manually alias the library to the CJS version, for example:

```
```ts twoslash
export default defineNuxtConfig({
 alias: {
 'sample-library': 'sample-library/dist/sample-library.cjs.js'
 }
})
...`
```

## ### Default Exports

A dependency with CommonJS format, can use `module.exports`` or `exports`` to provide a default export:

```
```js [node_modules/cjs-pkg/index.js]
module.exports = { test: 123 }
// or
exports.test = 123
...`
```

This normally works well if we `require` such dependency:

```
```js [test.cjs]
const pkg = require('cjs-pkg')

console.log(pkg) // { test: 123 }
```
```

[Node.js in native ESM mode](https://nodejs.org/api/esm.html#interoperability-with-commonjs), [typescript with `esModuleInterop` enabled](https://www.typescriptlang.org/tsconfig#esModuleInterop) and bundlers such as webpack, provide a compatibility mechanism so that we can default import such library. This mechanism is often referred to as "interop require default":

```
```js
import pkg from 'cjs-pkg'

console.log(pkg) // { test: 123 }
```
```

However, because of the complexities of syntax detection and different bundle formats, there is always a chance that the interop default fails and we end up with something like this:

```
```js
import pkg from 'cjs-pkg'

console.log(pkg) // { default: { test: 123 } }
```
```

Also when using dynamic import syntax (in both CJS and ESM files), we always have this situation:

```
```js
import('cjs-pkg').then(console.log) // [Module: null prototype] { default: { test: '123' } }
```

In this case, we need to manually interop the default export:

```
```js
// Static import
import { default as pkg } from 'cjs-pkg'

// Dynamic import
import('cjs-pkg').then(m => m.default || m).then(console.log)
```
```

For handling more complex situations and more safety, we recommend and internally use [mlly](https://github.com/unjs/mlly) in Nuxt that can preserve named exports.

```
```js
import { interopDefault } from 'mlly'

// Assuming the shape is { default: { foo: 'bar' }, baz: 'qux' }
import myModule from 'my-module'

console.log(interopDefault(myModule)) // { foo: 'bar', baz: 'qux' }
```
```

## ## Library Author Guide

The good news is that it's relatively simple to fix issues of ESM compatibility. There are two main options:

1. **You can rename your ESM files to end with `.mjs`.**

This is the recommended and simplest approach. You may have to sort out issues with your library's dependencies and possibly with your build system, but in most cases, this should fix the problem for you. It's also recommended to rename your CJS files to end with `.cjs`, for the greatest explicitness.

1. **You can opt to make your entire library ESM-only.**

This would mean setting `"type": "module"` in your `package.json` and ensuring that your built library uses ESM syntax. However, you may face issues with your dependencies - and this approach means your library can only be consumed in an ESM context.

## ### Migration

The initial step from CJS to ESM is updating any usage of `require` to use `import` instead:

::code-group

```
```js [Before]
module.exports = ...

exports.hello = ...
```

```
...
```

```
```js [After]
export default ...
```

```
export const hello = ...
```
```

```
::
```

```
:::code-group
```

```
```js [Before]
const myLib = require('my-lib')
```
```

```
```js [After]
import myLib from 'my-lib'
// or
const myLib = await import('my-lib').then(lib => lib.default || lib)
```
```

```
::
```

In ESM Modules, unlike CJS, ``require``, ``require.resolve``, ``__filename`` and ``__dirname`` globals are not available and should be replaced with ``import()`` and ``import.meta.filename``.

```
:::code-group
```

```
```js [Before]
import { join } from 'path'
```

```
const newDir = join(__dirname, 'new-dir')
```
```

```
```js [After]
import { fileURLToPath } from 'node:url'
```

```
const newDir = fileURLToPath(new URL('./new-dir', import.meta.url))
```
```

```
::
```

```
:::code-group
```

```
```js [Before]
const someFile = require.resolve('./lib/foo.js')
```

```
```js [After]
import { resolvePath } from 'milly'
```

```
const someFile = await resolvePath('my-lib', { url: import.meta.url })
```
```

```
::
```

### ### Best Practices

- Prefer named exports rather than default export. This helps reduce CJS conflicts. (see [Default exports](#default-exports) section)
- Avoid depending on Node.js built-ins and CommonJS or Node.js-only dependencies as much as possible to make your library usable in Browsers and Edge Workers without needing Nitro polyfills.
- Use new ``exports`` field with conditional exports. ([read more](https://nodejs.org/api/packages.html#conditional-exports)).

```
```json
{
  "exports": {
    ".": {
      "import": "./dist/mymodule.mjs"
    }
  }
}
```
```



```

title: Configuration
description: Nuxt is configured with sensible defaults to make you productive.
navigation.icon: i-ph-gear-duotone

```

By default, Nuxt is configured to cover most use cases. The `[`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config)` file can override or extend this default configuration.

## ## Nuxt Configuration

The `[`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config)` file is located at the root of a Nuxt project and can override or extend the application's behavior.

A minimal configuration file exports the ``defineNuxtConfig`` function containing an object with your configuration. The ``defineNuxtConfig`` helper is globally available without import.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  // My Nuxt config
})
---
```

This file will often be mentioned in the documentation, for example to add custom scripts, register modules or change rendering modes.

```
::read-more{to="/docs/api/configuration/nuxt-config"}
Every option is described in the Configuration Reference.
::
```

```
::note
You don't have to use TypeScript to build an application with Nuxt. However, it is strongly recommended to use the `.ts` extension for the `nuxt.config` file. This way you can benefit from hints in your IDE to avoid typos and mistakes while editing your configuration.
::
```

Environment overrides

You can configure fully typed, per-environment overrides in your `nuxt.config`

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 $production: {
 routeRules: {
 '/*': { isr: true }
 }
 },
 $development: {
 //
 }
})

```

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=DFZI2iVCrNc" target="_blank"}
Watch a video from Alexander Lichter about the env-aware `nuxt.config.ts`.
::
```

```
::note
If you're authoring layers, you can also use the `$meta` key to provide metadata that you or the consumers of your layer might use.
::
```

## ### Environment Variables and Private Tokens

The ``runtimeConfig`` API exposes values like environment variables to the rest of your application. By default, these keys are only available server-side. The keys within ``runtimeConfig.public`` are also available client-side.

Those values should be defined in ``nuxt.config`` and can be overridden using environment variables.

```
::code-group
```

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  runtimeConfig: {
    // The private keys which are only available server-side
    apiSecret: '123',
    // Keys within public are also exposed client-side
    public: {
      apiBase: '/api'
    }
  }
})
```

```

```
```bash [.env]
# This will override the value of apiSecret
Nuxt_API_SECRET=api_secret_token
```
```

::

These variables are exposed to the rest of your application using the [`useRuntimeConfig()`](/docs/api/composables/use-runtime-config) composable.

```
```vue [pages/index.vue]
<script setup lang="ts">
const runtimeConfig = useRuntimeConfig()
</script>
```
```

:read-more{to="/docs/guide/going-further/runtime-config"}

## App Configuration

The `app.config.ts` file, located in the source directory (by default the root of the project), is used to expose public variables that can be determined at build time. Contrary to the `runtimeConfig` option, these can not be overridden using environment variables.

A minimal configuration file exports the `defineAppConfig` function containing an object with your configuration. The `defineAppConfig` helper is globally available without import.

```
```ts [app.config.ts]
export default defineAppConfig({
  title: 'Hello Nuxt',
  theme: {
    dark: true,
    colors: {
      primary: '#ff0000'
    }
  }
})
```
```

These variables are exposed to the rest of your application using the [`useAppConfig`](/docs/api/composables/use-app-config) composable.

```
```vue [pages/index.vue]
<script setup lang="ts">
const appConfig = useAppConfig()
</script>
```
```

:read-more{to="/docs/guide/directory-structure/app-config"}

## `runtimeConfig` vs `app.config`

As stated above, `runtimeConfig` and `app.config` are both used to expose variables to the rest of your application. To determine whether you should use one or the other, here are some guidelines:

- `runtimeConfig`: Private or public tokens that need to be specified after build using environment variables.
- `app.config`: Public tokens that are determined at build time, website configuration such as theme variant, title and any project config that are not sensitive.

| Feature                   | <code>runtimeConfig</code> | <code>app.config</code> |
|---------------------------|----------------------------|-------------------------|
| Client Side               | Hydrated                   | Bundled                 |
| Environment Variables     | â€¦ Yes                    | â€¢ No                  |
| Reactive                  | â€¦ Yes                    | â€¦ Yes                 |
| Types support             | â€¦ Partial                | â€¦ Yes                 |
| Configuration per Request | â€¢ No                     | â€¦ Yes                 |
| Hot Module Replacement    | â€¢ No                     | â€¦ Yes                 |
| Non primitive JS types    | â€¢ No                     | â€¦ Yes                 |

## External Configuration Files

Nuxt uses [`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config) file as the single source of truth for configurations and skips reading external configuration files. During the course of building your project, you may have a need to configure those. The following table highlights common configurations and, where applicable, how they can be configured with Nuxt.

| Name                           | Config File             | How To Configure                                                                        |
|--------------------------------|-------------------------|-----------------------------------------------------------------------------------------|
| [Nitro](https://nitro.unjs.io) | ~~`nitro.config.ts`~~   | Use [ <code>nitro</code> ](/docs/api/nuxt-config#nitro) key in <code>nuxt.config</code> |
| [PostCSS](https://postcss.org) | ~~`postcss.config.js`~~ | Use [ <code>postcss</code> ](/docs/api/nuxt-config#postcss) key                         |

```
in `nuxt.config`
[Vite](https://vitejs.dev) | ~~`vite.config.ts`~~ | Use [`vite`](/docs/api/nuxt-config#vite) key in
`nuxt.config`
[webpack](https://webpack.js.org) | ~~`webpack.config.ts`~~ | Use [`webpack`](/docs/api/nuxt-config#webpack-1)
key in `nuxt.config`
```

Here is a list of other common config files:

| Name                                                                                            | Config File          | How To Configure                                       |
|-------------------------------------------------------------------------------------------------|----------------------|--------------------------------------------------------|
| [TypeScript](https://www.typescriptlang.org) (/docs/guide/concepts/typescript#nuxttsconfigjson) | `tsconfig.json`      | [More Info]                                            |
| [ESLint](https://eslint.org) (https://eslint.org/docs/latest/use/configure/configuration-files) | `eslint.config.js`   | [More Info]                                            |
| [Prettier](https://prettier.io) (https://prettier.io/docs/en/configuration.html)                | `.prettierrc.json`   | [More Info]                                            |
| [Stylelint](https://stylelint.io) (https://stylelint.io/user-guide/configure)                   | `.stylelintrc.json`  | [More Info](https://stylelint.io/user-guide/configure) |
| [TailwindCSS](https://tailwindcss.com) (https://tailwindcss.nuxtjs.org/tailwind/config)         | `tailwind.config.js` | [More Info]                                            |
| [Vitest](https://vitest.dev)                                                                    | `vitest.config.ts`   | [More Info](https://vitest.dev/config)                 |

## Vue Configuration

### With Vite

If you need to pass options to `@vitejs/plugin-vue`` or `@vitejs/plugin-vue-jsx``, you can do this in your `nuxt.config`` file.

- ``vite.vue`` for `@vitejs/plugin-vue``. Check available options [here](https://github.com/vitejs/vite-plugin-vue/tree/main/packages/plugin-vue).
- ``vite.vueJsx`` for `@vitejs/plugin-vue-jsx``. Check available options [here](https://github.com/vitejs/vite-plugin-vue/tree/main/packages/plugin-vue-jsx).

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 vite: {
 vue: {
 customElement: true
 },
 vueJsx: {
 mergeProps: true
 }
 }
})
...

:read-more{to="/docs/api/configuration/nuxt-config#vue"}
```

### With webpack

If you use webpack and need to configure ``vue-loader``, you can do this using `webpack.loaders.vue`` key inside your `nuxt.config`` file. The available options are [defined here](https://github.com/vuejs/vue-loader/blob/main/src/index.ts#L32-L62).

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 webpack: {
 loaders: {
 vue: {
 hotReload: true,
 }
 }
 }
})
...

:read-more{to="/docs/api/configuration/nuxt-config#loaders"}
```

### Enabling Experimental Vue Features

You may need to enable experimental features in Vue, such as ``propsDestructure``. Nuxt provides an easy way to do that in `nuxt.config.ts``, no matter which builder you are using:

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 vue: {
 propsDestructure: true
 }
})
...

experimental `reactivityTransform` migration from Vue 3.4 and Nuxt 3.9
```

Since Nuxt 3.9 and Vue 3.4, ``reactivityTransform`` has been moved from `Vue` to `Vue Macros` which has a [Nuxt integration] (<https://vue-macros.dev/guide/nuxt-integration.html>).

`:read-more{to="/docs/api/configuration/nuxt-config#vue-1"}`

```

title: Upgrade Guide
description: 'Learn how to upgrade to the latest Nuxt version.'
navigation.icon: i-ph-arrow-circle-up-duotone

```

## ## Upgrading Nuxt

### ### Latest release

To upgrade Nuxt to the [latest release](https://github.com/nuxt/nuxt/releases), use the `nuxi upgrade` command.

```
::code-group
```

```
```bash [npm]
npx nuxi upgrade
```
```

```
```bash [yarn]
yarn dlx nuxi upgrade
```
```

```
```bash [pnpm]
pnpm dlx nuxi upgrade
```
```

```
```bash [bun]
bun x nuxi upgrade
```
```

```
::
```

### ### Nightly Release Channel

To use the latest Nuxt build and test features before their release, read about the [nightly release channel](/docs/guide/going-further/nightly-release-channel) guide.

```
::alert{type="warning"}
The nightly release channel `latest` tag is currently tracking the Nuxt v4 branch, meaning that it is particularly likely to have breaking changes right now - be careful!
```

You can opt in to the 3.x branch nightly releases with `"nuxt": "npm:nuxt-nightly@3x"`.

```
::
```

## ## Testing Nuxt 4

Nuxt 4 is planned to be released **on or before June 14** (though obviously this is dependent on having enough time after Nitro's major release to be properly tested in the community, so be aware that this is not an exact date).

Until then, it is possible to test many of Nuxt 4's breaking changes from Nuxt version 3.12+.

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=r4wFKlcJK6c" target="_blank"}
Watch a video from Alexander Lichter showing how to opt in to Nuxt 4's breaking changes already.
::
```

### ### Opting in to Nuxt 4

First, upgrade Nuxt to the [latest release](https://github.com/nuxt/nuxt/releases).

Then you can set your `compatibilityVersion` to match Nuxt 4 behavior:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  future: {
    compatibilityVersion: 4,
  },
  // To re-enable _all_ Nuxt v3 behavior, set the following options:
  // srcDir: '.',
  // dir: {
  //   app: 'app'
  // },
  // experimental: {
  //   sharedPrerenderData: false,
  //   compileTemplate: true,
  //   resetAsyncDataToUndefined: true,
  //   templateUtils: true,
  //   relativeWatchPaths: true,
  //   defaults: {
  //     useAsyncData: {
  //       deep: true
  //     }
  //   }
  // }
})
```
```

```

// },
// unhead: {
// renderSSRHeadOptions: {
// omitLineBreaks: false
// }
// }
// }
})
...

```

When you set your `compatibilityVersion` to `4`, defaults throughout your Nuxt configuration will change to opt in to Nuxt v4 behavior, but you can granularly re-enable Nuxt v3 behavior when testing, following the commented out lines above. Please file issues if so, so that we can address them in Nuxt or in the ecosystem.

### ### Migrating to Nuxt 4

Breaking or significant changes will be noted here along with migration steps for backward/forward compatibility.

```

::alert
This section is subject to change until the final release, so please check back here regularly if you are testing Nuxt 4
using `compatibilityVersion: 4`.
::

```

### #### Migrating Using Codemods

To facilitate the upgrade process, we have collaborated with the [Codemod](https://github.com/codemod-com/codemod) team to automate many migration steps with some open-source codemods.

```

::note
If you encounter any issues, please report them to the Codemod team with `npx codemod feedback` ðŸ™
::

```

For a complete list of Nuxt 4 codemods, detailed information on each, their source, and various ways to run them, visit the [Codemod Registry](https://go.codemod.com/codemod-registry).

You can run all the codemods mentioned in this guide using the following `codemod` recipe:

```

```bash
npx codemod@latest nuxt/4/migration-recipe
```

```

This command will execute all codemods in sequence, with the option to deselect any that you do not wish to run. Each codemod is also listed below alongside its respective change and can be executed independently.

### #### New Directory Structure

ðŸš! **\*\*Impact Level\*\***: Significant

Nuxt now defaults to a new directory structure, with backwards compatibility (so if Nuxt detects you are using the old structure, such as with a top-level `pages/` directory, this new structure will not apply).

ðŸ’% [See full RFC](https://github.com/nuxt/nuxt/issues/26444)

### #### What Changed

- \* the new Nuxt default `srcDir` is `app/` by default, and most things are resolved from there.
- \* `serverDir` now defaults to `/server` rather than `/server`
- \* `layers/`, `modules/` and `public/` are resolved relative to `` by default
- \* if using [Nuxt Content v2.13+](https://github.com/nuxt/content/pull/2649), `content/` is resolved relative to ``
- \* a new `dir.app` is added, which is the directory we look for `router.options.ts` and `spa-loading-template.html` - this defaults to `/`

<details>

<summary>An example v4 folder structure.</summary>

```

```sh
.output/
.nuxt/
app/
  assets/
  components/
  composables/
  layouts/
  middleware/
  pages/
  plugins/
  utils/
  app.config.ts
  app.vue
  router.options.ts
content/
layers/
modules/

```

```
node_modules/  
public/  
server/  
  api/  
  middleware/  
  plugins/  
  routes/  
  utils/  
nuxt.config.ts  
```
```

</details>

ðŸ’% For more details, see the [PR implementing this change](https://github.com/nuxt/nuxt/pull/27029).

##### Reasons for Change

- 1. **Performance** - placing all your code in the root of your repo causes issues with `.git/` and `node_modules/` folders being scanned/included by FS watchers which can significantly delay startup on non-Mac OSes.
- 1. **IDE type-safety** - `server/` and the rest of your app are running in two entirely different contexts with different global imports available, and making sure `server/` isn't `_inside_` the same folder as the rest of your app is a big first step to ensuring you get good auto-completes in your IDE.

##### Migration Steps

- 1. Create a new directory called `app/`.
- 1. Move your `assets/`, `components/`, `composables/`, `layouts/`, `middleware/`, `pages/`, `plugins/` and `utils/` folders under it, as well as `app.vue`, `error.vue`, `app.config.ts`. If you have an `app/router-options.ts` or `app/spa-loading-template.html`, these paths remain the same.
- 1. Make sure your `nuxt.config.ts`, `content/`, `layers/`, `modules/`, `public/` and `server/` folders remain outside the `app/` folder, in the root of your project.

::tip  
You can automate this migration by running `npx codemod@latest nuxt/4/file-structure`  
::

However, migration is not required. If you wish to keep your current folder structure, Nuxt should auto-detect it. (If it does not, please raise an issue.) The one exception is that if you already have a custom `srcDir`. In this case, you should be aware that your `modules/`, `public/` and `server/` folders will be resolved from your `rootDir` rather than from your custom `srcDir`. You can override this by configuring `dir.modules`, `dir.public` and `serverDir` if you need to.

You can also force a v3 folder structure with the following configuration:

```
```ts [nuxt.config.ts]  
export default defineNuxtConfig({  
  // This reverts the new srcDir default from `app` back to your root directory  
  srcDir: '.',  
  // This specifies the directory prefix for `app/router.options.ts` and `app/spa-loading-template.html`  
  dir: {  
    app: 'app'  
  }  
})  
```
```

#### Shared Prerender Data

ðŸŠ! **Impact Level**: Medium

##### What Changed

We enabled a previously experimental feature to share data from `useAsyncData` and `useFetch` calls, across different pages. See [original PR](https://github.com/nuxt/nuxt/pull/24894).

##### Reasons for Change

This feature automatically shares payload `_data_` between pages that are prerendered. This can result in a significant performance improvement when prerendering sites that use `useAsyncData` or `useFetch` and fetch the same data in different pages.

For example, if your site requires a `useFetch` call for every page (for example, to get navigation data for a menu, or site settings from a CMS), this data would only be fetched once when prerendering the first page that uses it, and then cached for use when prerendering other pages.

##### Migration Steps

Make sure that any unique key of your data is always resolvable to the same data. For example, if you are using `useAsyncData` to fetch data related to a particular page, you should provide a key that uniquely matches that data. (`useFetch` should do this automatically for you.)

```
```ts [app/pages/test/[slug]].vue  
// This would be unsafe in a dynamic page (e.g. `[slug].vue`) because the route slug makes a difference  
// to the data fetched, but Nuxt can't know that because it's not reflected in the key.  
const route = useRoute()
```

```
const { data } = await useAsyncData(async () => {
  return await $fetch(`/api/my-page/${route.params.slug}`)
})
// Instead, you should use a key that uniquely identifies the data fetched.
const { data } = await useAsyncData(route.params.slug, async () => {
  return await $fetch(`/api/my-page/${route.params.slug}`)
})
...
```

Alternatively, you can disable this feature with:

```
``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  experimental: {
    sharedPrerenderData: false
  }
})
...
```

Default `data` and `error` values in `useAsyncData` and `useFetch`

ðŸš! ****Impact Level****: Minimal

What Changed

`data` and `error` objects returned from `useAsyncData` will now default to `undefined`.

Reasons for Change

Previously `data` was initialized to `null` but reset in `clearNuxtData` to `undefined`. `error` was initialized to `null`. This change is to bring greater consistency.

Migration Steps

If you were checking if `data.value` or `error.value` were `null`, you can update these checks to check for `undefined` instead.

```
::tip
You can automate this step by running `npx codemod@latest nuxt/4/default-data-error-value`
::
```

If you encounter any issues you can revert back to the previous behavior with:

```
``ts twoslash [nuxt.config.ts]
// @errors: 2353
export default defineNuxtConfig({
  experimental: {
    defaults: {
      useAsyncData: {
        value: 'null',
        errorValue: 'null'
      }
    }
  }
})
...
```

Please report an issue if you are doing this, as we do not plan to keep this as configurable.

Removal of deprecated `boolean` values for `dedupe` option when calling `refresh` in `useAsyncData` and `useFetch`

ðŸš! ****Impact Level****: Minimal

What Changed

Previously it was possible to pass `dedupe: boolean` to `refresh`. These were aliases of `cancel` (`true`) and `defer` (`false`).

```
``ts twoslash [app.vue]
// @errors: 2322
const { refresh } = await useAsyncData(async () => ({ message: 'Hello, Nuxt 3!' }))

async function refreshData () {
  await refresh({ dedupe: true })
}
...
```

Reasons for Change

These aliases were removed, for greater clarity.

The issue came up when adding `dedupe` as an option to `useAsyncData`, and we removed the boolean values as they ended up being `_opposites_`.

`refresh({ dedupe: false })` meant 'do not `_cancel_` existing requests in favour of this new one'. But passing ``dedupe: true`` within the options of ``useAsyncData`` means 'do not make any new requests if there is an existing pending request.' (See [PR] (<https://github.com/nuxt/nuxt/pull/24564#pullrequestreview-1764584361>)).

Migration Steps

The migration should be straightforward:

```
``diff
  const { refresh } = await useAsyncData(async () => ({ message: 'Hello, Nuxt 3!' }))

  async function refreshData () {
-    await refresh({ dedupe: true })
+    await refresh({ dedupe: 'cancel' })

-    await refresh({ dedupe: false })
+    await refresh({ dedupe: 'defer' })
  }
``
```

::tip
You can automate this step by running ``npx codemod@latest nuxt/4/deprecated-dedupe-value``
::

Respect defaults when clearing ``data`` in ``useAsyncData`` and ``useFetch``

ðŸŒˆ| ****Impact Level****: Minimal

What Changed

If you provide a custom ``default`` value for ``useAsyncData``, this will now be used when calling ``clear`` or ``clearNuxtData`` and it will be reset to its default value rather than simply unset.

Reasons for Change

Often users set an appropriately empty value, such as an empty array, to avoid the need to check for ``null``/``undefined`` when iterating over it. This should be respected when resetting/clearing the data.

Migration Steps

If you encounter any issues you can revert back to the previous behavior, for now, with:

```
``ts twoslash [nuxt.config.ts]
// @errors: 2353
export default defineNuxtConfig({
  experimental: {
    resetAsyncDataToUndefined: true,
  }
})
``
```

Please report an issue if you are doing so, as we do not plan to keep this as configurable.

Shallow Data Reactivity in ``useAsyncData`` and ``useFetch``

ðŸŒˆ| ****Impact Level****: Minimal

The ``data`` object returned from ``useAsyncData``, ``useFetch``, ``useLazyAsyncData`` and ``useLazyFetch`` is now a ``shallowRef`` rather than a ``ref``.

What Changed

When new data is fetched, anything depending on ``data`` will still be reactive because the entire object is replaced. But if your code changes a property `_within_` that data structure, this will not trigger any reactivity in your app.

Reasons for Change

This brings a ****significant**** performance improvement for deeply nested objects and arrays because Vue does not need to watch every single property/array for modification. In most cases, ``data`` should also be immutable.

Migration Steps

In most cases, no migration steps are required, but if you rely on the reactivity of the data object then you have two options:

1. You can granularly opt in to deep reactivity on a per-composable basis:
``diff
- const { data } = useFetch('/api/test')
+ const { data } = useFetch('/api/test', { deep: true })
``
1. You can change the default behavior on a project-wide basis (not recommended):
``ts twoslash [nuxt.config.ts]

```
export default defineNuxtConfig({
  experimental: {
    defaults: {
      useAsyncData: {
        deep: true
      }
    }
  }
})
````
```

:::tip

If you need to, you can automate this step by running `npx codemod@latest nuxt/4/shallow-function-reactivity`  
::

#### #### Absolute Watch Paths in `builder:watch`

ðŸš! **\*\*Impact Level\*\***: Minimal

##### ##### What Changed

The Nuxt `builder:watch` hook now emits a path which is absolute rather than relative to your project `srcDir`.

##### ##### Reasons for Change

This allows us to support watching paths which are outside your `srcDir`, and offers better support for layers and other more complex patterns.

##### ##### Migration Steps

We have already proactively migrated the public Nuxt modules which we are aware use this hook. See [issue #25339](https://github.com/nuxt/nuxt/issues/25339).

However, if you are a module author using the `builder:watch` hook and wishing to remain backwards/forwards compatible, you can use the following code to ensure that your code works the same in both Nuxt v3 and Nuxt v4:

```
``diff
+ import { relative, resolve } from 'node:fs'
 // ...
 nuxt.hook('builder:watch', async (event, path) => {
+ path = relative(nuxt.options.srcDir, resolve(nuxt.options.srcDir, path))
 // ...
 })
````
```

:::tip

You can automate this step by running `npx codemod@latest nuxt/4/absolute-watch-path`
::

Removal of `window.__NUXT__` object

What Changed

We are removing the global `window.__NUXT__` object after the app finishes hydration.

Reasons for Change

This opens the way to multi-app patterns ([#21635](https://github.com/nuxt/nuxt/issues/21635)) and enables us to focus on a single way to access Nuxt app data - `useNuxtApp()`.

Migration Steps

The data is still available, but can be accessed with `useNuxtApp().payload`:

```
``diff
- console.log(window.__NUXT__)
+ console.log(useNuxtApp().payload)
````
```

#### #### Directory index scanning

ðŸš! **\*\*Impact Level\*\***: Medium

##### ##### What Changed

Child folders in your `middleware/` folder are also scanned for `index` files and these are now also registered as middleware in your project.

##### ##### Reasons for Change

Nuxt scans a number of folders automatically, including `middleware/` and `plugins/`.

Child folders in your `plugins/` folder are scanned for `index` files and we wanted to make this behavior consistent between

scanned directories.

#### ##### Migration Steps

Probably no migration is necessary but if you wish to revert to previous behavior you can add a hook to filter out these middleware:

```
``ts
export default defineNuxtConfig({
 hooks: {
 'app:resolve'(app) {
 app.middleware = app.middleware.filter(mw => !/\index\.[^/]+\$/ .test(mw.path))
 }
 }
})
``
```

#### #### Template Compilation Changes

ðŸŒˆ| **\*\*Impact Level\*\***: Minimal

#### ##### What Changed

Previously, Nuxt used ``lodash/template`` to compile templates located on the file system using the ``.ejs`` file format/syntax.

In addition, we provided some template utilities (``serialize``, ``importName``, ``importSources``) which could be used for code-generation within these templates, which are now being removed.

#### ##### Reasons for Change

In Nuxt v3 we moved to a `'virtual'` syntax with a ``getContents()`` function which is much more flexible and performant.

In addition, ``lodash/template`` has had a succession of security issues. These do not really apply to Nuxt projects because it is being used at build-time, not runtime, and by trusted code. However, they still appear in security audits. Moreover, ``lodash`` is a hefty dependency and is unused by most projects.

Finally, providing code serialization functions directly within Nuxt is not ideal. Instead, we maintain projects like [unjs/knitwork](http://github.com/unjs/knitwork) which can be dependencies of your project, and where security issues can be reported/resolved directly without requiring an upgrade of Nuxt itself.

#### ##### Migration Steps

We have raised PRs to update modules using EJS syntax, but if you need to do this yourself, you have three backwards/forwards-compatible alternatives:

- \* Moving your string interpolation logic directly into ``getContents()``.
- \* Using a custom function to handle the replacement, such as in <https://github.com/nuxt-modules/color-mode/pull/240>.
- \* Continuing to use ``lodash``, as a dependency of `_your_` project rather than Nuxt:

```
``diff
+ import { readFileSync } from 'node:fs'
+ import { template } from 'lodash-es'
+ // ...
+ addTemplate({
+ fileName: 'appinsights-vue.js'
+ options: { /* some options */ },
+ src: resolver.resolve('./runtime/plugin.ejs'),
+ getContents({ options }) {
+ const contents = readFileSync(resolver.resolve('./runtime/plugin.ejs'), 'utf-8')
+ return template(contents)({ options })
+ },
+ })
``
```

Finally, if you are using the template utilities (``serialize``, ``importName``, ``importSources``), you can replace them as follows with utilities from ``knitwork``:

```
``ts
import { genDynamicImport, genImport, genSafeVariableName } from 'knitwork'

const serialize = (data: any) => JSON.stringify(data, null, 2).replace(/"{{(.+)}}"(=?,$$)/gm, r =>
JSON.parse(r).replace(/^(.*)$$/, '$1'))

const importSources = (sources: string | string[], { lazy = false } = {}) => {
 return toArray(sources).map((src) => {
 if (lazy) {
 return `const ${genSafeVariableName(src)} = ${genDynamicImport(src, { comment: `webpackChunkName:
${JSON.stringify(src)}` })}`
 }
 return genImport(src, genSafeVariableName(src))
 }).join('\n')
}
```

```
const importName = genSafeVariableName
...

::tip
You can automate this step by running `npx codemod@latest nuxt/4/template-compilation-changes`
::
```

#### Removal of Experimental Features

ðŸŠ! **\*\*Impact Level\*\***: Minimal

##### What Changed

Four experimental features are no longer configurable in Nuxt 4:

- \* `experimental.treeshakeClientOnly` will be `true` (default since v3.0)
- \* `experimental.configSchema` will be `true` (default since v3.3)
- \* `experimental.polyfillVueUseHead` will be `false` (default since v3.4)
- \* `experimental.respectNoSSRHeader` will be `false` (default since v3.4)
- \* `vite.devBundler` is no longer configurable - it will use `vite-node` by default

##### Reasons for Change

These options have been set to their current values for some time and we do not have a reason to believe that they need to remain configurable.

##### Migration Steps

- \* `polyfillVueUseHead` is implementable in user-land with [this plugin](https://github.com/nuxt/nuxt/blob/f209158352b09d1986aa320e29ff36353b91c358/packages/nuxt/src/head/runtime/plugins/vueuse-head-polyfill.ts#L10-L11)
- \* `respectNoSSRHeader` is implementable in user-land with [server middleware](https://github.com/nuxt/nuxt/blob/c660b39447f0d5b8790c0826092638d321cd6821/packages/nuxt/src/core/runtime/nitro/no-ssr.ts#L8-L9)

## Nuxt 2 vs Nuxt 3+

In the table below, there is a quick comparison between 3 versions of Nuxt:

| Feature / Version       | Nuxt 2         | Nuxt Bridge  | Nuxt 3+      |
|-------------------------|----------------|--------------|--------------|
| Vue                     | 2              | 2            | 3            |
| Stability               | ðŸŠ Stable     | ðŸŠ Stable   | ðŸŠ Stable   |
| Performance             | ðŸŽ Fast       | â€ˆi, Faster | ðŸŠ€ Fastest |
| Nitro Engine            | â€             | â€           | â€           |
| ESM support             | ðŸ€™ Partial   | ðŸ’ Better   | â€           |
| TypeScript              | â€ˆi, Opt-in   | ðŸŠŠ Partial | â€           |
| Composition API         | â€             | ðŸŠŠ Partial | â€           |
| Options API             | â€             | â€           | â€           |
| Components Auto Import  | â€             | â€           | â€           |
| `<script setup>` syntax | â€             | ðŸŠŠ Partial | â€           |
| Auto Imports            | â€             | â€           | â€           |
| webpack                 | 4              | 4            | 5            |
| Vite                    | â€Š i, Partial | ðŸŠŠ Partial | â€           |
| Nuxi CLI                | â€ Old         | â€ nuxi      | â€ nuxi      |
| Static sites            | â€             | â€           | â€           |

## Nuxt 2 to Nuxt 3+

The migration guide provides a step-by-step comparison of Nuxt 2 features to Nuxt 3+ features and guidance to adapt your current application.

```
::read-more{to="/docs/migration/overview"}
Check out the **guide to migrating from Nuxt 2 to Nuxt 3**.
::
```

## Nuxt 2 to Nuxt Bridge

If you prefer to progressively migrate your Nuxt 2 application to Nuxt 3, you can use Nuxt Bridge. Nuxt Bridge is a compatibility layer that allows you to use Nuxt 3+ features in Nuxt 2 with an opt-in mechanism.

```
::read-more{to="/docs/bridge/overview"}
Migrate from Nuxt 2 to Nuxt Bridge
::
```

```

title: "Runtime Config"
description: "Nuxt provides a runtime config API to expose configuration and secrets within your application."

```

## ## Exposing

To expose config and environment variables to the rest of your app, you will need to define runtime configuration in your `[`nuxt.config`](/docs/guide/directory-structure/nuxt-config)` file, using the `[`runtimeConfig`](/docs/api/nuxt-config#runtimeconfig)` option.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 runtimeConfig: {
 // The private keys which are only available within server-side
 apiSecret: '123',
 // Keys within public, will be also exposed to the client-side
 public: {
 apiBase: '/api'
 }
 }
})
``
```

When adding ``apiBase`` to the ``runtimeConfig.public``, Nuxt adds it to each page payload. We can universally access ``apiBase`` in both server and browser.

```
``ts
const runtimeConfig = useRuntimeConfig()

console.log(runtimeConfig.apiSecret)
console.log(runtimeConfig.public.apiBase)
``
```

```
::tip
Public runtime config is accessible in Vue templates with `$config.public`.
::
```

## ### Serialization

Your runtime config will be serialized before being passed to Nitro. This means that anything that cannot be serialized and then deserialized (such as functions, Sets, Maps, and so on), should not be set in your ``nuxt.config``.

Instead of passing non-serializable objects or functions into your application from your ``nuxt.config``, you can place this code in a Nuxt or Nitro plugin or middleware.

## ### Environment Variables

The most common way to provide configuration is by using [Environment Variables](https://medium.com/chingu/an-introduction-to-environment-variables-and-how-to-use-them-f602f66d15fa).

```
::note
Nuxi CLI has built-in support for reading your `.env` file in development, build and generate. But when you run your built server, your `.env` file will not be read.
:read-more{to="/docs/guide/directory-structure/env"}
::
```

Runtime config values are **automatically replaced by matching environment variables at runtime**.

There are two key requirements:

1. Your desired variables must be defined in your ``nuxt.config``. This ensures that arbitrary environment variables are not exposed to your application code.

1. Only a specially-named environment variable can override a runtime config property. That is, an uppercase environment variable starting with ``Nuxt_`` which uses ``_`` to separate keys and case changes.

```
::warning
Setting the default of `runtimeConfig` values to differently named environment variables (for example setting `myVar` to `process.env.OTHER_VARIABLE`) will only work during build-time and will break on runtime.
It is advised to use environment variables that match the structure of your `runtimeConfig` object.
::
```

```
::tip{icon="i-ph-video-duotone" to="https://youtu.be/_FYV5WfiWvs" target="_blank"}
Watch a video from Alexander Lichter showcasing the top mistake developers make using runtimeConfig.
::
```

## #### Example

```
``sh [.env]
Nuxt_API_SECRET=api_secret_token
Nuxt_PUBLIC_API_BASE=https://nuxtjs.org
``
```

```

````ts [nuxt.config.ts]
export default defineNuxtConfig({
  runtimeConfig: {
    apiSecret: '', // can be overridden by NUXT_API_SECRET environment variable
    public: {
      apiBase: '', // can be overridden by NUXT_PUBLIC_API_BASE environment variable
    }
  },
})
````

```

## ## Reading

### ### Vue App

Within the Vue part of your Nuxt app, you will need to call [`useRuntimeConfig()`](/docs/api/composables/use-runtime-config) to access the runtime config.

::important

The behavior is different between the client-side and server-side:

- On client-side, only keys in `runtimeConfig.public` are available, and the object is both writable and reactive.

- On server-side, the entire runtime config is available, but it is read-only to avoid context sharing.

::

```

````vue [pages/index.vue]
<script setup lang="ts">
const config = useRuntimeConfig()

console.log('Runtime config:', config)
if (import.meta.server) {
  console.log('API secret:', config.apiSecret)
}
</script>

```

```

<template>
  <div>
    <div>Check developer console!</div>
  </div>
</template>
````

```

::caution  
**\*\*Security note:\*\*** Be careful not to expose runtime config keys to the client-side by either rendering them or passing them to `useState`.  
 ::

### ### Plugins

If you want to use the runtime config within any (custom) plugin, you can use [`useRuntimeConfig()`](/docs/api/composables/use-runtime-config) inside of your `defineNuxtPlugin` function.

```

````ts [plugins/config.ts]
export default defineNuxtPlugin((nuxtApp) => {
  const config = useRuntimeConfig()

  console.log('API base URL:', config.public.apiBase)
});
````

```

### ### Server Routes

You can access runtime config within the server routes as well using `useRuntimeConfig`.

```

````ts [server/api/test.ts]
export default defineEventHandler(async (event) => {
  const { apiSecret } = useRuntimeConfig(event)
  const result = await $fetch('https://my.api.com/test', {
    headers: {
      Authorization: `Bearer ${apiSecret}`
    }
  })
  return result
})
````

```

::note  
 Giving the `event` as argument to `useRuntimeConfig` is optional, but it is recommended to pass it to get the runtime config overwritten by [environment variables](/docs/guide/going-further/runtime-config#environment-variables) at runtime for server routes.  
 ::

## ## Typing Runtime Config

Nuxt tries to automatically generate a typescript interface from provided runtime config using [unjs/untyped] (<https://github.com/unjs/untyped>).

But it is also possible to type your runtime config manually:

```
``ts [index.d.ts]
declare module 'nuxt/schema' {
 interface RuntimeConfig {
 apiSecret: string
 }
 interface PublicRuntimeConfig {
 apiBase: string
 }
}
// It is always important to ensure you import/export something when augmenting a type
export {}
``
```

::note

`nuxt/schema` is provided as a convenience for end-users to access the version of the schema used by Nuxt in their project. Module authors should instead augment `@nuxt/schema`.

::

```

title: "Nightly Release Channel"
description: "The nightly release channel allows using Nuxt built directly from the latest commits to the repository."

```

Nuxt lands commits, improvements, and bug fixes every day. You can opt in to test them earlier before the next release.

After a commit is merged into the `main` branch of [nuxt/nuxt](https://github.com/nuxt/nuxt) and **passes all tests**, we trigger an automated npm release, using GitHub Actions.

You can use these 'nightly' releases to beta test new features and changes.

The build and publishing method and quality of these 'nightly' releases are the same as stable ones. The only difference is that you should often check the GitHub repository for updates. There is a slight chance of regressions not being caught during the review process and by the automated tests. Therefore, we internally use this channel to double-check everything before each release.

```
::note
Features that are only available on the nightly release channel are marked with an alert in the documentation.
::
```

```
::alert{type="warning"}
The `latest` nightly release channel is currently tracking the Nuxt v4 branch, meaning that it is particularly likely to have breaking changes right now - be careful!
```

```
You can opt in to the 3.x branch nightly releases with `"nuxt": "npm:nuxt-nightly@3x"`.
::
```

## ## Opting In

Update `nuxt` dependency inside `package.json`:

```
```diff [package.json]
{
  "devDependencies": {
--    "nuxt": "^3.0.0"
++    "nuxt": "npm:nuxt-nightly@latest"
  }
}
```
```

Remove lockfile (`package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`, or `bun.lockb`) and reinstall dependencies.

## ## Opting Out

Update `nuxt` dependency inside `package.json`:

```
```diff [package.json]
{
  "devDependencies": {
--    "nuxt": "npm:nuxt-nightly@latest"
++    "nuxt": "^3.0.0"
  }
}
```
```

Remove lockfile (`package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`, or `bun.lockb`) and reinstall dependencies.

## ## Using Nightly `nuxi`

```
::note
All cli dependencies are bundled because of the building method for reducing `nuxi` package size. :br You can get dependency updates and CLI improvements using the nightly release channel.
::
```

To try the latest version of [nuxt/cli](https://github.com/nuxt/cli):

```
```bash [Terminal]
npx nuxi-nightly@latest [command]
```
```

```
::read-more{to="/docs/api/commands"}
Read more about the available commands.
::
```



```

title: useHeadSafe
description: The recommended way to provide head data with user input.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/unjs/unhead/blob/main/packages/unhead/src/composables/useHeadSafe.ts
 size: xs

```

The `useHeadSafe` composable is a wrapper around the `[`useHead`](/docs/api/composables/use-head)` composable that restricts the input to only allow safe values.

## ## Usage

You can pass all the same values as `[`useHead`](/docs/api/composables/use-head)`

```

```ts
useHeadSafe({
  script: [
    { id: 'xss-script', innerHTML: 'alert("xss")' }
  ],
  meta: [
    { 'http-equiv': 'refresh', content: '0;javascript:alert(1)' }
  ]
})
// Will safely generate
// <script id="xss-script"></script>
// <meta content="0;javascript:alert(1)">
```

::read-more{to="https://unhead.unjs.io/usage/composables/use-head-safe" target="_blank"}
Read more on `unhead` documentation.
::

```

## ## Type

```

```ts
useHeadSafe(input: MaybeComputedRef<HeadSafe>): void
```

```

The whitelist of safe values is:

```

```ts
export default {
  htmlAttrs: ['id', 'class', 'lang', 'dir'],
  bodyAttrs: ['id', 'class'],
  meta: ['id', 'name', 'property', 'charset', 'content'],
  noscript: ['id', 'textContent'],
  script: ['id', 'type', 'textContent'],
  link: ['id', 'color', 'crossorigin', 'fetchpriority', 'href', 'hreflang', 'imagesrcset', 'imagesizes', 'integrity',
'media', 'referrerpolicy', 'rel', 'sizes', 'type'],
}
```

```

See `[@unhead/schema](https://github.com/unjs/unhead/blob/main/packages/schema/src/safeSchema.ts)` for more detailed types.

```

title: "<NuxtPage>"
description: The <NuxtPage> component is required to display pages located in the pages/ directory.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/pages/runtime/page.ts
 size: xs

```

`<NuxtPage>` is a built-in component that comes with Nuxt. It lets you display top-level or nested pages located in the `[`pages/`](/docs/guide/directory-structure/pages)` directory.

```

::note
`<NuxtPage>` is a wrapper around [<RouterView>](https://router.vuejs.org/api/interfaces/RouterViewProps.html#interface-routerviewprops) component from Vue Router. :br
It accepts same `name` and `route` props.
::

```

```

::warning
`<NuxtPage>` should be used instead of `<RouterView>` as the former takes additional care on internal states. Otherwise,
`useRoute()` may return incorrect paths.
::

```

## ## Props

- `name`: tells `<RouterView>` to render the component with the corresponding name in the matched route record's components option.
  - type: ``string``
- `route`: route location that has all of its components resolved.
  - type: ``RouteLocationNormalized``
- `pageKey`: control when the `<NuxtPage>` component is re-rendered.
  - type: ``string`` or ``function``
- `transition`: define global transitions for all pages rendered with the `<NuxtPage>` component.
  - type: ``boolean`` or ``TransitionProps``
- `keepalive`: control state preservation of pages rendered with the `<NuxtPage>` component.
  - type: ``boolean`` or ``KeepAliveProps``

```

::tip
Nuxt automatically resolves the `name` and `route` by scanning and rendering all Vue component files found in the `/pages`
directory.
::

```

## ## Example

For example, passing `static` key, `<NuxtPage>` component is rendered only once when it is mounted.

```

```vue [app.vue]
<template>
  <NuxtPage page-key="static" />
</template>
```

```

You can also use a dynamic key based on the current route:

```

```html
<NuxtPage :page-key="route => route.fullPath" />
```

```

```

::warning
Don't use `$route` object here as it can cause problems with how `<NuxtPage>` renders pages with `<Suspense>`.
::

```

Alternatively, `pageKey` can be passed as a `key` value via [`definePageMeta`](/docs/api/utils/define-page-meta) from the `<script>` section of your Vue component in the `/pages` directory.

```

```vue [pages/my-page.vue]
<script setup lang="ts">
definePageMeta({
  key: route => route.fullPath
})
</script>
```

```

```

:link-example{to="/docs/examples/routing/pages"}

```

## ## Page's Ref

To get the `ref` of a page component, access it through `ref.value.pageRef`

```

```vue [app.vue]
<script setup lang="ts">
const page = ref()

```

```
function logFoo () {
  page.value.pageRef.foo()
}
</script>
```

```
<template>
  <NuxtPage ref="page" />
</template>
````
```

```
````vue [my-page.vue]
<script setup lang="ts">
const foo = () => {
  console.log('foo method called')
}
```

```
defineExpose({
  foo,
})
</script>
````
```

## ## Custom Props

In addition, `<NuxtPage>` also accepts custom props that you may need to pass further down the hierarchy.

These custom props are accessible via `attrs` in the Nuxt app.

```
``html
<NuxtPage :foobar="123" />
````
```

For example, in the above example, the value of `foobar` will be available using `$attrs.foobar` in the template or `useAttrs().foobar` in `<script setup>`.

```
:read-more{to="/docs/guide/directory-structure/pages"}
```

```

---
title: useHead
description: useHead customizes the head properties of individual pages of your Nuxt app.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/unjs/unhead/blob/main/packages/unhead/src/composables/useHead.ts
    size: xs
---

```

The `[`useHead`](/docs/api/composables/use-head)` composable function allows you to manage your head tags in a programmatic and reactive way, powered by `[Unhead](https://unhead.unjs.io)`. If the data comes from a user or other untrusted source, we recommend you check out `[`useHeadSafe`](/docs/api/composables/use-head-safe)`.

```
:read-more{to="/docs/getting-started/seo-meta"}
```

Type

```

````ts
useHead(meta: MaybeComputedRef<MetaObject>): void
````

```

Below are the non-reactive types for `[`useHead`](/docs/api/composables/use-head)` .

```

````ts
interface MetaObject {
 title?: string
 titleTemplate?: string | ((title?: string) => string)
 base?: Base
 link?: Link[]
 meta?: Meta[]
 style?: Style[]
 script?: Script[]
 noscript?: Noscript[]
 htmlAttrs?: HtmlAttributes
 bodyAttrs?: BodyAttributes
}
````

```

See `[@unhead/schema](https://github.com/unjs/unhead/blob/main/packages/schema/src/schema.ts)` for more detailed types.

```

::note
The properties of `useHead` can be dynamic, accepting `ref`, `computed` and `reactive` properties. `meta` parameter can also accept a function returning an object to make the entire object reactive.
::

```

Params

```

### `meta`

**Type**: `MetaObject`

```

An object accepting the following head metadata:

- ``meta``: Each element in the array is mapped to a newly-created `<meta>` tag, where object properties are mapped to the corresponding attributes.
 - `**Type**`: ``Array<Record<string, any>>``
- ``link``: Each element in the array is mapped to a newly-created `<link>` tag, where object properties are mapped to the corresponding attributes.
 - `**Type**`: ``Array<Record<string, any>>``
- ``style``: Each element in the array is mapped to a newly-created `<style>` tag, where object properties are mapped to the corresponding attributes.
 - `**Type**`: ``Array<Record<string, any>>``
- ``script``: Each element in the array is mapped to a newly-created `<script>` tag, where object properties are mapped to the corresponding attributes.
 - `**Type**`: ``Array<Record<string, any>>``
- ``noscript``: Each element in the array is mapped to a newly-created `<noscript>` tag, where object properties are mapped to the corresponding attributes.
 - `**Type**`: ``Array<Record<string, any>>``
- ``titleTemplate``: Configures dynamic template to customize the page title on an individual page.
 - `**Type**`: ``string` | `((title: string) => string)``
- ``title``: Sets static page title on an individual page.
 - `**Type**`: ``string``
- ``bodyAttrs``: Sets attributes of the `<body>` tag. Each object property is mapped to the corresponding attribute.
 - `**Type**`: ``Record<string, any>``
- ``htmlAttrs``: Sets attributes of the `<html>` tag. Each object property is mapped to the corresponding attribute.
 - `**Type**`: ``Record<string, any>``

```

---
title: "<NuxtLayout>"
description: "Nuxt provides the <NuxtLayout> component to show layouts on pages and error pages."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-layout.ts
    size: xs
---

```

You can use `<NuxtLayout />` component to activate the `default` layout on `app.vue` or `error.vue`.

```

```vue [app.vue]
<template>
 <NuxtLayout>
 some page content
 </NuxtLayout>
</template>
```

```

`:read-more{to="/docs/guide/directory-structure/layouts"}`

Props

- `name`: Specify a layout name to be rendered, can be a string, reactive reference or a computed property. It **must** match the name of the corresponding layout file in the `[layouts/](/docs/guide/directory-structure/layouts)` directory.
 - **type**: `string`
 - **default**: `default`

```

```vue [pages/index.vue]
<script setup lang="ts">
 // layouts/custom.vue
 const layout = 'custom'
</script>

<template>
 <NuxtLayout :name="layout">
 <NuxtPage />
 </NuxtLayout>
</template>
```

```

note
Please note the layout name is normalized to kebab-case, so if your layout file is named `errorLayout.vue`, it will become `error-layout` when passed as a `name` property to `<NuxtLayout />`.

```

```vue [error.vue]
<template>
 <NuxtLayout name="error-layout">
 <NuxtPage />
 </NuxtLayout>
</template>
```

```

`:read-more{to="/docs/guide/directory-structure/layouts"}`
Read more about dynamic layouts.

- `fallback`: If an invalid layout is passed to the `name` prop, no layout will be rendered. Specify a `fallback` layout to be rendered in this scenario. It **must** match the name of the corresponding layout file in the `[layouts/](/docs/guide/directory-structure/layouts)` directory.
 - **type**: `string`
 - **default**: `null`

Additional Props

`NuxtLayout` also accepts any additional props that you may need to pass to the layout. These custom props are then made accessible as attributes.

```

```vue [pages/some-page.vue]
<template>
 <div>
 <NuxtLayout name="custom" title="I am a custom layout">
 <-- ... -->
 </NuxtLayout>
 </div>
</template>
```

```

In the above example, the value of `title` will be available using `$attrs.title` in the template or `useAttrs().title` in `<script setup>` at `custom.vue`.

```

```vue [layouts/custom.vue]
<script setup lang="ts">
const layoutCustomProps = useAttrs()

console.log(layoutCustomProps.title) // I am a custom layout
</script>
```

```

Transitions

`<NuxtLayout />` renders incoming content via `not the root element of the page component.

::code-group

```

```vue [pages/index.vue]
<template>
 <div>
 <NuxtLayout name="custom">
 <template #header> Some header template content. </template>
 </NuxtLayout>
 </div>
</template>
```

```

```

```vue [layouts/custom.vue]
<template>
 <div>
 <!-- named slot -->
 <slot name="header" />
 </div>
</template>
```

```

::

:read-more{to="/docs/getting-started/transitions"}

Layout's Ref

To get the ref of a layout component, access it through `ref.value.layoutRef`.

::code-group

```

```vue [app.vue]
<script setup lang="ts">
const layout = ref()

function logFoo () {
 layout.value.layoutRef.foo()
}
</script>

```

```

<template>
 <NuxtLayout ref="layout">
 default layout
 </NuxtLayout>
</template>
```

```

```

```vue [layouts/default.vue]
<script setup lang="ts">
const foo = () => console.log('foo')
defineExpose({
 foo
})
</script>

```

```

<template>
 <div>
 default layout
 <slot />
 </div>
</template>
```

```

::

:read-more{to="/docs/guide/directory-structure/layouts"}

```
---
title: "Lifecycle Hooks"
description: "Nuxt provides a powerful hooking system to expand almost every aspect using hooks."
---
```

```
::tip
The hooking system is powered by [unjs/hookable](https://github.com/unjs/hookable).
::
```

Nuxt Hooks (Build Time)

These hooks are available for [Nuxt Modules](/docs/guide/going-further/modules) and build context.

Within `nuxt.config`

```
```js [nuxt.config]
export default defineNuxtConfig({
 hooks: {
 close: () => { }
 }
})
```
```

Within Nuxt Modules

```
```js
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule({
 setup (options, nuxt) {
 nuxt.hook('close', async () => { })
 }
})
```
```

```
::read-more{to="/docs/api/advanced/hooks#nuxt-hooks-build-time"}
Explore all available Nuxt hooks.
::
```

App Hooks (Runtime)

App hooks can be mainly used by [Nuxt Plugins](/docs/guide/directory-structure/plugins) to hook into rendering lifecycle but could also be used in Vue composables.

```
```js [plugins/test.ts]
export default defineNuxtPlugin((nuxtApp) => {
 nuxtApp.hook('page:start', () => {
 /* your code goes here */
 })
})
```
```

```
::read-more{to="/docs/api/advanced/hooks#app-hooks-runtime"}
Explore all available App hooks.
::
```

Server Hooks (Runtime)

These hooks are available for [server plugins](/docs/guide/directory-structure/server#server-plugins) to hook into Nitro's runtime behavior.

```
```js [~/server/plugins/test.ts]
export default defineNitroPlugin((nitroApp) => {
 nitroApp.hooks.hook('render:html', (html, { event }) => {
 console.log('render:html', html)
 html.bodyAppend.push('<hr>Appended by custom plugin')
 })

 nitroApp.hooks.hook('render:response', (response, { event }) => {
 console.log('render:response', response)
 })
})
```
```

```
::read-more{to="/docs/api/advanced/hooks#nitro-app-hooks-runtime-server-side"}
Learn more about available Nitro lifecycle hooks.
::
```

Additional Hooks

You can add additional hooks by augmenting the types provided by Nuxt. This can be useful for modules.

```
```ts
```

```
import { HookResult } from "@nuxt/schema";

declare module '#app' {
 interface RuntimeNuxtHooks {
 'your-nuxt-runtime-hook': () => HookResult
 }
 interface NuxtHooks {
 'your-nuxt-hook': () => HookResult
 }
}

declare module 'nitro/types' {
 interface NitroRuntimeHooks {
 'your-nitro-hook': () => void;
 }
}
...
```



```

title: "layouts"
head.title: "layouts/"
description: "Nuxt provides a layouts framework to extract common UI patterns into reusable layouts."
navigation.icon: i-ph-folder-duotone

::tip{icon="i-ph-rocket-launch-duotone" color="gray" }
For best performance, components placed in this directory will be automatically loaded via asynchronous import when used.
::

Enable Layouts

Layouts are enabled by adding [<NuxtLayout>](/docs/api/components/nuxt-layout) to your [app.vue](/docs/guide/directory-structure/app):

```vue [app.vue]
<template>
  <NuxtLayout>
    <NuxtPage />
  </NuxtLayout>
</template>
```

To use a layout:
- Set a layout property in your page with [definePageMeta](/docs/api/utils/define-page-meta).
- Set the name prop of <NuxtLayout>.

::note
The layout name is normalized to kebab-case, so someLayout becomes some-layout.
::

::note
If no layout is specified, layouts/default.vue will be used.
::

::important
If you only have a single layout in your application, we recommend using [app.vue](/docs/guide/directory-structure/app) instead.
::

::important
Unlike other components, your layouts must have a single root element to allow Nuxt to apply transitions between layout changes - and this root element cannot be a <slot />.
::

Default Layout

Add a ~/layouts/default.vue:

```vue [layouts/default.vue]
<template>
  <div>
    <p>Some default layout content shared across all pages</p>
    <slot />
  </div>
</template>
```

In a layout file, the content of the page will be displayed in the <slot /> component.

Named Layout


```

```bash [Directory Structure]
-| layouts/
--| default.vue
--| custom.vue
```

Then you can use the custom layout in your page:

```vue twoslash [pages/about.vue]
<script setup lang="ts">
definePageMeta({
 layout: 'custom'
})
</script>
```

::read-more{to="/docs/guide/directory-structure/pages#page-metadata"}
Learn more about definePageMeta.
::

```


```

You can directly override the default layout for all pages using the ``name`` property of [`<NuxtLayout>`]  
(/docs/api/components/nuxt-layout):

```
```vue [app.vue]
<script setup lang="ts">
// You might choose this based on an API call or logged-in status
const layout = "custom";
</script>

<template>
  <NuxtLayout :name="layout">
    <NuxtPage />
  </NuxtLayout>
</template>
```
```

If you have a layout in nested directories, the layout's name will be based on its own path directory and filename, with duplicate segments being removed.

| File                            | Layout Name     |
|---------------------------------|-----------------|
| ~/layouts/desktop/default.vue   | desktop-default |
| ~/layouts/desktop-base/base.vue | desktop-base    |
| ~/layouts/desktop/index.vue     | desktop         |

For clarity, we recommend that the layout's filename matches its name:

| File                                   | Layout Name     |
|----------------------------------------|-----------------|
| ~/layouts/desktop/DesktopDefault.vue   | desktop-default |
| ~/layouts/desktop-base/DesktopBase.vue | desktop-base    |
| ~/layouts/desktop/Desktop.vue          | desktop         |

[:link-example{to="/docs/examples/features/layouts"}](/docs/examples/features/layouts)

## ## Changing the Layout Dynamically

You can also use the [`setPageLayout`](/docs/api/utils/set-page-layout) helper to change the layout dynamically:

```
```vue twoslash
<script setup lang="ts">
function enableCustomLayout () {
  setPageLayout('custom')
}
definePageMeta({
  layout: false,
});
</script>

<template>
  <div>
    <button @click="enableCustomLayout">Update layout</button>
  </div>
</template>
```
```

[:link-example{to="/docs/examples/features/layouts"}](/docs/examples/features/layouts)

## ## Overriding a Layout on a Per-page Basis

If you are using pages, you can take full control by setting `layout: false` and then using the `<NuxtLayout>` component within the page.

::code-group

```
```vue [pages/index.vue]
<script setup lang="ts">
definePageMeta({
  layout: false,
})
</script>

<template>
  <div>
    <NuxtLayout name="custom">
      <template #header> Some header template content. </template>

      The rest of the page
    </NuxtLayout>
  </div>
</template>
```
```

```
``vue [layouts/custom.vue]
<template>
 <div>
 <header>
 <slot name="header">
 Default header content
 </slot>
 </header>
 <main>
 <slot />
 </main>
 </div>
</template>
``
```

::

::important

If you use `<NuxtLayout>` within your pages, make sure it is not the root element (or [disable layout/page transitions] (/docs/getting-started/transitions#disable-transitions)).

::

```

title: 'Code Style'
description: "Nuxt supports ESLint out of the box"

```

## ## ESLint

The recommended approach for Nuxt is to enable ESLint support using the [`@nuxt/eslint`] (<https://eslint.nuxt.com/packages/module>) module, that will setup project-aware ESLint configuration for you.

`:::callout{icon="i-ph-lightbulb-duotone"}`  
The module is designed for the [new ESLint flat config format](<https://eslint.org/docs/latest/use/configure/configuration-files-new>) with is the [default format since ESLint v9](<https://eslint.org/blog/2024/04/eslint-v9.0.0-released/>).

If you are using the legacy `.eslintrc` config, you will need to [configure manually with `@nuxt/eslint-config`] (<https://eslint.nuxt.com/packages/config#legacy-config-format>). We highly recommend you to migrate over the flat config to be future-proof.

`:::`

## ## Quick Setup

```
```bash
npx nuxi module add eslint
```
```

Start your Nuxt app, a `eslint.config.mjs` file will be generated under your project root. You can customize it as needed.

You can learn more about the module and customizations in [Nuxt ESLint's documentation] (<https://eslint.nuxt.com/packages/module>).

```

title: 'TypeScript'
description: "Nuxt is fully typed and provides helpful shortcuts to ensure you have access to accurate type information when you are coding."

```

## ## Type-checking

By default, Nuxt doesn't check types when you run [``nuxi dev``](/docs/api/commands/dev) or [``nuxi build``](/docs/api/commands/build), for performance reasons.

To enable type-checking at build or development time, install ``vue-tsc`` and ``typescript`` as development dependency:

::code-group

```
```bash [yarn]
yarn add --dev vue-tsc typescript
```

```bash [npm]
npm install --save-dev vue-tsc typescript
```

```bash [pnpm]
pnpm add -D vue-tsc typescript
```

```bash [bun]
bun add -D vue-tsc typescript
```
```

::

Then, run [``nuxi typecheck``](/docs/api/commands/typecheck) command to check your types:

```
```bash [Terminal]
npx nuxi typecheck
```
```

To enable type-checking at build time, you can also use the [``typescript.typeCheck``](/docs/api/nuxt-config#typecheck) option in your ``nuxt.config`` file:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  typescript: {
    typeCheck: true
  }
})
```
```

## ## Auto-generated Types

When you run ``nuxi dev`` or ``nuxi build``, Nuxt generates the following files for IDE type support (and type checking):

### ``nuxt/nuxt.d.ts``

This file contains the types of any modules you are using, as well as the key types that Nuxt requires. Your IDE should recognize these types automatically.

Some of the references in the file are to files that are only generated within your ``buildDir`` (``nuxt``) and therefore for full typings, you will need to run ``nuxi dev`` or ``nuxi build``.

### ``nuxt/tsconfig.json``

This file contains the recommended basic TypeScript configuration for your project, including resolved aliases injected by Nuxt or modules you are using, so you can get full type support and path auto-complete for aliases like ``~/file`` or ``#build/file``.

[Read more about how to extend this configuration](/docs/guide/directory-structure/tsconfig).

::tip{icon="i-ph-video-duotone" to="https://youtu.be/umLI7SlPygY" target="\_blank"}  
Watch a video from Daniel Roe explaining built-in Nuxt aliases.  
::

::note  
Nitro also [auto-generates types](/docs/guide/concepts/server-engine#typed-api-routes) for API routes. Plus, Nuxt also generates types for globally available components and [auto-imports from your composables](/docs/guide/directory-structure/composables), plus other core functionality.  
::

::note  
Keep in mind that all options extended from ``./.nuxt/tsconfig.json`` will be overwritten by the options defined in your ``tsconfig.json``.

Overwriting options such as `"compilerOptions.paths"` with your own configuration will lead TypeScript to not factor in the module resolutions from `./.nuxt/tsconfig.json`. This can lead to module resolutions such as `#imports` not being recognized.  
:br :br  
In case you need to extend options provided by `./.nuxt/tsconfig.json` further, you can use the `[`alias` property]` (`/docs/api/nuxt-config#alias`) within your `nuxt.config`. `nuxi` will pick them up and extend `./.nuxt/tsconfig.json` accordingly.  
::

## ## Strict Checks

TypeScript comes with certain checks to give you more safety and analysis of your program.

[Strict checks](<https://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html#getting-stricter-checks>) are enabled by default in Nuxt to give you greater type safety.

If you are currently converting your codebase to TypeScript, you may want to temporarily disable strict checks by setting `strict` to `false` in your `nuxt.config`:

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  typescript: {
    strict: false
  }
})
```:
```

```

title: 'clearNuxtState'
description: Delete the cached state of useState.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/state.ts
 size: xs

```

```
:::note
This method is useful if you want to invalidate the state of `useState`.
:::
```

## ## Type

```
```:ts
clearNuxtState (keys?: string | string[] | ((key: string) => boolean)): void
```:
```

## ## Parameters

- `keys`: One or an array of keys that are used in [`useState`](/docs/api/composables/use-state) to delete their cached state. If no keys are provided, **all state** will be invalidated.

```

title: 'Views'
description: 'Nuxt provides several component layers to implement the user interface of your application.'
navigation.icon: i-ph-layout-duotone

```

### ## `app.vue`

![The app.vue file is the entry point of your application](/assets/docs/getting-started/views/app.svg)

By default, Nuxt will treat this file as the **entrypoint** and render its content for every route of the application.

```
```vue [app.vue]
<template>
  <div>
    <h1>Welcome to the homepage</h1>
  </div>
</template>
```
```

#### ::tip

If you are familiar with Vue, you might wonder where `main.js` is (the file that normally creates a Vue app). Nuxt does this behind the scene.

::

### ## Components

![Components are reusable pieces of UI](/assets/docs/getting-started/views/components.svg)

Most components are reusable pieces of the user interface, like buttons and menus. In Nuxt, you can create these components in the [`components/`](/docs/guide/directory-structure/components) directory, and they will be automatically available across your application without having to explicitly import them.

::code-group

```
```vue [app.vue]
<template>
  <div>
    <h1>Welcome to the homepage</h1>
    <AppAlert>
      This is an auto-imported component.
    </AppAlert>
  </div>
</template>
```
```

```
```vue [components/AppAlert.vue]
<template>
  <span>
    <slot />
  </span>
</template>
```
```

::

### ## Pages

![Pages are views tied to a specific route](/assets/docs/getting-started/views/pages.svg)

Pages represent views for each specific route pattern. Every file in the [`pages/`](/docs/guide/directory-structure/pages) directory represents a different route displaying its content.

To use pages, create `pages/index.vue` file and add `

::code-group

```
```vue [pages/index.vue]
<template>
  <div>
    <h1>Welcome to the homepage</h1>
    <AppAlert>
      This is an auto-imported component
    </AppAlert>
  </div>
</template>
```
```

```
```vue [pages/about.vue]
<template>
  <section>
```



```
    <p>This page will be displayed at the /about route.</p>
  </section>
</template>
````
```

::

```
:read-more{title="Routing Section" to="/docs/getting-started/routing"}
```

## ## Layouts

![Layouts are wrapper around pages](/assets/docs/getting-started/views/layouts.svg)

Layouts are wrappers around pages that contain a common User Interface for several pages, such as a header and footer display. Layouts are Vue files using `page content. The `layouts/default.vue` file will be used by default. Custom layouts can be set as part of your page metadata.

```
::note
If you only have a single layout in your application, we recommend using [`app.vue`](/docs/guide/directory-structure/app)
with [`${NuxtPage />}`](/docs/api/components/nuxt-page) instead.
::
```

```
::code-group
```

```
```vue [app.vue]
<template>
  <div>
    <NuxtLayout>
      <NuxtPage />
    </NuxtLayout>
  </div>
</template>
```
```

```
```vue [layouts/default.vue]
<template>
  <div>
    <AppHeader />
    <slot />
    <AppFooter />
  </div>
</template>
```
```

```
```vue [pages/index.vue]
<template>
  <div>
    <h1>Welcome to the homepage</h1>
    <AppAlert>
      This is an auto-imported component
    </AppAlert>
  </div>
</template>
```
```

```
```vue [pages/about.vue]
<template>
  <section>
    <p>This page will be displayed at the /about route.</p>
  </section>
</template>
```
```

::

If you want to create more layouts and learn how to use them in your pages, find more information in the [Layouts section](/docs/guide/directory-structure/layouts).

## ## Advanced: Extending the HTML template

```
::note
If you only need to modify the `<head>`, you can refer to the [SEO and meta section](/docs/getting-started/seo-meta).
::
```

You can have full control over the HTML template by adding a Nitro plugin that registers a hook. The callback function of the `render:html` hook allows you to mutate the HTML before it is sent to the client.

```
```ts twoslash [server/plugins/extend-html.ts]
export default defineNitroPlugin((nitroApp) => {
  nitroApp.hooks.hook('render:html', (html, { event }) => {
    // This will be an object representation of the html template.
    console.log(html)
    html.head.push(`<meta name="description" content="My custom description" />`)
  })
})
```

```
  })  
  // You can also intercept the response here.  
  nitroApp.hooks.hook('render:response', (response, { event }) => { console.log(response) })  
  })  
  ``
```

```
:read-more{to="/docs/guide/going-further/hooks"}
```

```
---
title: "defineNuxtComponent"
description: defineNuxtComponent() is a helper function for defining type safe components with Options API.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/component.ts
    size: xs
---
```

```
:::note
`defineNuxtComponent()` is a helper function for defining type safe Vue components using options API similar to
[`defineComponent()`](https://vuejs.org/api/general.html#definecomponent). `defineNuxtComponent()` wrapper also adds support
for `asyncData` and `head` component options.
:::
```

```
:::note
Using `
```

```

---
title: 'createError'
description: Create an error object with additional metadata.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/error.ts
    size: xs
---

```

You can use this function to create an error object with additional metadata. It is usable in both the Vue and Nitro portions of your app, and is meant to be thrown.

Parameters

```
- `err`: `string | { cause, data, message, name, stack, statusCode, statusMessage, fatal }`
```

You can pass either a string or an object to the `createError` function. If you pass a string, it will be used as the error `message`, and the `statusCode` will default to `500`. If you pass an object, you can set multiple properties of the error, such as `statusCode`, `message`, and other error properties.

In Vue App

If you throw an error created with `createError`:

- on server-side, it will trigger a full-screen error page which you can clear with `clearError`.
- on client-side, it will throw a non-fatal error for you to handle. If you need to trigger a full-screen error page, then you can do this by setting `fatal: true`.

Example

```

```vue [pages/movies/[slug\].vue]
<script setup lang="ts">
const route = useRoute()
const { data } = await useFetch(`/api/movies/${route.params.slug}`)
if (!data.value) {
 throw createError({ statusCode: 404, statusMessage: 'Page Not Found' })
}
</script>
```

```

In API Routes

Use `createError` to trigger error handling in server API routes.

Example

```

```ts [server/api/error.ts]
export default eventHandler(() => {
 throw createError({
 statusCode: 404,
 statusMessage: 'Page Not Found'
 })
})
```

```

In API routes, using `createError` by passing an object with a short `statusMessage` is recommended because it can be accessed on the client side. Otherwise, a `message` passed to `createError` on an API route will not propagate to the client. Alternatively, you can use the `data` property to pass data back to the client. In any case, always consider avoiding to put dynamic user input to the message to avoid potential security issues.

:read-more{to="/docs/getting-started/error-handling"}

```

---
title: 'modules'
head.title: 'modules/'
description: Use the modules/ directory to automatically register local modules within your application.
navigation.icon: i-ph-folder-duotone
---

```

It is a good place to place any local modules you develop while building your application.

The auto-registered files patterns are:

```

- `modules/*/index.ts`
- `modules/*.ts`

```

You don't need to add those local modules to your `[`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config)` separately.

```

::code-group

```

```

```ts twoslash [modules/hello/index.ts]
// `nuxt/kit` is a helper subpath import you can use when defining local modules
// that means you do not need to add `@nuxt/kit` to your project's dependencies
import { createResolver, defineNuxtModule, addServerHandler } from 'nuxt/kit'

```

```

export default defineNuxtModule({
 meta: {
 name: 'hello'
 },
 setup () {
 const { resolve } = createResolver(import.meta.url)

 // Add an API route
 addServerHandler({
 route: '/api/hello',
 handler: resolve('./runtime/api-route')
 })
 }
})
```

```

```

```ts twoslash [modules/hello/runtime/api-route.ts]
export default defineEventHandler(() => {
 return { hello: 'world' }
})
```

```

```

::

```

When starting Nuxt, the ``hello`` module will be registered and the ``/api/hello`` route will be available.

Modules are executed in the following sequence:

- First, the modules defined in `[`nuxt.config.ts`](/docs/api/nuxt-config#modules-1)` are loaded.
- Then, modules found in the ``modules/`` directory are executed, and they load in alphabetical order.

You can change the order of local module by adding a number to the front of each directory name:

```

```bash [Directory structure]
modules/
 1.first-module/
 index.ts
 2.second-module.ts
```

```

```

:read-more{to="/docs/guide/going-further/modules"}

```

```

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/creating-your-first-module-from-scratch?friend=nuxt"
target="_blank"}

```

Watch Vue School video about Nuxt private modules.

```

::

```

```

---
title: "middleware"
description: "Nuxt provides middleware to run code before navigating to a particular route."
head.title: "middleware/"
navigation.icon: i-ph-folder-duotone
---

```

Nuxt provides a customizable **route middleware** framework you can use throughout your application, ideal for extracting code that you want to run before navigating to a particular route.

There are three kinds of route middleware:

1. Anonymous (or inline) route middleware are defined directly within the page.
2. Named route middleware, placed in the ``middleware/`` and automatically loaded via asynchronous import when used on a page.
3. Global route middleware, placed in the ``middleware/`` with a ``.global`` suffix and is run on every route change.

The first two kinds of route middleware can be defined in `[`definePageMeta`](/docs/api/utils/define-page-meta)`.

```

::note
Name of middleware are normalized to kebab-case: `myMiddleware` becomes `my-middleware`.
::

```

```

::note
Route middleware run within the Vue part of your Nuxt app. Despite the similar name, they are completely different from
[server middleware](/docs/guide/directory-structure/server#server-middleware), which are run in the Nitro server part of your
app.
::

```

Usage

Route middleware are navigation guards that receive the current route and the next route as arguments.

```

`ts twoslash [middleware/my-middleware.ts]
export default defineNuxtRouteMiddleware((to, from) => {
  if (to.params.id === '1') {
    return abortNavigation()
  }
  // In a real app you would probably not redirect every route to `/`
  // however it is important to check `to.path` before redirecting or you
  // might get an infinite redirect loop
  if (to.path !== '/') {
    return navigateTo('/')
  }
})
`

```

Nuxt provides two globally available helpers that can be returned directly from the middleware.

1. `[`navigateTo`](/docs/api/utils/navigate-to)` - Redirects to the given route
2. `[`abortNavigation`](/docs/api/utils/abort-navigation)` - Aborts the navigation, with an optional error message.

Unlike [navigation guards](https://router.vuejs.org/guide/advanced/navigation-guards.html#global-before-guards) from ``vue-router``, a third `next()` argument is not passed, and **redirect or route cancellation** is handled by returning a value from the middleware.

Possible return values are:

- * nothing (a simple ``return`` or no return at all) - does not block navigation and will move to the next middleware function, if any, or complete the route navigation
- * ``return navigateTo('/')`` - redirects to the given path and will set the redirect code to `[`302` Found]` (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302) if the redirect happens on the server side
- * ``return navigateTo('/', { redirectCode: 301 })`` - redirects to the given path and will set the redirect code to `[`301` Moved Permanently]` (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301) if the redirect happens on the server side
- * ``return abortNavigation()`` - stops the current navigation
- * ``return abortNavigation(error)`` - rejects the current navigation with an error

```

:read-more{to="/docs/api/utils/navigate-to"}
:read-more{to="/docs/api/utils/abort-navigation"}

```

```

::important

```

We recommend using the helper functions above for performing redirects or stopping navigation. Other possible return values described in [the vue-router docs](https://router.vuejs.org/guide/advanced/navigation-guards.html#global-before-guards) may work but there may be breaking changes in future.

```

::

```

Middleware Order

Middleware runs in the following order:

1. Global Middleware
2. Page defined middleware order (if there are multiple middleware declared with the array syntax)

For example, assuming you have the following middleware and component:

```

```text [middleware/ directory]
middleware/
--| analytics.global.ts
--| setup.global.ts
--| auth.ts
```

```vue twoslash [pages/profile.vue]
<script setup lang="ts">
definePageMeta({
 middleware: [
 function (to, from) {
 // Custom inline middleware
 },
 'auth',
],
});
</script>
```

```

You can expect the middleware to be run in the following order:

1. `analytics.global.ts`
2. `setup.global.ts`
3. Custom inline middleware
4. `auth.ts`

Ordering Global Middleware

By default, global middleware is executed alphabetically based on the filename.

However, there may be times you want to define a specific order. For example, in the last scenario, `setup.global.ts` may need to run before `analytics.global.ts`. In that case, we recommend prefixing global middleware with 'alphabetical' numbering.

```

```text [Directory structure]
middleware/
--| 01.setup.global.ts
--| 02.analytics.global.ts
--| auth.ts
```

```

:::note

In case you're new to 'alphabetical' numbering, remember that filenames are sorted as strings, not as numeric values. For example, `10.new.global.ts` would come before `2.new.global.ts`. This is why the example prefixes single digit numbers with `0`.

:::

When Middleware Runs

If your site is server-rendered or generated, middleware for the initial page will be executed both when the page is rendered and then again on the client. This might be needed if your middleware needs a browser environment, such as if you have a generated site, aggressively cache responses, or want to read a value from local storage.

However, if you want to avoid this behaviour you can do so:

```

```ts twoslash [middleware/example.ts]
export default defineNuxtRouteMiddleware(to => {
 // skip middleware on server
 if (import.meta.server) return
 // skip middleware on client side entirely
 if (import.meta.client) return
 // or only skip middleware on initial client load
 const nuxtApp = useNuxtApp()
 if (import.meta.client && nuxtApp.isHydrating && nuxtApp.payload.serverRendered) return
})
```

```

:::note

Rendering an error page is an entirely separate page load, meaning any registered middleware will run again. You can use [`useError`](/docs/getting-started/error-handling#useerror) in middleware to check if an error is being handled.

:::

Adding Middleware Dynamically

It is possible to add global or named route middleware manually using the [`addRouteMiddleware()`](/docs/api/utils/add-route-middleware) helper function, such as from within a plugin.

```

```ts twoslash
export default defineNuxtPlugin(() => {
 addRouteMiddleware('global-test', () => {
 console.log('this global middleware was added in a plugin and will be run on every route change')
 })
})
```

```

```

    }, { global: true })

    addRouteMiddleware('named-test', () => {
      console.log('this named middleware was added in a plugin and would override any existing middleware of the same name')
    })
  })
  ...

```

Example

```

```bash [Directory Structure]
-| middleware/
---| auth.ts
...

```

In your page file, you can reference this route middleware:

```

```vue twoslash
<script setup lang="ts">
definePageMeta({
  middleware: ["auth"]
  // or middleware: 'auth'
})
</script>
```

```

Now, before navigation to that page can complete, the `auth` route middleware will be run.

```
:link-example{to="/docs/examples/routing/middleware"}
```

## ## Setting Middleware At Build Time

Instead of using `definePageMeta` on each page, you can add named route middleware within the `pages:extend` hook.

```

```ts twoslash [nuxt.config.ts]
import type { NuxtPage } from 'nuxt/schema'

export default defineNuxtConfig({
  hooks: {
    'pages:extend' (pages) {
      function setMiddleware (pages: NuxtPage[]) {
        for (const page of pages) {
          if (/* some condition */ true) {
            page.meta ||= {}
            // Note that this will override any middleware set in `definePageMeta` in the page
            page.meta.middleware = ['named']
          }
          if (page.children) {
            setMiddleware(page.children)
          }
        }
      }
      setMiddleware(pages)
    }
  }
})
```

```



```

title: "<NuxtLink>"
description: "Nuxt provides <NuxtLink> component to handle any kind of links within your application."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-link.ts
 size: xs

```

```
::note
`<NuxtLink>` is a drop-in replacement for both Vue Router's `<RouterLink>` component and HTML's `<a>` tag. It intelligently determines whether the link is _internal_ or _external_ and renders it accordingly with available optimizations (prefetching, default attributes, etc.)
::
```

## ## Internal Routing

In this example, we use `<NuxtLink>` component to link to another page of the application.

```
```vue [pages/index.vue]
<template>
  <NuxtLink to="/about">
    About page
  </NuxtLink>
  <!-- <a href="/about">...</a> (+Vue Router & prefetching) -->
</template>
```
```

## ### Passing Params to Dynamic Routes

In this example, we pass the `id` param to link to the route `~/pages/posts/[id].vue`.

```
```vue [pages/index.vue]
<template>
  <NuxtLink :to="{ name: 'posts-id', params: { id: 123 } }">
    Post 123
  </NuxtLink>
</template>
```
```

```
::tip
Check out the Pages panel in Nuxt DevTools to see the route name and the params it might take.
::
```

## ### Handling 404s

When using `<NuxtLink>` for `/public` directory files or when pointing to a different app on the same domain, you should use the `external` prop.

Using `external` forces the link to be rendered as an `a` tag instead of a Vue Router `RouterLink`.

```
```vue [pages/index.vue]
<template>
  <NuxtLink to="/the-important-report.pdf" external>
    Download Report
  </NuxtLink>
  <!-- <a href="/the-important-report.pdf"></a> -->
</template>
```
```

The external logic is applied by default when using absolute URLs and when providing a `target` prop.

## ## External Routing

In this example, we use `<NuxtLink>` component to link to a website.

```
```vue [app.vue]
<template>
  <NuxtLink to="https://nuxtjs.org">
    Nuxt website
  </NuxtLink>
  <!-- <a href="https://nuxtjs.org" rel="noopener noreferrer">...</a> -->
</template>
```
```

## ## `target` and `rel` Attributes

A `rel` attribute of `noopener noreferrer` is applied by default to absolute links and links that open in new tabs.

- `noopener` solves a [security bug](https://mathiasbynens.github.io/rel-noopener/) in older browsers.
- `noreferrer` improves privacy for your users by not sending the `Referer` header to the linked site.

These defaults have no negative impact on SEO and are considered [best practice]

(<https://developer.chrome.com/docs/lighthouse/best-practices/external-anchors-use-rel-noopener>).

When you need to overwrite this behavior you can use the `rel` and `noRel` props.

```
``vue [app.vue]
<template>
 <NuxtLink to="https://twitter.com/nuxt_js" target="_blank">
 Nuxt Twitter
 </NuxtLink>
 <!-- ... -->

 <NuxtLink to="https://discord.nuxtjs.org" target="_blank" rel="noopener">
 Nuxt Discord
 </NuxtLink>
 <!-- ... -->

 <NuxtLink to="https://github.com/nuxt" no-rel>
 Nuxt GitHub
 </NuxtLink>
 <!-- ... -->

 <NuxtLink to="/contact" target="_blank">
 Contact page opens in another tab
 </NuxtLink>
 <!-- ... -->
</template>
``
```

## ## Props

### ### RouterLink

When not using `external`, `` supports all Vue Router's [`RouterLink` props](<https://router.vuejs.org/api/interfaces/RouterLinkProps.html>)

- `to`: Any URL or a [route location object](<https://router.vuejs.org/api/#RouteLocation>) from Vue Router
- `custom`: Whether `` should wrap its content in an `` element. It allows taking full control of how a link is rendered and how navigation works when it is clicked. Works the same as [Vue Router's `custom` prop](<https://router.vuejs.org/api/interfaces/RouterLinkProps.html#Properties-custom>)
- `exactActiveClass`: A class to apply on exact active links. Works the same as [Vue Router's `exact-active-class` prop](<https://router.vuejs.org/api/interfaces/RouterLinkProps.html#Properties-exactActiveClass>) on internal links. Defaults to Vue Router's default `"router-link-exact-active"`)
- `replace`: Works the same as [Vue Router's `replace` prop](<https://router.vuejs.org/api/interfaces/RouteLocationOptions.html#Properties-replace>) on internal links
- `ariaCurrentValue`: An `aria-current` attribute value to apply on exact active links. Works the same as [Vue Router's `aria-current-value` prop](<https://router.vuejs.org/api/interfaces/RouterLinkProps.html#Properties-ariaCurrentValue>) on internal links
- `activeClass`: A class to apply on active links. Works the same as [Vue Router's `active-class` prop](<https://router.vuejs.org/api/interfaces/RouterLinkProps.html#Properties-activeClass>) on internal links. Defaults to Vue Router's default (`"router-link-active"`)

### ### NuxtLink

- `href`: An alias for `to`. If used with `to`, `href` will be ignored
- `noRel`: If set to `true`, no `rel` attribute will be added to the link
- `external`: Forces the link to be rendered as an `a` tag instead of a Vue Router `RouterLink`.
- `prefetch`: When enabled will prefetch middleware, layouts and payloads (when using [payloadExtraction](<https://nuxt.com/docs/api/nuxt-config#crossoriginprefetch>)) of links in the viewport. Used by the experimental [crossOriginPrefetch](<https://nuxt.com/docs/api/nuxt-config#crossoriginprefetch>) config.
- `noPrefetch`: Disables prefetching.
- `prefetchedClass`: A class to apply to links that have been prefetched.

### ### Anchor

- `target`: A `target` attribute value to apply on the link
- `rel`: A `rel` attribute value to apply on the link. Defaults to `"noopener noreferrer"` for external links.

::tip

Defaults can be overwritten, see [overwriting defaults]([#overwriting-defaults](#)) if you want to change them.

::

## ## Overwriting Defaults

### ### In Nuxt Config

You can overwrite some `` defaults in your [nuxt.config](<https://nuxt.com/docs/api/nuxt-config#defaults>)

::important

These options will likely be moved elsewhere in the future, such as into `app.config` or into the `app/` directory.

::

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
```

```
experimental: {
 defaults: {
 nuxtLink: {
 // default values
 componentName: 'NuxtLink',
 externalRelAttribute: 'noopener noreferrer',
 activeClass: 'router-link-active',
 exactActiveClass: 'router-link-exact-active',
 prefetchedClass: undefined, // can be any valid string class name
 trailingSlash: undefined // can be 'append' or 'remove'
 }
 }
}
})
...
```

### ### Custom Link Component

You can overwrite ``<NuxtLink>`` defaults by creating your own link component using ``defineNuxtLink``.

```
```js [components/MyNuxtLink.ts]
export default defineNuxtLink({
  componentName: 'MyNuxtLink',
  /* see signature below for more */
})
...
```
```

You can then use ``<MyNuxtLink />`` component as usual with your new defaults.

### ### ``defineNuxtLink`` Signature

```
```ts
interface NuxtLinkOptions {
  componentName?: string;
  externalRelAttribute?: string;
  activeClass?: string;
  exactActiveClass?: string;
  prefetchedClass?: string;
  trailingSlash?: 'append' | 'remove'
}
function defineNuxtLink(options: NuxtLinkOptions): Component {}
...
```
```

- ``componentName``: A name for the component. Default is ``NuxtLink``.
- ``externalRelAttribute``: A default ``rel`` attribute value applied on external links. Defaults to ``"noopener noreferrer"`. Set it to ``""` to disable
- ``activeClass``: A default class to apply on active links. Works the same as [Vue Router's ``linkActiveClass`` option] (<https://router.vuejs.org/api/interfaces/RouterOptions.html#Properties-linkActiveClass>). Defaults to Vue Router's default (`"router-link-active"`)
- ``exactActiveClass``: A default class to apply on exact active links. Works the same as [Vue Router's ``linkExactActiveClass`` option] (<https://router.vuejs.org/api/interfaces/RouterOptions.html#Properties-linkExactActiveClass>). Defaults to Vue Router's default (`"router-link-exact-active"`)
- ``prefetchedClass``: A default class to apply to links that have been prefetched.
- ``trailingSlash``: An option to either add or remove trailing slashes in the ``href``. If unset or not matching the valid values ``append`` or ``remove``, it will be ignored.

```
:link-example{to="/docs/examples/routing/pages"}
```

```

title: 'useHydration'
description: 'Allows full control of the hydration cycle to set and receive data from the server.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/hydrate.ts
 size: xs

```

``useHydration`` is a built-in composable that provides a way to set data on the server side every time a new HTTP request is made and receive that data on the client side. This way ``useHydration`` allows you to take full control of the hydration cycle.

```
:::note
This is an advanced composable and is mostly used internally (`useAsyncData`) or by Nuxt modules.
:::
```

## ## Type

```
```:ts [signature]
useHydration <T> (key: string, get: () => T, set: (value: T) => void) => {}
```:
```

You can use ``useHydration()`` within composables, plugins and components.

``useHydration`` accepts three parameters:

- ``key``: unique key that identifies the data in your Nuxt application
  - **Type**: ``String``
- ``get``: function that returns the value to set the initial data
  - **Type**: ``Function``
- ``set``: function that receives the data on the client-side
  - **Type**: ``Function``

Once the initial data is returned using the ``get`` function on the server side, you can access that data within ``nuxtApp.payload`` using the unique key that is passed as the first parameter in ``useHydration`` composable.

```
:read-more{to="/docs/getting-started/data-fetching"}
```

```

title: 'Assets'
description: 'Nuxt offers two options for your assets.'
navigation.icon: i-ph-image-duotone

```

Nuxt uses two directories to handle assets like stylesheets, fonts or images.

- The [`public/`](/docs/guide/directory-structure/public) directory content is served at the server root as-is.
- The [`assets/`](/docs/guide/directory-structure/assets) directory contains by convention every asset that you want the build tool (Vite or webpack) to process.

## ## Public Directory

The [`public/`](/docs/guide/directory-structure/public) directory is used as a public server for static assets publicly available at a defined URL of your application.

You can get a file in the [`public/`](/docs/guide/directory-structure/public) directory from your application's code or from a browser by the root URL `/`.

### ### Example

For example, referencing an image file in the `public/img/` directory, available at the static URL `/img/nuxt.png`:

```
```vue [app.vue]
<template>
  
</template>
```
```

## ## Assets Directory

Nuxt uses [Vite](https://vitejs.dev/guide/assets.html) (default) or [webpack](https://webpack.js.org/guides/asset-management) to build and bundle your application. The main function of these build tools is to process JavaScript files, but they can be extended through [plugins](https://vitejs.dev/plugins) (for Vite) or [loaders](https://webpack.js.org/loaders) (for webpack) to process other kind of assets, like stylesheets, fonts or SVG. This step transforms the original file mainly for performance or caching purposes (such as stylesheets minification or browser cache invalidation).

By convention, Nuxt uses the [`assets/`](/docs/guide/directory-structure/assets) directory to store these files but there is no auto-scan functionality for this directory, and you can use any other name for it.

In your application's code, you can reference a file located in the [`assets/`](/docs/guide/directory-structure/assets) directory by using the `~/assets/` path.

### ### Example

For example, referencing an image file that will be processed if a build tool is configured to handle this file extension:

```
```vue [app.vue]
<template>
  
</template>
```
```

::note

Nuxt won't serve files in the [`assets/`](/docs/guide/directory-structure/assets) directory at a static URL like `/assets/my-file.png`. If you need a static URL, use the [`public/`](#public-directory) directory.

::

### ### Global Styles Imports

To globally insert statements in your Nuxt components styles, you can use the [`Vite`](/docs/api/nuxt-config#vite) option at your [`nuxt.config`](/docs/api/nuxt-config) file.

#### #### Example

In this example, there is a [sass partial](https://sass-lang.com/documentation/at-rules/use#partials) file containing color variables to be used by your Nuxt [pages](/docs/guide/directory-structure/pages) and [components](/docs/guide/directory-structure/components)

::code-group

```
```scss [assets/_colors.scss]
$primary: #49240F;
$secondary: #E4A79D;
```
```

```
```sass [assets/_colors.sass]
$primary: #49240F
$secondary: #E4A79D
```
```

::

In your `nuxt.config`

::code-group

```
``ts twoslash [SCSS]
export default defineNuxtConfig({
 vite: {
 css: {
 preprocessorOptions: {
 scss: {
 additionalData: '@use "~/assets/_colors.scss" as *;'
 }
 }
 }
 }
})
``
```

```
``ts twoslash [SASS]
export default defineNuxtConfig({
 vite: {
 css: {
 preprocessorOptions: {
 sass: {
 additionalData: '@use "~/assets/_colors.sass" as *\n'
 }
 }
 }
 }
})
``
```

::

```

title: 'Styling'
description: 'Learn how to style your Nuxt application.'
navigation.icon: i-ph-palette-duotone

```

Nuxt is highly flexible when it comes to styling. Write your own styles, or reference local and external stylesheets. You can use CSS preprocessors, CSS frameworks, UI libraries and Nuxt modules to style your application.

## ## Local Stylesheets

If you're writing local stylesheets, the natural place to put them is the [`assets/`` directory](/docs/guide/directory-structure/assets).

### ### Importing Within Components

You can import stylesheets in your pages, layouts and components directly. You can use a javascript import, or a css [`@import`` statement](https://developer.mozilla.org/en-US/docs/Web/CSS/@import).

```
``vue [pages/index.vue]
<script>
// Use a static import for server-side compatibility
import '~/assets/css/first.css'

// Caution: Dynamic imports are not server-side compatible
import('~/assets/css/first.css')
</script>
```

```
<style>
@import url("~/assets/css/second.css");
</style>
``
```

```
::tip
The stylesheets will be inlined in the HTML rendered by Nuxt.
::
```

### ### The CSS Property

You can also use the `css`` property in the Nuxt configuration. The natural place for your stylesheets is the [`assets/`` directory](/docs/guide/directory-structure/assets). You can then reference its path and Nuxt will include it to all the pages of your application.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 css: ['~/assets/css/main.css']
})
``
```

```
::tip
The stylesheets will be inlined in the HTML rendered by Nuxt, injected globally and present in all pages.
::
```

### ### Working With Fonts

Place your local fonts files in your `~/public/`` directory, for example in `~/public/fonts``. You can then reference them in your stylesheets using `url()``.

```
``css [assets/css/main.css]
@font-face {
 font-family: 'FarAwayGalaxy';
 src: url('/fonts/FarAwayGalaxy.woff') format('woff');
 font-weight: normal;
 font-style: normal;
 font-display: swap;
}
``
```

Then reference your fonts by name in your stylesheets, pages or components:

```
``vue
<style>
h1 {
 font-family: 'FarAwayGalaxy', sans-serif;
}
</style>
``
```

### ### Stylesheets Distributed Through NPM

You can also reference stylesheets that are distributed through npm. Let's use the popular `animate.css`` library as an example.

```
::code-group
```

```
```bash [npm]
npm install animate.css
```
```

```
```bash [yarn]
yarn add animate.css
```
```

```
```bash [pnpm]
pnpm install animate.css
```
```

```
```bash [bun]
bun install animate.css
```
```

```
::
```

Then you can reference it directly in your pages, layouts and components:

```
```vue [app.vue]
<script>
import 'animate.css'
</script>

<style>
@import url("animate.css");
</style>
```
```

The package can also be referenced as a string in the `css` property of your Nuxt configuration.

```
```ts [nuxt.config.ts]
export default defineNuxtConfig({
  css: ['animate.css']
})
```
```

## ## External Stylesheets

You can include external stylesheets in your application by adding a `link` element in the head section of your `nuxt.config` file. You can achieve this result using different methods. Note that local stylesheets can also be included like this.

You can manipulate the head with the `[`app.head`](/docs/api/nuxt-config#head)` property of your Nuxt configuration:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  app: {
    head: {
      link: [{ rel: 'stylesheet', href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css' }]
    }
  }
})
```
```

## ### Dynamically Adding Stylesheets

You can use the `useHead` composable to dynamically set a value in your head in your code.

```
:read-more{to="/docs/api/composables/use-head"}
```

```
```ts twoslash
useHead({
  link: [{ rel: 'stylesheet', href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css' }]
})
```
```

Nuxt uses ``unhead`` under the hood, and you can refer to its full documentation [here](https://unhead.unjs.io).

## ### Modifying The Rendered Head With A Nitro Plugin

If you need more advanced control, you can intercept the rendered html with a hook and modify the head programmatically.

Create a plugin in `~/server/plugins/my-plugin.ts` like this:

```
```ts twoslash [server/plugins/my-plugin.ts]
export default defineNitroPlugin((nitro) => {
  nitro.hooks.hook('render:html', (html) => {
    html.head.push('<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css">')
  })
})
```
```



```
...
```

External stylesheets are render-blocking resources: they must be loaded and processed before the browser renders the page. Web pages that contain unnecessarily large styles take longer to render. You can read more about it on [web.dev] (<https://web.dev/defer-non-critical-css>).

## ## Using Preprocessors

To use a preprocessor like SCSS, Sass, Less or Stylus, install it first.

```
::code-group
```

```
```bash [Sass & SCSS]
npm install -D sass
```
```

```
```bash [Less]
npm install -D less
```
```

```
```bash [Stylus]
npm install -D stylus
```
```

```
::
```

The natural place to write your stylesheets is the ``assets`` directory. You can then import your source files in your ``app.vue`` (or layouts files) using your preprocessor's syntax.

```
```vue [pages/app.vue]
<style lang="scss">
@use "~/assets/scss/main.scss";
</style>
```
```

Alternatively, you can use the ``css`` property of your Nuxt configuration.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  css: ['~/assets/scss/main.scss']
})
```
```

```
::tip
In both cases, the compiled stylesheets will be inlined in the HTML rendered by Nuxt.
::
```

If you need to inject code in pre-processed files, like a [sass partial](<https://sass-lang.com/documentation/at-rules/use#partials>) with color variables, you can do so with the vite [preprocessors options](<https://vitejs.dev/config/shared-options.html#css-preprocessoroptions>).

Create some partials in your ``assets`` directory:

```
::code-group
```

```
```scss [assets/_colors.scss]
$primary: #49240F;
$secondary: #E4A79D;
```
```

```
```sass [assets/_colors.sass]
$primary: #49240F
$secondary: #E4A79D
```
```

```
::
```

Then in your ``nuxt.config`` :

```
::code-group
```

```
```ts twoslash [SCSS]
export default defineNuxtConfig({
  vite: {
    css: {
      preprocessorOptions: {
        scss: {
          additionalData: '@use "~/assets/_colors.scss" as *;'
        }
      }
    }
  }
})
```
```

```

...

```ts twoslash [SASS]
export default defineNuxtConfig({
  vite: {
    css: {
      preprocessorOptions: {
        sass: {
          additionalData: '@use "~/assets/_colors.sass" as *\\n'
        }
      }
    }
  }
})
...

```

```

::

```

Nuxt uses Vite by default. If you wish to use webpack instead, refer to each preprocessor loader [documentation] (<https://webpack.js.org/loaders/sass-loader>).

Single File Components (SFC) Styling

One of the best things about Vue and SFC is how great it is at naturally dealing with styling. You can directly write CSS or preprocessor code in the style block of your components file, therefore you will have fantastic developer experience without having to use something like CSS-in-JS. However if you wish to use CSS-in-JS, you can find 3rd party libraries and modules that support it, such as [pinceau](<https://github.com/Tahul/pinceau>).

You can refer to the [Vue docs](<https://vuejs.org/api/sfc-css-features.html>) for a comprehensive reference about styling components in SFC.

Class And Style Bindings

You can leverage Vue SFC features to style your components with class and style attributes.

```

::code-group

```

```

```vue [Ref and Reactive]
<script setup lang="ts">
const isActive = ref(true)
const hasError = ref(false)
const classObject = reactive({
 active: true,
 'text-danger': false
})
</script>

<template>
 <div class="static" :class="{ active: isActive, 'text-danger': hasError }"></div>
 <div :class="classObject"></div>
</template>
```

```

```

```vue [Computed]
<script setup lang="ts">
const isActive = ref(true)
const error = ref(null)

const classObject = computed(() => ({
 active: isActive.value && !error.value,
 'text-danger': error.value && error.value.type === 'fatal'
}))
</script>

<template>
 <div :class="classObject"></div>
</template>
```

```

```

```vue [Array]
<script setup lang="ts">
const isActive = ref(true)
const errorClass = ref('text-danger')
</script>

<template>
 <div :class="[active: isActive], errorClass"></div>
</template>
```

```

```

```vue [Style]
<script setup lang="ts">
const activeColor = ref('red')

```

```
const fontSize = ref(30)
const styleObject = reactive({ color: 'red', fontSize: '13px' })
</script>
```

```
<template>
 <div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
 <div :style="[baseStyles, overridingStyles]"></div>
 <div :style="styleObject"></div>
</template>
```
```

```
::
```

Refer to the [Vue docs](https://vuejs.org/guide/essentials/class-and-style.html) for more information.

Dynamic Styles With `v-bind`

You can reference JavaScript variable and expression within your style blocks with the v-bind function. The binding will be dynamic, meaning that if the variable value changes, the style will be updated.

```
```vue
<script setup lang="ts">
const color = ref("red")
</script>

<template>
 <div class="text">hello</div>
</template>

<style>
.text {
 color: v-bind(color);
}
</style>
```
```

Scoped Styles

The scoped attribute allows you to style components in isolation. The styles declared with this attribute will only apply to this component.

```
```vue
<template>
 <div class="example">hi</div>
</template>

<style scoped>
.example {
 color: red;
}
</style>
```
```

CSS Modules

You can use [CSS Modules](https://github.com/css-modules/css-modules) with the module attribute. Access it with the injected `\$style` variable.

```
```vue
<template>
 <p :class="$style.red">This should be red</p>
</template>

<style module>
.red {
 color: red;
}
</style>
```
```

Preprocessors Support

SFC style blocks support preprocessors syntax. Vite come with built-in support for .scss, .sass, .less, .styl and .stylus files without configuration. You just need to install them first, and they will be available directly in SFC with the lang attribute.

```
::code-group
```

```
```vue [SCSS]
<style lang="scss">
 /* Write scss here */
</style>
```
```

```

```vue [Sass]
<style lang="sass">
 /* Write sass here */
</style>
```

```vue [LESS]
<style lang="less">
 /* Write less here */
</style>
```

```vue [Stylus]
<style lang="stylus">
 /* Write stylus here */
</style>
```

```

::

You can refer to the [Vite CSS docs](https://vitejs.dev/guide/features.html#css) and the [@vitejs/plugin-vue docs](https://github.com/vitejs/vite-plugin-vue/tree/main/packages/plugin-vue).
For webpack users, refer to the [vue loader docs](https://vue-loader.vuejs.org).

Using PostCSS

Nuxt comes with postcss built-in. You can configure it in your `nuxt.config` file.

```

```ts [nuxt.config.ts]
export default defineNuxtConfig({
 postcss: {
 plugins: {
 'postcss-nested': {},
 "postcss-custom-media": {}
 }
 }
})
```

```

For proper syntax highlighting in SFC, you can use the postcss lang attribute.

```

```vue
<style lang="postcss">
 /* Write postcss here */
</style>
```

```

By default, Nuxt comes with the following plugins already pre-configured:

- [postcss-import](https://github.com/postcss/postcss-import): Improves the `@import` rule
- [postcss-url](https://github.com/postcss/postcss-url): Transforms `url()` statements
- [autoprefixer](https://github.com/postcss/autoprefixer): Automatically adds vendor prefixes
- [cssnano](https://cssnano.github.io/cssnano): Minification and purge

Leveraging Layouts For Multiple Styles

If you need to style different parts of your application completely differently, you can use layouts. Use different styles for different layouts.

```

```vue
<template>
 <div class="default-layout">
 <h1>Default Layout</h1>
 <slot />
 </div>
</template>

<style>
.default-layout {
 color: red;
}
</style>
```

```

:read-more{to="/docs/guide/directory-structure/layouts"}

Third Party Libraries And Modules

Nuxt isn't opinionated when it comes to styling and provides you with a wide variety of options. You can use any styling tool that you want, such as popular libraries like [UnoCSS](https://unocss.dev) or [Tailwind CSS](https://tailwindcss.com).

The community and the Nuxt team have developed plenty of Nuxt modules to make the integration easier.

You can discover them on the [modules section](/modules) of the website.
Here are a few modules to help you get started:

- [UnoCSS](/modules/unocss): Instant on-demand atomic CSS engine
- [Tailwind CSS](/modules/tailwindcss): Utility-first CSS framework
- [Fontaine](https://github.com/nuxt-modules/fontaine): Font metric fallback
- [Pinceau](https://github.com/Tahul/pinceau): Adaptable styling framework
- [Nuxt UI](https://ui.nuxt.com): A UI Library for Modern Web Apps
- [Panda CSS](https://panda-css.com/docs/installation/nuxt): CSS-in-JS engine that generates atomic CSS at build time

Nuxt modules provide you with a good developer experience out of the box, but remember that if your favorite tool doesn't have a module, it doesn't mean that you can't use it with Nuxt! You can configure it yourself for your own project. Depending on the tool, you might need to use a [Nuxt plugin](/docs/guide/directory-structure/plugins) and/or [make your own module](/docs/guide/going-further/modules). Share them with the [community](/modules) if you do!

Easily Load Webfonts

You can use [the Nuxt Google Fonts module](https://github.com/nuxt-modules/google-fonts) to load Google Fonts.

If you are using [UnoCSS](https://unocss.dev/integrations/nuxt), note that it comes with a [web fonts presets](https://unocss.dev/presets/web-fonts) to conveniently load fonts from common providers, including Google Fonts and more.

Advanced

Transitions

Nuxt comes with the same `` element that Vue has, and also has support for the experimental [View Transitions API](/docs/getting-started/transitions#view-transitions-api-experimental).

:read-more{to="/docs/getting-started/transitions"}

Font Advanced Optimization

We would recommend using [Fontaine](https://github.com/nuxt-modules/fontaine) to reduce your [CLS](https://web.dev/cls). If you need something more advanced, consider creating a Nuxt module to extend the build process or the Nuxt runtime.

::tip

Always remember to take advantage of the various tools and techniques available in the Web ecosystem at large to make styling your application easier and more efficient. Whether you're using native CSS, a preprocessor, postcss, a UI library or a module, Nuxt has got you covered. Happy styling!

::

LCP Advanced optimizations

You can do the following to speed-up the download of your global CSS files:

- Use a CDN so the files are physically closer to your users
- Compress your assets, ideally using Brotli
- Use HTTP2/HTTP3 for delivery
- Host your assets on the same domain (do not use a different subdomain)

Most of these things should be done for you automatically if you're using modern platforms like Cloudflare, Netlify or Vercel.

You can find an LCP optimization guide on [web.dev](https://web.dev/optimize-lcp).

If all of your CSS is inlined by Nuxt, you can (experimentally) completely stop external CSS files from being referenced in your rendered HTML.

You can achieve that with a hook, that you can place in a module, or in your Nuxt configuration file.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  hooks: {
    'build:manifest': (manifest) => {
      // find the app entry, css list
      const css = manifest['node_modules/nuxt/dist/app/entry.js']?.css
      if (css) {
        // start from the end of the array and go to the beginning
        for (let i = css.length - 1; i >= 0; i--) {
          // if it starts with 'entry', remove it from the list
          if (css[i].startsWith('entry')) css.splice(i, 1)
        }
      }
    },
  },
})
``
```

```
---
title: "Nuxt Kit"
description: "@nuxt/kit provides features for module authors."
---
```

Nuxt Kit provides composable utilities to make interacting with [Nuxt Hooks](/docs/api/advanced/hooks), the [Nuxt Interface](/docs/guide/going-further/internals#the-nuxt-interface) and developing [Nuxt Modules](/docs/guide/going-further/modules) super easy.

```
::read-more{to="/docs/api/kit"}
Discover all Nuxt Kit utilities.
::
```

Usage

Install Dependency

You can install the latest Nuxt Kit by adding it to the `dependencies` section of your `package.json`. However, please consider always explicitly installing the `@nuxt/kit` package even if it is already installed by Nuxt.

```
::note
`@nuxt/kit` and `@nuxt/schema` are key dependencies for Nuxt. If you are installing it separately, make sure that the
versions of `@nuxt/kit` and `@nuxt/schema` are equal to or greater than your `nuxt` version to avoid any unexpected behavior.
::
```

```
```json [package.json]
{
 "dependencies": {
 "@nuxt/kit": "npm:@nuxt/kit-nightly@latest"
 }
}
```
```

Import Kit Utilities

```
```js [test.mjs]
import { useNuxt } from '@nuxt/kit'
```
```

```
::read-more{to="/docs/api/kit"}
```

```
::note
Nuxt Kit utilities are only available for modules and not meant to be imported in runtime (components, Vue composables,
pages, plugins, or server routes).
::
```

Nuxt Kit is an [esm-only package](/docs/guide/concepts/esm) meaning that you **cannot** `require('@nuxt/kit')`. As a workaround, use dynamic import in the CommonJS context:

```
```js [test.cjs]
// This does NOT work!
// const kit = require('@nuxt/kit')
async function main() {
 const kit = await import('@nuxt/kit')
}
main()
```
```

```
---
title: "useId"
description: Generate an SSR-friendly unique identifier that can be passed to accessibility attributes.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/id.ts
    size: xs
---
```

```
::important
This composable is available since [Nuxt v3.10](/blog/v3-10#ssr-safe-accessible-unique-id-creation).
::
```

`useId` generates an SSR-friendly unique identifier that can be passed to accessibility attributes.

Call `useId` at the top level of your component to generate a unique string identifier:

```
``vue [components/EmailField.vue]
<script setup lang="ts">
  const id = useId()
</script>

<template>
  <div>
    <label :for="id">Email</label>
    <input :id="id" name="email" type="email" />
  </div>
</template>
``
```

```
::note
`useId` must be used in a component with a single root element, as it uses this root element's attributes to pass the id from
server to client.
::
```

Parameters

`useId` does not take any parameters.

Returns

`useId` returns a unique string associated with this particular `useId` call in this particular component.

```
---
title: '<NuxtLoadingIndicator>'
description: 'Display a progress bar between page navigations.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-loading-indicator.ts
    size: xs
---
```

Usage

Add `<NuxtLoadingIndicator>` in your `[`app.vue`](/docs/guide/directory-structure/app)` or `[`layouts/`](/docs/guide/directory-structure/layouts)`.

```
```vue [app.vue]
<template>
 <NuxtLoadingIndicator />
 <NuxtLayout>
 <NuxtPage />
 </NuxtLayout>
</template>
```
```

`:link-example{to="/docs/examples/routing/pages"}`

Slots

You can pass custom HTML or components through the loading indicator's default slot.

Props

- ``color``: The color of the loading bar. It can be set to ``false`` to turn off explicit color styling.
- ``errorColor``: The color of the loading bar when ``error`` is set to ``true``.
- ``height``: Height of the loading bar, in pixels (default ``3``).
- ``duration``: Duration of the loading bar, in milliseconds (default ``2000``).
- ``throttle``: Throttle the appearing and hiding, in milliseconds (default ``200``).
- ``estimatedProgress``: By default Nuxt will back off as it approaches 100%. You can provide a custom function to customize the progress estimation, which is a function that receives the duration of the loading bar (above) and the elapsed time. It should return a value between 0 and 100.

`::note`
This component is optional. `:br`
To achieve full customization, you can implement your own one based on [its source code]
(<https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-loading-indicator.ts>).
`::`

`::note`
You can hook into the underlying indicator instance using [the ``useLoadingIndicator`` composible](/docs/api/composables/use-loading-indicator), which will allow you to trigger start/finish events yourself.
`::`

`::tip`
The loading indicator's speed gradually decreases after reaching a specific point controlled by ``estimatedProgress``. This adjustment provides a more accurate reflection of longer page loading times and prevents the indicator from prematurely showing 100% completion.
`::`


```
---
title: "<NuxtErrorBoundary>"
description: The <NuxtErrorBoundary> component handles client-side errors happening in its default slot.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-error-boundary.ts
    size: xs
---
```

```
:::tip
The `<NuxtErrorBoundary>` uses Vue's [`onErrorCaptured`](https://vuejs.org/api/composition-api-lifecycle.html#onerrorcaptured) hook under the hood.
:::
```

Events

- `@error`: Event emitted when the default slot of the component throws an error.

```
```vue
<template>
 <NuxtErrorBoundary @error="logSomeError">
 <!-- ... -->
 </NuxtErrorBoundary>
</template>
```
```

Slots

- `#error`: Specify a fallback content to display in case of error.

```
```vue
<template>
 <NuxtErrorBoundary>
 <!-- ... -->
 <template #error="{ error }">
 <p>An error occurred: {{ error }}</p>
 </template>
 </NuxtErrorBoundary>
</template>
```
```

:read-more{to="/docs/getting-started/error-handling"}

```
---
title: useLazyAsyncData
description: This wrapper around useAsyncData triggers navigation immediately.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
    size: xs
---
```

Description

By default, [`useAsyncData`](/docs/api/composables/use-async-data) blocks navigation until its async handler is resolved. `useLazyAsyncData` provides a wrapper around [`useAsyncData`](/docs/api/composables/use-async-data) that triggers navigation before the handler is resolved by setting the `lazy` option to `true`.

```
::note
`useLazyAsyncData` has the same signature as [useAsyncData](/docs/api/composables/use-async-data).
::
```

```
:read-more{to="/docs/api/composables/use-async-data"}
```

Example

```
``vue [pages/index.vue]
<script setup lang="ts">
/* Navigation will occur before fetching is complete.
   Handle 'pending' and 'error' states directly within your component's template
*/
const { status, data: count } = await useLazyAsyncData('count', () => $fetch('/api/count'))

watch(count, (newCount) => {
  // Because count might start out null, you won't have access
  // to its contents immediately, but you can watch it.
})
</script>

<template>
  <div>
    {{ status === 'pending' ? 'Loading' : count }}
  </div>
</template>
``
```

```
::warning
`useLazyAsyncData` is a reserved function name transformed by the compiler, so you should not name your own function
`useLazyAsyncData`.
::
```

```
:read-more{to="/docs/getting-started/data-fetching"}
```

```
---
title: "node_modules"
description: "The package manager stores the dependencies of your project in the node_modules/ directory."
head.title: "node_modules/"
navigation.icon: i-ph-folder-duotone
---
```

The package manager ([`npm`](https://docs.npmjs.com/cli/commands/npm) or [`yarn`](https://yarnpkg.com) or [`pnpm`](https://pnpm.io/cli/install) or [`bun`](https://bun.sh/package-manager)) creates this directory to store the dependencies of your project.

::important

This directory should be added to your [`.gitignore`](/docs/guide/directory-structure/gitignore) file to avoid pushing the dependencies to your repository.

::

```
---
title: "Module Author Guide"
description: "Learn how to create a Nuxt Module to integrate, enhance or extend any Nuxt applications."
image: '/socials/module-author-guide.jpg'
---
```

Nuxt's [configuration](/docs/api/nuxt-config) and [hooks](/docs/guide/going-further/hooks) systems make it possible to customize every aspect of Nuxt and add any integration you might need (Vue plugins, CMS, server routes, components, logging, etc.).

****Nuxt Modules**** are functions that sequentially run when starting Nuxt in development mode using ``nuxi dev`` or building a project for production with ``nuxi build``. With modules, you can encapsulate, properly test, and share custom solutions as npm packages without adding unnecessary boilerplate to your project, or requiring changes to Nuxt itself.

Quick Start

We recommend you get started with Nuxt Modules using our [starter template](https://github.com/nuxt/starter/tree/module):

::code-group

```
```bash [npm]
npx nuxi init -t module my-module
```
```

```
```bash [yarn]
yarn dlx nuxi init -t module my-module
```
```

```
```bash [pnpm]
pnpm dlx nuxi init -t module my-module
```
```

```
```bash [bun]
bun x nuxi init -t module my-module
```
```

::

This will create a ``my-module`` project with all the boilerplate necessary to develop and publish your module.

****Next steps:****

1. Open ``my-module`` in your IDE of choice
2. Install dependencies using your favorite package manager
3. Prepare local files for development using ``npm run dev:prepare``
4. Follow this document to learn more about Nuxt Modules

Using the Starter

Learn how to perform basic tasks with the module starter.

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/navigating-the-official-starter-template?friend=nuxt" target="_blank"}

Watch Vue School video about Nuxt module starter template.

::

How to Develop

While your module source code lives inside the ``src`` directory, in most cases, to develop a module, you need a Nuxt application. That's what the ``playground`` directory is about. It's a Nuxt application you can tinker with that is already configured to run with your module.

You can interact with the playground like with any Nuxt application.

- Launch its development server with ``npm run dev``, it should reload itself as you make changes to your module in the ``src`` directory
- Build it with ``npm run dev:build``

::note

All other ``nuxi`` commands can be used against the ``playground`` directory (e.g. ``nuxi <COMMAND> playground``). Feel free to declare additional ``dev:*`` scripts within your ``package.json`` referencing them for convenience.

::

How to Test

The module starter comes with a basic test suite:

- A linter powered by [ESLint](https://eslint.org), run it with ``npm run lint``
- A test runner powered by [Vitest](https://vitest.dev), run it with ``npm run test`` or ``npm run test:watch``

::tip

Feel free to augment this default test strategy to better suit your needs.

::

How to Build

Nuxt Modules come with their own builder provided by [`@nuxt/module-builder`](https://github.com/nuxt/module-builder#readme). This builder doesn't require any configuration on your end, supports TypeScript, and makes sure your assets are properly bundled to be distributed to other Nuxt applications.

You can build your module by running ``npm run prepack``.

::tip

While building your module can be useful in some cases, most of the time you won't need to build it on your own: the ``playground`` takes care of it while developing, and the release script also has you covered when publishing.

::

How to Publish

::important

Before publishing your module to npm, makes sure you have an [npmjs.com](https://www.npmjs.com) account and that you're authenticated to it locally with ``npm login``.

::

While you can publish your module by bumping its version and using the ``npm publish`` command, the module starter comes with a release script that helps you make sure you publish a working version of your module to npm and more.

To use the release script, first, commit all your changes (we recommend you follow [Conventional Commits](https://www.conventionalcommits.org) to also take advantage of automatic version bump and changelog update), then run the release script with ``npm run release``.

When running the release script, the following will happen:

- First, it will run your test suite by:
 - Running the linter (``npm run lint``)
 - Running unit, integration, and e2e tests (``npm run test``)
 - Building the module (``npm run prepack``)
- Then, if your test suite went well, it will proceed to publish your module by:
 - Bumping your module version and generating a changelog according to your Conventional Commits
 - Publishing the module to npm (for that purpose, the module will be built again to ensure its updated version number is taken into account in the published artifact)
 - Pushing a git tag representing the newly published version to your git remote origin

::tip

As with other scripts, feel free to fine-tune the default ``release`` script in your ``package.json`` to better suit your needs.

::

Developing Modules

Nuxt Modules come with a variety of powerful APIs and patterns allowing them to alter a Nuxt application in pretty much any way possible. This section teaches you how to take advantage of those.

Module Anatomy

We can consider two kinds of Nuxt Modules:

- published modules are distributed on npm - you can see a list of some community modules on [the Nuxt website](/modules).
- "local" modules, they exist within a Nuxt project itself, either [inlined in Nuxt config](/docs/api/nuxt-config#modules) or as part of [the ``modules`` directory](/docs/guide/directory-structure/modules).

In either case, their anatomy is similar.

Module Definition

::note

When using the starter, your module definition is available at ``src/module.ts``.

::

The module definition is the entry point of your module. It's what gets loaded by Nuxt when your module is referenced within a Nuxt configuration.

At a low level, a Nuxt Module definition is a simple, potentially asynchronous, function accepting inline user options and a ``nuxt`` object to interact with Nuxt.

```ts

```
export default function (inlineOptions, nuxt) {
 // You can do whatever you like here..
 console.log(inlineOptions.token) // `123`
 console.log(nuxt.options.dev) // `true` or `false`
 nuxt.hook('ready', async nuxt => {
 console.log('Nuxt is ready')
 })
}
```

You can get type-hint support for this function using the higher-level ``defineNuxtModule`` helper provided by [Nuxt Kit]

(/docs/guide/going-further/kit).

```
``ts
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule((options, nuxt) => {
 nuxt.hook('pages:extend', pages => {
 console.log(`Discovered ${pages.length} pages`)
 })
})
``
```

However, **we do not recommend** using this low-level function definition. Instead, to define a module, **we recommend** using the object-syntax with `meta` property to identify your module, especially when publishing to npm.

This helper makes writing Nuxt modules more straightforward by implementing many common patterns needed by modules, guaranteeing future compatibility and improving the experience for both module authors and users.

```
``ts
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule({
 meta: {
 // Usually the npm package name of your module
 name: '@nuxtjs/example',
 // The key in `nuxt.config` that holds your module options
 configKey: 'sample',
 // Compatibility constraints
 compatibility: {
 // Semver version of supported nuxt versions
 nuxt: '>=3.0.0'
 }
 },
 // Default configuration options for your module, can also be a function returning those
 defaults: {},
 // Shorthand sugar to register Nuxt hooks
 hooks: {},
 // The function holding your module logic, it can be asynchronous
 setup(moduleOptions, nuxt) {
 // ...
 }
})
``
```

Ultimately `defineNuxtModule` returns a wrapper function with the lower level `(inlineOptions, nuxt)` module signature. This wrapper function applies defaults and other necessary steps before calling your `setup` function:

- Support `defaults` and `meta.configKey` for automatically merging module options
- Type hints and automated type inference
- Add shims for basic Nuxt 2 compatibility
- Ensure module gets installed only once using a unique key computed from `meta.name` or `meta.configKey`
- Automatically register Nuxt hooks
- Automatically check for compatibility issues based on module meta
- Expose `getOptions` and `getMeta` for internal usage of Nuxt
- Ensuring backward and upward compatibility as long as the module is using `defineNuxtModule` from the latest version of `@nuxt/kit`
- Integration with module builder tooling

#### #### Runtime Directory

```
::note
When using the starter, the runtime directory is available at `src/runtime`.
::
```

Modules, like everything in a Nuxt configuration, aren't included in your application runtime. However, you might want your module to provide, or inject runtime code to the application it's installed on. That's what the runtime directory enables you to do.

Inside the runtime directory, you can provide any kind of assets related to the Nuxt App:

- Vue components
- Composables
- [Nuxt plugins](/docs/guide/directory-structure/plugins)

To the [server engine](/docs/guide/concepts/server-engine), Nitro:

- API routes
- Middlewares
- Nitro plugins

Or any other kind of asset you want to inject in users' Nuxt applications:

- Stylesheets
- 3D models
- Images
- etc.

You'll then be able to inject all those assets inside the application from your [module definition](#module-definition).

```
::tip
Learn more about asset injection in [the recipes section](#recipes).
::
```

```
::warning
Published modules cannot leverage auto-imports for assets within their runtime directory. Instead, they have to import them
explicitly from `#imports` or alike.
```

Indeed, auto-imports are not enabled for files within `node\_modules` (the location where a published module will eventually live) for performance reasons.

If you are using the module starter, auto-imports will not be enabled in your playground either.

```
::
```

### ### Tooling

Modules come with a set of first-party tools to help you with their development.

#### #### `@nuxt/module-builder`

[Nuxt Module Builder](https://github.com/nuxt/module-builder#readme) is a zero-configuration build tool taking care of all the heavy lifting to build and ship your module. It ensures proper compatibility of your module build artifact with Nuxt applications.

#### #### `@nuxt/kit`

[Nuxt Kit](/docs/guide/going-further/kit) provides composable utilities to help your module interact with Nuxt applications. It's recommended to use Nuxt Kit utilities over manual alternatives whenever possible to ensure better compatibility and code readability of your module.

```
:read-more{to="/docs/guide/going-further/kit"}
```

#### #### `@nuxt/test-utils`

[Nuxt Test Utils](/docs/getting-started/testing) is a collection of utilities to help set up and run Nuxt applications within your module tests.

### ### Recipes

Find here common patterns used to author modules.

#### #### Altering Nuxt Configuration

Nuxt configuration can be read and altered by modules. Here's an example of a module enabling an experimental feature.

```
``js
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule({
 setup (options, nuxt) {
 // We create the `experimental` object if it doesn't exist yet
 nuxt.options.experimental ||= {}
 nuxt.options.experimental.componentIslands = true
 }
})
``
```

When you need to handle more complex configuration alterations, you should consider using [defu](https://github.com/unjs/defu).

```
::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/extending-and-altering-nuxt-configuration-and-options?friend=nuxt" target="_blank"}
Watch Vue School video about altering Nuxt configuration.
::
```

#### #### Exposing Options to Runtime

Because modules aren't part of the application runtime, their options aren't either. However, in many cases, you might need access to some of these module options within your runtime code. We recommend exposing the needed config using Nuxt's [`runtimeConfig`](/docs/api/nuxt-config#runtimeconfig).

```
<!-- TODO: Update after #18466 (or equivalent) -->
```

```
``js
import { defineNuxtModule } from '@nuxt/kit'
import { defu } from 'defu'

export default defineNuxtModule({
 setup (options, nuxt) {
 nuxt.options.runtimeConfig.public.myModule = defu(nuxt.options.runtimeConfig.public.myModule, {
```

```

 foo: options.foo
 })
}
})
...

```

Note that we use [`defu`](<https://github.com/unjs/defu>) to extend the public runtime configuration the user provides instead of overwriting it.

You can then access your module options in a plugin, component, the application like any other runtime configuration:

```

```js
const options = useRuntimeConfig().public.myModule
...

::warning
Be careful not to expose any sensitive module configuration on the public runtime config, such as private API keys, as they will end up in the public bundle.
::

:read-more{to="/docs/guide/going-further/runtime-config"}

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/passing-and-exposing-module-options?friend=nuxt" target="_blank"}
Watch Vue School video about passing and exposing Nuxt module options.
::

```

Injecting Plugins With `addPlugin`

Plugins are a common way for a module to add runtime logic. You can use the `addPlugin` utility to register them from your module.

```

```js
import { defineNuxtModule, addPlugin, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
 setup (options, nuxt) {
 // Create resolver to resolve relative paths
 const { resolve } = createResolver(import.meta.url)

 addPlugin(resolve('./runtime/plugin'))
 }
})
...

:read-more{to="/docs/guide/going-further/kit"}

```

#### #### Injecting Vue Components With `addComponent`

If your module should provide Vue components, you can use the `addComponent` utility to add them as auto-imports for Nuxt to resolve.

```

```js
import { defineNuxtModule, addComponent } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)

    // From the runtime directory
    addComponent({
      name: 'MySuperComponent', // name of the component to be used in vue templates
      export: 'MySuperComponent', // (optional) if the component is a named (rather than default) export
      filePath: resolver.resolve('runtime/components/MySuperComponent.vue')
    })

    // From a library
    addComponent({
      name: 'MyAwesomeComponent', // name of the component to be used in vue templates
      export: 'MyAwesomeComponent', // (optional) if the component is a named (rather than default) export
      filePath: '@vue/awesome-components'
    })
  }
})
...

```

Alternatively, you can add an entire directory by using `addComponentsDir`.

```

```ts
import { defineNuxtModule, addComponentsDir } from '@nuxt/kit'

export default defineNuxtModule({
 setup(options, nuxt) {

```



```

const resolver = createResolver(import.meta.url)

addComponentsDir({
 path: resolver.resolve('runtime/components')
})
}
})
...

```

#### #### Injecting Composables With `addImports` and `addImportsDir`

If your module should provide composables, you can use the `addImports` utility to add them as auto-imports for Nuxt to resolve.

```

```ts
import { defineNuxtModule, addImports, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)

    addImports({
      name: 'useComposable', // name of the composable to be used
      as: 'useComposable',
      from: resolver.resolve('runtime/composables/useComposable') // path of composable
    })
  }
})
...

```

Alternatively, you can add an entire directory by using `addImportsDir`.

```

```ts
import { defineNuxtModule, addImportsDir, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
 setup(options, nuxt) {
 const resolver = createResolver(import.meta.url)

 addImportsDir(resolver.resolve('runtime/composables'))
 }
})
...

```

#### #### Injecting Server Routes With `addServerHandler`

```

```ts
import { defineNuxtModule, addServerHandler, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)

    addServerHandler({
      route: '/api/hello',
      handler: resolver.resolve('./runtime/server/api/hello/index.get')
    })
  }
})
...

```

You can also add a dynamic server route:

```

```ts
import { defineNuxtModule, addServerHandler, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
 setup(options, nuxt) {
 const resolver = createResolver(import.meta.url)

 addServerHandler({
 route: '/api/hello/:name',
 handler: resolver.resolve('./runtime/server/api/hello/[name].get')
 })
 }
})
...

```

#### #### Injecting Other Assets

If your module should provide other kinds of assets, they can also be injected. Here's a simple example module injecting a stylesheet through Nuxt's `css` array.

```

```js
import { defineNuxtModule, addPlugin, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  setup (options, nuxt) {
    const { resolve } = createResolver(import.meta.url)

    nuxt.options.css.push(resolve('./runtime/style.css'))
  }
})
```

```

And a more advanced one, exposing a folder of assets through [Nitro](/docs/guide/concepts/server-engine)'s `publicAssets` option:

```

```js
import { defineNuxtModule, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  setup (options, nuxt) {
    const { resolve } = createResolver(import.meta.url)

    nuxt.hook('nitro:config', async (nitroConfig) => {
      nitroConfig.publicAssets ||= []
      nitroConfig.publicAssets.push({
        dir: resolve('./runtime/public'),
        maxAge: 60 * 60 * 24 * 365 // 1 year
      })
    })
  }
})
```

```

#### #### Using Other Modules in Your Module

If your module depends on other modules, you can add them by using Nuxt Kit's `installModule` utility. For example, if you wanted to use Nuxt Tailwind in your module, you could add it as below:

```

```ts
import { defineNuxtModule, createResolver, installModule } from '@nuxt/kit'

export default defineNuxtModule<ModuleOptions>({
  async setup (options, nuxt) {
    const { resolve } = createResolver(import.meta.url)

    // We can inject our CSS file which includes Tailwind's directives
    nuxt.options.css.push(resolve('./runtime/assets/styles.css'))

    await installModule('@nuxtjs/tailwindcss', {
      // module configuration
      exposeConfig: true,
      config: {
        darkMode: 'class',
        content: {
          files: [
            resolve('./runtime/components/**/*.vue,mjs,ts'),
            resolve('./runtime/*.mjs,js,ts')
          ]
        }
      }
    })
  }
})
```

```

#### #### Using Hooks

[Lifecycle hooks](/docs/guide/going-further/hooks) allow you to expand almost every aspect of Nuxt. Modules can hook to them programmatically or through the `hooks` map in their definition.

```

```js
import { defineNuxtModule, addPlugin, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  // Hook to the `app:error` hook through the `hooks` map
  hooks: {
    'app:error': (err) => {
      console.info(`This error happened: ${err}`);
    }
  },
  setup (options, nuxt) {
    // Programmatically hook to the `pages:extend` hook
    nuxt.hook('pages:extend', (pages) => {

```

```

        console.info(`Discovered ${pages.length} pages`);
    })
  }
})
...

```

```

:read-more{to="/docs/api/advanced/hooks"}

```

```

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/nuxt-lifecycle-hooks?friend=nuxt" target="_blank"}
Watch Vue School video about using Nuxt lifecycle hooks in modules.
::

```

```

::note
**Module cleanup**
:br
:br
If your module opens, handles, or starts a watcher, you should close it when the Nuxt lifecycle is done. The `close` hook is
available for this.

```

```

...`ts
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule({
  setup (options, nuxt) {
    nuxt.hook('close', async nuxt => {
      // Your custom code here
    })
  }
})
...
::

```

Adding Templates/Virtual Files

If you need to add a virtual file that can be imported into the user's app, you can use the `addTemplate` utility.

```

...`ts
import { defineNuxtModule, addTemplate } from '@nuxt/kit'

export default defineNuxtModule({
  setup (options, nuxt) {
    // The file is added to Nuxt's internal virtual file system and can be imported from '#build/my-module-feature.mjs'
    addTemplate({
      filename: 'my-module-feature.mjs',
      getContents: () => 'export const myModuleFeature = () => "hello world !"'
    })
  }
})
...

```

Adding Type Declarations

You might also want to add a type declaration to the user's project (for example, to augment a Nuxt interface or provide a global type of your own). For this, Nuxt provides the `addTypeTemplate` utility that both writes a template to the disk and adds a reference to it in the generated `nuxt.d.ts` file.

If your module should augment types handled by Nuxt, you can use `addTypeTemplate` to perform this operation:

```

...`js
import { defineNuxtModule, addTemplate, addTypeTemplate } from '@nuxt/kit'

export default defineNuxtModule({
  setup (options, nuxt) {
    addTypeTemplate({
      filename: 'types/my-module.d.ts',
      getContents: () => `// Generated by my-module
      interface MyModuleNitroRules {
        myModule?: { foo: 'bar' }
      }
      declare module 'nitro/types' {
        interface NitroRouteRules extends MyModuleNitroRules {}
        interface NitroRouteConfig extends MyModuleNitroRules {}
      }
      export {}`
    })
  }
})
...

```

If you need more granular control, you can use the `prepare:types` hook to register a callback that will inject your types.

```

...`ts
const template = addTemplate({ /* template options */ })

```

```
nuxt.hook('prepare:types', ({ references }) => {
  references.push({ path: template.dst })
})
```,
```

#### #### Updating Templates

If you need to update your templates/virtual files, you can leverage the ``updateTemplates`` utility like this :

```
``ts
nuxt.hook('builder:watch', async (event, path) => {
 if (path.includes('my-module-feature.config')) {
 // This will reload the template that you registered
 updateTemplates({ filter: t => t.filename === 'my-module-feature.mjs' })
 }
})
```,
```

Testing

Testing helps ensuring your module works as expected given various setup. Find in this section how to perform various kinds of tests against your module.

Unit and Integration

`::tip`
We're still discussing and exploring how to ease unit and integration testing on Nuxt Modules.

[Check out this RFC to join the conversation](<https://github.com/nuxt/nuxt/discussions/18399>).
`::`

End to End

[Nuxt Test Utils](/docs/getting-started/testing) is the go-to library to help you test your module in an end-to-end way. Here's the workflow to adopt with it:

1. Create a Nuxt application to be used as a "fixture" inside ``test/fixtures/*``
2. Setup Nuxt with this fixture inside your test file
3. Interact with the fixture using utilities from ``@nuxt/test-utils`` (e.g. fetching a page)
4. Perform checks related to this fixture (e.g. "HTML contains ...")
5. Repeat

In practice, the fixture:

```
``js [test/fixtures/ssr/nuxt.config.ts]
// 1. Create a Nuxt application to be used as a "fixture"
import MyModule from '../../src/module'

export default defineNuxtConfig({
  ssr: true,
  modules: [
    MyModule
  ]
})
```,
```

And its test:

```
``js [test/rendering.ts]
import { describe, it, expect } from 'vitest'
import { fileURLToPath } from 'node:url'
import { setup, $fetch } from '@nuxt/test-utils'

describe('ssr', async () => {
 // 2. Setup Nuxt with this fixture inside your test file
 await setup({
 rootDir: fileURLToPath(new URL('./fixtures/ssr', import.meta.url)),
 })

 it('renders the index page', async () => {
 // 3. Interact with the fixture using utilities from `@nuxt/test-utils`
 const html = await $fetch('/')

 // 4. Perform checks related to this fixture
 expect(html).toContain('<div>ssr</div>')
 })
})

// 5. Repeat
describe('csr', async () => { /* ... */ })
```,
```

`::tip`

An example of such a workflow is available on [the module starter] (<https://github.com/nuxt/starter/blob/module/test/basic.test.ts>).
::

Manual QA With Playground and Externally

Having a playground Nuxt application to test your module when developing it is really useful. [The module starter integrates one for that purpose](#how-to-develop).

You can test your module with other Nuxt applications (applications that are not part of your module repository) locally. To do so, you can use [``npm pack``] (<https://docs.npmjs.com/cli/commands/npm-pack>) command, or your package manager equivalent, to create a tarball from your module. Then in your test project, you can add your module to ``package.json`` packages as: ``"my-module": "file:/path/to/tarball.tgz"``.

After that, you should be able to reference ``my-module`` like in any regular project.

Best Practices

With great power comes great responsibility. While modules are powerful, here are some best practices to keep in mind while authoring modules to keep applications performant and developer experience great.

Async Modules

As we've seen, Nuxt Modules can be asynchronous. For example, you may want to develop a module that needs fetching some API or calling an async function.

However, be careful with asynchronous behaviors as Nuxt will wait for your module to setup before going to the next module and starting the development server, build process, etc. Prefer deferring time-consuming logic to Nuxt hooks.

```
::warning
If your module takes more than **1 second** to setup, Nuxt will emit a warning about it.
::
```

Always Prefix Exposed Interfaces

Nuxt Modules should provide an explicit prefix for any exposed configuration, plugin, API, composable, or component to avoid conflict with other modules and internals.

Ideally, you should prefix them with your module's name (e.g. if your module is called ``nuxt-foo``, expose ``<FooButton>`` and ``useFooBar()`` and **not** ``<Button>`` and ``useBar()``).

Be TypeScript Friendly

Nuxt 3, has first-class TypeScript integration for the best developer experience.

Exposing types and using TypeScript to develop modules benefits users even when not using TypeScript directly.

Avoid CommonJS Syntax

Nuxt 3, relies on native ESM. Please read [Native ES Modules](/docs/guide/concepts/esm) for more information.

Document Module Usage

Consider documenting module usage in the readme file:

- Why use this module?
- How to use this module?
- What does this module do?

Linking to the integration website and documentation is always a good idea.

Provide a StackBlitz Demo or Boilerplate

It's a good practice to make a minimal reproduction with your module and [StackBlitz](<https://nuxt.new/s/v3>) that you add to your module readme.

This not only provides potential users of your module a quick and easy way to experiment with the module but also an easy way for them to build minimal reproductions they can send you when they encounter issues.

Do Not Advertise With a Specific Nuxt Version

Nuxt 3, Nuxt Kit, and other new toolings are made to have both forward and backward compatibility in mind.

Please use "X for Nuxt" instead of "X for Nuxt 3" to avoid fragmentation in the ecosystem and prefer using ``meta.compatibility`` to set Nuxt version constraints.

Stick With Starter Defaults

The module starter comes with a default set of tools and configurations (e.g. ESLint configuration). If you plan on open-sourcing your module, sticking with those defaults ensures your module shares a consistent coding style with other [community modules](/modules) out there, making it easier for others to contribute.

Ecosystem

[Nuxt Module ecosystem](/modules) represents more than 15 million monthly NPM downloads and provides extended functionalities and integrations with all sort of tools. You can be part of this ecosystem!

```
::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/exploring-nuxt-modules-ecosystem-and-module-types?friend=nuxt" target="_blank"}
Watch Vue School video about Nuxt module types.
::
```

Module Types

****Official modules**** are modules prefixed (scoped) with `@nuxt/`` (e.g. [`@nuxt/content``](https://content.nuxtjs.org)). They are made and maintained actively by the Nuxt team. Like with the framework, contributions from the community are more than welcome to help make them better!

****Community modules**** are modules prefixed (scoped) with `@nuxtjs/`` (e.g. [`@nuxtjs/tailwindcss``](https://tailwindcss.nuxtjs.org)). They are proven modules made and maintained by community members. Again, contributions are welcome from anyone.

****Third party and other community modules**** are modules (often) prefixed with `nuxt-``. Anyone can make them, using this prefix allows these modules to be discoverable on npm. This is the best starting point to draft and try an idea!

****Private or personal modules**** are modules made for your own use case or company. They don't need to follow any naming rules to work with Nuxt and are often seen scoped under an npm organization (e.g. `@my-company/nuxt-auth``)

Listing Your Community Module

Any community modules are welcome to be listed on [the module list](/modules). To be listed, [open an issue in the nuxt/modules](https://github.com/nuxt/modules/issues/new?template=module_request.yml) repository. The Nuxt team can help you to apply best practices before listing.

Joining `nuxt-modules` and `@nuxtjs/`

By moving your modules to [nuxt-modules](https://github.com/nuxt-modules), there is always someone else to help, and this way, we can join forces to make one perfect solution.

If you have an already published and working module, and want to transfer it to `nuxt-modules``, [open an issue in nuxt/modules](https://github.com/nuxt/modules/issues/new).

By joining `nuxt-modules`` we can rename your community module under the `@nuxtjs/`` scope and provide a subdomain (e.g. `my-module.nuxtjs.org``) for its documentation.

<!-- ## Module Internals

Maybe just a quick section touching on "how modules work" under the hood, priority, etc. -->

```
---
title: "Custom Routing"
description: "In Nuxt 3, your routing is defined by the structure of your files inside the pages directory. However, since it uses vue-router under the hood, Nuxt offers you several ways to add custom routes in your project."
---
```

Adding custom routes

In Nuxt 3, your routing is defined by the structure of your files inside the [pages directory](/docs/guide/directory-structure/pages). However, since it uses [vue-router](https://router.vuejs.org) under the hood, Nuxt offers you several ways to add custom routes in your project.

Router Config

Using [router options](/docs/guide/recipes/custom-routing#router-options), you can optionally override or extend your routes using a function that accepts the scanned routes and returns customized routes.

If it returns `null` or `undefined`, Nuxt will fall back to the default routes (useful to modify input array).

```
```:ts [app/router.options.ts]
import type { RouterConfig } from '@nuxt/schema'

export default {
 // https://router.vuejs.org/api/interfaces/routeroptions.html#routes
 routes: (_routes) => [
 {
 name: 'home',
 path: '/',
 component: () => import('~pages/home.vue').then(r => r.default || r)
 }
],
} satisfies RouterConfig
```:
```

::note
Nuxt will not augment any new routes you return from the `routes` function with metadata defined in `definePageMeta` of the component you provide. If you want that to happen, you should use the `pages:extend` hook which is [called at build-time](/docs/api/advanced/hooks#nuxt-hooks-build-time).
::

Pages Hook

You can add, change or remove pages from the scanned routes with the `pages:extend` nuxt hook.

For example, to prevent creating routes for any `.ts` files:

```
```:ts [nuxt.config.ts]
import type { NuxtPage } from '@nuxt/schema'

export default defineNuxtConfig({
 hooks: {
 'pages:extend' (pages) {
 // add a route
 pages.push({
 name: 'profile',
 path: '/profile',
 file: '~/extra-pages/profile.vue'
 })

 // remove routes
 function removePagesMatching (pattern: RegExp, pages: NuxtPage[] = []) {
 const pagesToRemove: NuxtPage[] = []
 for (const page of pages) {
 if (page.file && pattern.test(page.file)) {
 pagesToRemove.push(page)
 } else {
 removePagesMatching(pattern, page.children)
 }
 }
 for (const page of pagesToRemove) {
 pages.splice(pages.indexOf(page), 1)
 }
 }
 removePagesMatching(/\.ts$/, pages)
 }
 }
})
```:
```

Nuxt Module

If you plan to add a whole set of pages related with a specific functionality, you might want to use a [Nuxt module](/modules).

The [Nuxt kit](/docs/guide/going-further/kit) provides a few ways [to add routes](/docs/api/kit/pages):

- [`extendPages`](/docs/api/kit/pages#extendpages) (callback: `pages => void`)
- [`extendRouteRules`](/docs/api/kit/pages#extendrouterules) (route: string, rule: `NitroRouteConfig`, options: `ExtendRouteRulesOptions`)

Router Options

On top of customizing options for [`vue-router`](https://router.vuejs.org/api/interfaces/routeroptions.html), Nuxt offers [additional options](/docs/api/nuxt-config#router) to customize the router.

Using `app/router.options`

This is the recommended way to specify [router options](/docs/api/nuxt-config#router).

```
``ts [app/router.options.ts]
import type { RouterConfig } from '@nuxt/schema'

export default {
  satisfies RouterConfig
}
```

It is possible to add more router options files by adding files within the `pages:routerOptions` hook. Later items in the array override earlier ones.

`::alert`
Adding a router options file in this hook will switch on page-based routing, unless `optional` is set, in which case it will only apply when page-based routing is already enabled.
`::`

```
``ts [nuxt.config.ts]
import { createResolver } from '@nuxt/kit'

export default defineNuxtConfig({
  hooks: {
    'pages:routerOptions' ({ files }) {
      const resolver = createResolver(import.meta.url)
      // add a route
      files.push({
        path: resolver.resolve('./runtime/app/router-options'),
        optional: true
      })
    }
  }
})
```

Using `nuxt.config`

Note: Only JSON serializable [options](/docs/api/nuxt-config#router) are configurable:

- `linkActiveClass`
- `linkExactActiveClass`
- `end`
- `sensitive`
- `strict`
- `hashMode`
- `scrollBehaviorType`

```
``js [nuxt.config]
export default defineNuxtConfig({
  router: {
    options: {}
  }
})
```

Hash Mode (SPA)

You can enable hash history in SPA mode using the `hashMode` [config](/docs/api/nuxt-config#router). In this mode, router uses a hash character (#) before the actual URL that is internally passed. When enabled, the **URL is never sent to the server** and **SSR is not supported**.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  ssr: false,
  router: {
    options: {
      hashMode: true
    }
  }
})
```


Scroll Behavior for hash links

You can optionally customize the scroll behavior for hash links. When you set the [config](/docs/api/nuxt-config#router) to be `smooth` and you load a page with a hash link (e.g. `https://example.com/blog/my-article#comments`), you will see that the browser smoothly scrolls to this anchor.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  router: {
    options: {
      scrollBehaviorType: 'smooth'
    }
  }
})
``
```

Custom History (advanced)

You can optionally override history mode using a function that accepts the base URL and returns the history mode. If it returns `null` or `undefined`, Nuxt will fallback to the default history.

```
``ts [app/router.options.ts]
import type { RouterConfig } from '@nuxt/schema'
import { createMemoryHistory } from 'vue-router'

export default {
  // https://router.vuejs.org/api/interfaces/routeroptions.html
  history: base => import.meta.client ? createMemoryHistory(base) : null /* default */
} satisfies RouterConfig
``
```

```
---
navigation.title: 'Vite Plugins'
title: Using Vite Plugins in Nuxt
description: Learn how to integrate Vite plugins into your Nuxt project.
---
```

While Nuxt modules offer extensive functionality, sometimes a specific Vite plugin might meet your needs more directly.

First, we need to install the Vite plugin, for our example, we'll use `@rollup/plugin-yaml`:

::code-group

```
```bash [npm]
npm install @rollup/plugin-yaml
```
```

```
```bash [yarn]
yarn add @rollup/plugin-yaml
```
```

```
```bash [pnpm]
pnpm add @rollup/plugin-yaml
```
```

```
```bash [bun]
bun add @rollup/plugin-yaml
```
```

::

Next, we need to import and add it to our `[`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config)` file:

```
```ts [nuxt.config.ts]
import yaml from '@rollup/plugin-yaml'

export default defineNuxtConfig({
 vite: {
 plugins: [
 yaml()
]
 }
})
```
```

Now we installed and configured our Vite plugin, we can start using YAML files directly in our project.

For example, we can have a `config.yaml` that stores configuration data and import this data in our Nuxt components:

::code-group

```
```yaml [data/hello.yaml]
greeting: "Hello, Nuxt with Vite!"
```
```

```
```vue [components/Hello.vue]
<script setup>
import config from '~/data/hello.yaml'
</script>

<template>
 <h1>{{ config.greeting }}</h1>
</template>
```
```

::

```
---
title: "NuxtApp"
description: "In Nuxt 3, you can access runtime app context within composables, components and plugins."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts
---
```

In Nuxt 3, you can access runtime app context within composables, components and plugins.

```
::read-more{to="https://v2.nuxt.com/docs/internals-glossary/context#the-context" target="_blank"}
In Nuxt 2, this was referred to as Nuxt context.
::
```

Nuxt App Interface

```
::read-more{to="/docs/guide/going-further/internals#the-nuxtapp-interface"}
Jump over the `NuxtApp` interface documentation.
::
```

The Nuxt Context

Many composables and utilities, both built-in and user-made, may require access to the Nuxt instance. This doesn't exist everywhere on your application, because a fresh instance is created on every request.

Currently, the Nuxt context is only accessible in [plugins](/docs/guide/directory-structure/plugins), [Nuxt hooks](/docs/guide/going-further/hooks), [Nuxt middleware](/docs/guide/directory-structure/middleware), and [setup functions](https://vuejs.org/api/composition-api-setup.html) (in pages and components).

If a composable is called without access to the context, you may get an error stating that 'A composable that requires access to the Nuxt instance was called outside of a plugin, Nuxt hook, Nuxt middleware, or Vue setup function.' In that case, you can also explicitly call functions within this context by using [`nuxtApp.runWithContext`](/docs/api/composables/use-nuxt-app#runwithcontext).

Accessing NuxtApp

Within composables, plugins and components you can access `nuxtApp` with [`useNuxtApp()`](/docs/api/composables/use-nuxt-app):

```
``ts [composables/useMyComposable.ts]
export function useMyComposable () {
  const nuxtApp = useNuxtApp()
  // access runtime nuxt app instance
}
```

If your composable does not always need `nuxtApp` or you simply want to check if it is present or not, since [`useNuxtApp`](/docs/api/composables/use-nuxt-app) throws an exception, you can use [`tryUseNuxtApp`](/docs/api/composables/use-nuxt-app#tryusenuxtapp) instead.

Plugins also receive `nuxtApp` as the first argument for convenience.

```
::read-more{to="/docs/guide/directory-structure/plugins"}
```

Providing Helpers

You can provide helpers to be usable across all composables and application. This usually happens within a Nuxt plugin.

```
``ts
const nuxtApp = useNuxtApp()
nuxtApp.provide('hello', (name) => `Hello ${name}!`)

console.log(nuxtApp.$hello('name')) // Prints "Hello name!"
``
```

```
::read-more{to="/docs/guide/directory-structure/plugins#providing-helpers"}
It is possible to inject helpers by returning an object with a provide key in plugins.
::
```

```
::read-more{to="https://v2.nuxt.com/docs/directory-structure/plugins#inject-in-root--context" target="_blank"}
In Nuxt 2 plugins, this was referred to as inject function.
::
```

```

---
title: "defineNuxtRouteMiddleware"
description: "Create named route middleware using defineNuxtRouteMiddleware helper function."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
    size: xs
---

```

Route middleware are stored in the `[`middleware/`](/docs/guide/directory-structure/middleware)` of your Nuxt application (unless `[set otherwise](/docs/api/nuxt-config#middleware)`).

Type

```

```ts
defineNuxtRouteMiddleware(middleware: RouteMiddleware) => RouteMiddleware

interface RouteMiddleware {
 (to: RouteLocationNormalized, from: RouteLocationNormalized): ReturnType<NavigationGuard>
}
```

```

Parameters

`middleware`

- **Type**: `RouteMiddleware`

A function that takes two Vue Router's route location objects as parameters: the next route ``to`` as the first, and the current route ``from`` as the second.

Learn more about available properties of `RouteLocationNormalized` in the **[Vue Router docs]** (<https://router.vuejs.org/api/#RouteLocationNormalized>).

Examples

Showing Error Page

You can use route middleware to throw errors and show helpful error messages:

```

```ts [middleware/error.ts]
export default defineNuxtRouteMiddleware((to) => {
 if (to.params.id === '1') {
 throw createError({ statusCode: 404, statusMessage: 'Page Not Found' })
 }
})
```

```

The above route middleware will redirect a user to the custom error page defined in the `~/error.vue` file, and expose the error message and code passed from the middleware.

Redirection

Use `[`useState`](/docs/api/composables/use-state)` in combination with ``navigateTo`` helper function inside the route middleware to redirect users to different routes based on their authentication status:

```

```ts [middleware/auth.ts]
export default defineNuxtRouteMiddleware((to, from) => {
 const auth = useState('auth')

 if (!auth.value.isAuthenticated) {
 return navigateTo('/login')
 }

 if (to.path !== '/dashboard') {
 return navigateTo('/dashboard')
 }
})
```

```

Both `[navigateTo](/docs/api/utils/navigate-to)` and `[abortNavigation](/docs/api/utils/abort-navigation)` are globally available helper functions that you can use inside ``defineNuxtRouteMiddleware``.

```

---
navigation.title: 'Custom useFetch'
title: Custom useFetch in Nuxt
description: How to create a custom fetcher for calling your external API in Nuxt 3.
---

```

When working with Nuxt, you might be making the frontend and fetching an external API, and you might want to set some default options for fetching from your API.

The `$fetch` (</docs/api/utis/dollarfetch>) utility function (used by the `useFetch` (</docs/api/composables/use-fetch>) composable) is intentionally not globally configurable. This is important so that fetching behavior throughout your application remains consistent, and other integrations (like modules) can rely on the behavior of core utilities like `$fetch`.

However, Nuxt provides a way to create a custom fetcher for your API (or multiple fetchers if you have multiple APIs to call).

Custom `$fetch`

Let's create a custom `$fetch` instance with a [\[Nuxt plugin\]](/docs/guide/directory-structure/plugins).

```

::note
`$fetch` is a configured instance of ofetch (https://github.com/unjs/ofetch) which supports adding the base URL of your Nuxt server as well as direct function calls during SSR (avoiding HTTP roundtrips).
::

```

Let's pretend here that:

- The main API is <https://api.nuxt.com>
- We are storing the JWT token in a session with [\[nuxt-auth-utils\]](https://github.com/atinux/nuxt-auth-utils)
- If the API responds with a `401` status code, we redirect the user to the `/login` page

```

`ts [plugins/api.ts]
export default defineNuxtPlugin((nuxtApp) => {
  const { session } = useUserSession()

  const api = $fetch.create({
    baseURL: 'https://api.nuxt.com',
    onRequest({ request, options, error }) {
      if (session.value?.token) {
        const headers = options.headers || {}
        if (Array.isArray(headers)) {
          headers.push(['Authorization', `Bearer ${session.value?.token}`])
        } else if (headers instanceof Headers) {
          headers.set('Authorization', `Bearer ${session.value?.token}`)
        } else {
          headers.Authorization = `Bearer ${session.value?.token}`
        }
      }
    },
    async onResponseError({ response }) {
      if (response.status === 401) {
        await nuxtApp.runWithContext(() => navigateTo('/login'))
      }
    }
  })

  // Expose to useNuxtApp().$api
  return {
    provide: {
      api
    }
  }
})
`

```

With this Nuxt plugin, `$api` is exposed from `useNuxtApp()` to make API calls directly from the Vue components:

```

`vue [app.vue]
<script setup>
const { $api } = useNuxtApp()
const { data: modules } = await useAsyncData('modules', () => $api('/modules'))
</script>
`

```

```

::callout
Wrapping with useAsyncData (/docs/api/composables/use-async-data) **avoid double data fetching when doing server-side rendering** (server & client on hydration).
::

```

Custom `useFetch`/`useAsyncData`

Now that `$api` has the logic we want, let's create a `useAPI` composable to replace the usage of `useAsyncData` + `$api`:

```

```ts [composables/useAPI.ts]
import type { UseFetchOptions } from 'nuxt/app'

export function useAPI<T>(
 url: string | (() => string),
 options?: UseFetchOptions<T>,
) {
 return useFetch(url, {
 ...options,
 $fetch: useNuxtApp().$api
 })
}
```

```

Let's use the new composable and have a nice and clean component:

```

```vue [app.vue]
<script setup>
const { data: modules } = await useAPI('/modules')
</script>
```

```

::note

This example demonstrates how to use a custom `useFetch`, but the same structure is identical for a custom `useAsyncData`.

::

::callout{icon="i-simple-icons-youtube" color="red" to="https://www.youtube.com/watch?v=jXH8Tr-exhI"} Watch a video about custom `\$fetch` and Repository Pattern in Nuxt.

::

::note

We are currently discussing to find a cleaner way to let you create a custom fetcher, see

<https://github.com/nuxt/nuxt/issues/14736>.

::

```
---
title: 'defineRouteRules'
description: 'Define route rules for hybrid rendering at the page level.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/pages/runtime/composables.ts
    size: xs
---
```

```
::read-more{to="/docs/guide/going-further/experimental-features#inlinerrouterules" icon="i-ph-star-duotone"}
This feature is experimental and in order to use it you must enable the `experimental.inlineRouteRules` option in your
`nuxt.config`.
::
```

Usage

```
``vue [pages/index.vue]
<script setup lang="ts">
defineRouteRules({
  prerender: true
})
</script>

<template>
  <h1>Hello world!</h1>
</template>
``
```

Will be translated to:

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  routeRules: {
    '/': { prerender: true }
  }
})
``
```

::note

When running [`nuxt build`](/docs/api/commands/build), the home page will be pre-rendered in `.output/public/index.html` and statically served.

Notes

- A rule defined in `~/pages/foo/bar.vue` will be applied to `/foo/bar` requests.
- A rule in `~/pages/foo/[id].vue` will be applied to `/foo/**` requests.

For more control, such as if you are using a custom `path` or `alias` set in the page's [`definePageMeta`](/docs/api/utils/define-page-meta), you should set `routeRules` directly within your `nuxt.config`.

```
::read-more{to="/docs/guide/concepts/rendering#hybrid-rendering" icon="i-ph-medal-duotone"}
Read more about the `routeRules`.
::
```

```
---
title: 'Routing'
description: Nuxt file-system routing creates a route for every file in the pages/ directory.
navigation.icon: i-ph-signpost-duotone
---
```

One core feature of Nuxt is the file system router. Every Vue file inside the [`pages/`](/docs/guide/directory-structure/pages) directory creates a corresponding URL (or route) that displays the contents of the file. By using dynamic imports for each page, Nuxt leverages code-splitting to ship the minimum amount of JavaScript for the requested route.

Pages

Nuxt routing is based on [vue-router](https://router.vuejs.org) and generates the routes from every component created in the [`pages/` directory](/docs/guide/directory-structure/pages), based on their filename.

This file system routing uses naming conventions to create dynamic and nested routes:

```
::code-group
```

```
```bash [Directory Structure]
| pages/
---| about.vue
---| index.vue
---| posts/
-----| [id].vue
```
```

```
```json [Generated Router File]
{
 "routes": [
 {
 "path": "/about",
 "component": "pages/about.vue"
 },
 {
 "path": "/",
 "component": "pages/index.vue"
 },
 {
 "path": "/posts/:id",
 "component": "pages/posts/[id].vue"
 }
]
}
```

```
::
:read-more{to="/docs/guide/directory-structure/pages"}
```

## ## Navigation

The [`<NuxtLink>`](/docs/api/components/nuxt-link) component links pages between them. It renders an `<a>` tag with the `href` attribute set to the route of the page. Once the application is hydrated, page transitions are performed in JavaScript by updating the browser URL. This prevents full-page refreshes and allows for animated transitions.

When a [`<NuxtLink>`](/docs/api/components/nuxt-link) enters the viewport on the client side, Nuxt will automatically prefetch components and payload (generated pages) of the linked pages ahead of time, resulting in faster navigation.

```
```vue [pages/app.vue]
<template>
  <header>
    <nav>
      <ul>
        <li><NuxtLink to="/about">About</NuxtLink></li>
        <li><NuxtLink to="/posts/1">Post 1</NuxtLink></li>
        <li><NuxtLink to="/posts/2">Post 2</NuxtLink></li>
      </ul>
    </nav>
  </header>
</template>
```
```

```
:read-more{to="/docs/api/components/nuxt-link"}
```

## ## Route Parameters

The [`useRoute()`](/docs/api/composables/use-route) composable can be used in a `<script setup>` block or a `setup()` method of a Vue component to access the current route details.

```
```vue twoslash [pages/posts/[id\].vue]
<script setup lang="ts">
const route = useRoute()
```



```
// When accessing /posts/1, route.params.id will be 1
console.log(route.params.id)
</script>
````
```

[:read-more{to="/docs/api/composables/use-route"}](/docs/api/composables/use-route)

## ## Route Middleware

Nuxt provides a customizable route middleware framework you can use throughout your application, ideal for extracting code that you want to run before navigating to a particular route.

```
::note
Route middleware runs within the Vue part of your Nuxt app. Despite the similar name, they are completely different from server middleware, which are run in the Nitro server part of your app.
::
```

There are three kinds of route middleware:

1. Anonymous (or inline) route middleware, which are defined directly in the pages where they are used.
2. Named route middleware, which are placed in the [`middleware/`](/docs/guide/directory-structure/middleware) directory and will be automatically loaded via asynchronous import when used on a page. (\*\*Note\*\*: The route middleware name is normalized to kebab-case, so `someMiddleware` becomes `some-middleware`.)
3. Global route middleware, which are placed in the [`middleware/` directory](/docs/guide/directory-structure/middleware) (with a `.global` suffix) and will be automatically run on every route change.

Example of an `auth` middleware protecting the `/dashboard` page:

```
::code-group
```

```
````ts twoslash [middleware/auth.ts]
function isAuthenticated(): boolean { return false }
// ---cut---
export default defineNuxtRouteMiddleware((to, from) => {
  // isAuthenticated() is an example method verifying if a user is authenticated
  if (isAuthenticated() === false) {
    return navigateTo('/login')
  }
})
````
```

```
````vue twoslash [pages/dashboard.vue]
<script setup lang="ts">
definePageMeta({
  middleware: 'auth'
})
</script>
```

```
<template>
  <h1>Welcome to your dashboard</h1>
</template>
````
```

```
::

:read-more{to="/docs/guide/directory-structure/middleware"}
```

## ## Route Validation

Nuxt offers route validation via the `validate` property in [`definePageMeta()`](/docs/api/utils/define-page-meta) in each page you wish to validate.

The `validate` property accepts the `route` as an argument. You can return a boolean value to determine whether or not this is a valid route to be rendered with this page. If you return `false`, and another match can't be found, this will cause a 404 error. You can also directly return an object with `statusCode`/`statusMessage` to respond immediately with an error (other matches will not be checked).

If you have a more complex use case, then you can use anonymous route middleware instead.

```
````vue twoslash [pages/posts/[id\].vue]
<script setup lang="ts">
definePageMeta({
  validate: async (route) => {
    // Check if the id is made up of digits
    return typeof route.params.id === 'string' && /\d+$/i.test(route.params.id)
  }
})
</script>
````
```

[:read-more{to="/docs/api/utils/define-page-meta"}](/docs/api/utils/define-page-meta)

```

title: '<NuxtWelcome>'
description: The <NuxtWelcome> component greets users in new projects made from the starter template.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/assets/blob/main/packages/templates/templates/welcome/index.html
 size: xs

```

It includes links to the Nuxt documentation, source code, and social media accounts.

```
```vue [app.vue]
<template>
  <NuxtWelcome />
</template>
```
```

```
::read-more{to="https://templates.ui.nuxtjs.org/templates/welcome" target="_blank"}
Preview the `<NuxtWelcome />` component.
::
```

```
::tip
This component is part of [nuxt/assets](https://github.com/nuxt/assets).
::
```

```

title: 'definePageMeta'
description: 'Define metadata for your page components.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/pages/runtime/composables.ts
 size: xs

```

`definePageMeta` is a compiler macro that you can use to set metadata for your **page** components located in the [pages/] (/docs/guide/directory-structure/pages) directory (unless [set otherwise](/docs/api/nuxt-config#pages)). This way you can set custom metadata for each static or dynamic route of your Nuxt application.

```

```vue [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
  layout: 'default'
})
</script>
```

```

```
:read-more{to="/docs/guide/directory-structure/pages/#page-metadata"}
```

## ## Type

```

```ts
definePageMeta(meta: PageMeta) => void

interface PageMeta {
  validate?: (route: RouteLocationNormalized) => boolean | Promise<boolean> | Partial<NuxtError> |
Promise<Partial<NuxtError>>
  redirect?: RouteRecordRedirectOption
  name?: string
  path?: string
  alias?: string | string[]
  pageTransition?: boolean | TransitionProps
  layoutTransition?: boolean | TransitionProps
  viewTransition?: boolean | 'always'
  key?: false | string | ((route: RouteLocationNormalizedLoaded) => string)
  keepalive?: boolean | KeepAliveProps
  layout?: false | LayoutKey | Ref<LayoutKey> | ComputedRef<LayoutKey>
  middleware?: MiddlewareKey | NavigationGuard | Array<MiddlewareKey | NavigationGuard>
  scrollToTop?: boolean | ((to: RouteLocationNormalizedLoaded, from: RouteLocationNormalizedLoaded) => boolean)
  [key: string]: unknown
}
```

```

## ## Parameters

### ### `meta`

- **Type**: `PageMeta`

An object accepting the following page metadata:

**name**

- **Type**: `string`

You may define a name for this page's route. By default, name is generated based on path inside the [pages/] (/docs/guide/directory-structure/pages) directory.

**path**

- **Type**: `string`

You may define a [custom regular expression](#using-a-custom-regular-expression) if you have a more complex pattern than can be expressed with the file name.

**alias**

- **Type**: `string | string[]`

Aliases for the record. Allows defining extra paths that will behave like a copy of the record. Allows having paths shorthands like `users/:id` and `/u/:id`. All `alias` and `path` values must share the same params.

**keepalive**

- **Type**: `boolean` | [KeepAliveProps](https://vuejs.org/api/built-in-components.html#keepalive)

Set to `true` when you want to preserve page state across route changes or use the [KeepAliveProps](https://vuejs.org/api/built-in-components.html#keepalive) for a fine-grained control.

**\*\*`key`\*\***

- **\*\*Type\*\***: ``false` | `string` | `((route: RouteLocationNormalizedLoaded) => string)``

Set ``key`` value when you need more control over when the `<NuxtPage>` component is re-rendered.

**\*\*`layout`\*\***

- **\*\*Type\*\***: ``false` | `LayoutKey` | `Ref<LayoutKey>` | `ComputedRef<LayoutKey>``

Set a static or dynamic name of the layout for each route. This can be set to ``false`` in case the default layout needs to be disabled.

**\*\*`layoutTransition`\*\***

- **\*\*Type\*\***: ``boolean` | [TransitionProps](https://vuejs.org/api/built-in-components.html#transition)`

Set name of the transition to apply for current layout. You can also set this value to ``false`` to disable the layout transition.

**\*\*`middleware`\*\***

- **\*\*Type\*\***: ``MiddlewareKey` | [NavigationGuard]`

(<https://router.vuejs.org/api/interfaces/NavigationGuard.html#navigationguard>) | ``Array<MiddlewareKey | NavigationGuard>``

Define anonymous or named middleware directly within ``definePageMeta``. Learn more about [route middleware] (/docs/guide/directory-structure/middleware).

**\*\*`pageTransition`\*\***

- **\*\*Type\*\***: ``boolean` | [TransitionProps](https://vuejs.org/api/built-in-components.html#transition)`

Set name of the transition to apply for current page. You can also set this value to ``false`` to disable the page transition.

**\*\*`viewTransition`\*\***

- **\*\*Type\*\***: ``boolean` | 'always'``

**\*\*Experimental feature, only available when [enabled in your nuxt.config file](/docs/getting-started/transitions#view-transitions-api-experimental)\*\*</br>**

Enable/disable View Transitions for the current page.

If set to true, Nuxt will not apply the transition if the users browser matches ``prefers-reduced-motion: reduce`` (recommended). If set to ``always``, Nuxt will always apply the transition.

**\*\*`redirect`\*\***

- **\*\*Type\*\***: `[RouteRecordRedirectOption](https://router.vuejs.org/guide/essentials/redirect-and-alias.html#redirect-and-alias)`

Where to redirect if the route is directly matched. The redirection happens before any navigation guard and triggers a new navigation with the new target location.

**\*\*`validate`\*\***

- **\*\*Type\*\***: ``(route: RouteLocationNormalized) => boolean | Promise<boolean> | Partial<NuxtError> | Promise<Partial<NuxtError>>``

Validate whether a given route can validly be rendered with this page. Return true if it is valid, or false if not. If another match can't be found, this will mean a 404. You can also directly return an object with ``statusCode``/``statusMessage`` to respond immediately with an error (other matches will not be checked).

**\*\*`scrollToTop`\*\***

- **\*\*Type\*\***: ``boolean` | (to: RouteLocationNormalized, from: RouteLocationNormalized) => boolean``

Tell Nuxt to scroll to the top before rendering the page or not. If you want to overwrite the default scroll behavior of Nuxt, you can do so in ``~/app/router.options.ts`` (see [custom routing](/docs/guide/recipes/custom-routing#using-approuteroptions)) for more info.

**\*\*`[key: string]`\*\***

- **\*\*Type\*\***: ``any``

Apart from the above properties, you can also set **\*\*custom\*\*** metadata. You may wish to do so in a type-safe way by [augmenting the type of the ``meta`` object](/docs/guide/directory-structure/pages/#typing-custom-metadata).

## ## Examples

### ### Basic Usage

The example below demonstrates:

- how `key` can be a function that returns a value;
- how `keepalive` property makes sure that the `` component is not cached when switching between multiple components;
- adding `pageType` as a custom property:

```
```vue [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
  key: (route) => route.fullPath,

  keepalive: {
    exclude: ['modal']
  },

  pageType: 'Checkout'
})
</script>
```
```

### ### Defining Middleware

The example below shows how the middleware can be defined using a `function` directly within the `definePageMeta` or set as a `string` that matches the middleware file name located in the `middleware/` directory:

```
```vue [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
  // define middleware as a function
  middleware: [
    function (to, from) {
      const auth = useState('auth')

      if (!auth.value.authenticated) {
        return navigateTo('/login')
      }

      if (to.path !== '/checkout') {
        return navigateTo('/checkout')
      }
    }
  ],

  // ... or a string
  middleware: 'auth'

  // ... or multiple strings
  middleware: ['auth', 'another-named-middleware']
})
</script>
```
```

### ### Using a Custom Regular Expression

A custom regular expression is a good way to resolve conflicts between overlapping routes, for instance:

The two routes `/test-category` and `/1234-post` match both `[postId]-[postSlug].vue` and `[categorySlug].vue` page routes.

To make sure that we are only matching digits (`\d+`) for `postId` in the `[postId]-[postSlug]` route, we can add the following to the `[postId]-[postSlug].vue` page template:

```
```vue [pages/[postId\\]-[postSlug\\].vue]
<script setup lang="ts">
definePageMeta({
  path: '/:postId(\\d+)-:postSlug'
})
</script>
```
```

For more examples see [Vue Router's Matching Syntax](https://router.vuejs.org/guide/essentials/route-matching-syntax.html).

### ### Defining Layout

You can define the layout that matches the layout's file name located (by default) in the `[layouts/` directory] (/docs/guide/directory-structure/layouts)`. You can also disable the layout by setting the `layout` to `false`:

```
```vue [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
  // set custom layout
  layout: 'admin'

  // ... or disable a default layout
  layout: false
})
</script>
```
```

```
})
</script>
```
```

```
---
title: 'useLazyFetch'
description: This wrapper around useFetch triggers navigation immediately.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/fetch.ts
    size: xs
---
```

Description

By default, [`useFetch`](/docs/api/composables/use-fetch) blocks navigation until its async handler is resolved. `useLazyFetch` provides a wrapper around [`useFetch`](/docs/api/composables/use-fetch) that triggers navigation before the handler is resolved by setting the `lazy` option to `true`.

```
:::note
`useLazyFetch` has the same signature as [useFetch](/docs/api/composables/use-fetch).
:::
```

```
:read-more{to="/docs/api/composables/use-fetch"}
```

Example

```
``vue [pages/index.vue]
<script setup lang="ts">
/* Navigation will occur before fetching is complete.
 * Handle 'pending' and 'error' states directly within your component's template
 */
const { status, data: posts } = await useLazyFetch('/api/posts')
watch(posts, (newPosts) => {
  // Because posts might start out null, you won't have access
  // to its contents immediately, but you can watch it.
})
</script>
```

```
<template>
  <div v-if="status === 'pending'">
    Loading ...
  </div>
  <div v-else>
    <div v-for="post in posts">
      <!-- do something -->
    </div>
  </div>
</template>
``
```

```
:::note
`useLazyFetch` is a reserved function name transformed by the compiler, so you should not name your own function
`useLazyFetch`.
:::
```

```
:read-more{to="/docs/getting-started/data-fetching"}
```

```

---
title: "plugins"
description: "Nuxt has a plugins system to use Vue plugins and more at the creation of your Vue application."
head.title: "plugins/"
navigation.icon: i-ph-folder-duotone
---

```

Nuxt automatically reads the files in the `plugins/` directory and loads them at the creation of the Vue application.

```

::note
All plugins inside are auto-registered, you don't need to add them to your `nuxt.config` separately.
::

```

```

::note
You can use `.server` or `.client` suffix in the file name to load a plugin only on the server or client side.
::

```

Registered Plugins

Only files at the top level of the directory (or index files within any subdirectories) will be auto-registered as plugins.

```

```bash [Directory structure]
-| plugins/
---| foo.ts // scanned
---| bar/
----| baz.ts // not scanned
----| foz.vue // not scanned
----| index.ts // currently scanned but deprecated
```

```

Only `foo.ts` and `bar/index.ts` would be registered.

To add plugins in subdirectories, you can use the [`plugins`](/docs/api/nuxt-config#plugins-1) option in `nuxt.config.ts`:

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 plugins: [
 '~/plugins/bar/baz',
 '~/plugins/bar/foz'
]
})
```

```

Creating Plugins

The only argument passed to a plugin is [`nuxtApp`](/docs/api/composables/use-nuxt-app).

```

```ts twoslash [plugins/hello.ts]
export default defineNuxtPlugin(nuxtApp => {
 // Doing something with nuxtApp
})
```

```

Object Syntax Plugins

It is also possible to define a plugin using an object syntax, for more advanced use cases. For example:

```

```ts twoslash [plugins/hello.ts]
export default defineNuxtPlugin({
 name: 'my-plugin',
 enforce: 'pre', // or 'post'
 async setup (nuxtApp) {
 // this is the equivalent of a normal functional plugin
 },
 hooks: {
 // You can directly register Nuxt app runtime hooks here
 'app:created'() {
 const nuxtApp = useNuxtApp()
 // do something in the hook
 }
 },
 env: {
 // Set this value to `false` if you don't want the plugin to run when rendering server-only or island components.
 islands: true
 }
})
```

```

```

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=2aXZyXB1QGQ" target="_blank"}
Watch a video from Alexander Lichter about the Object Syntax for Nuxt plugins.
::

```

```

::note

```


If you are using the object-syntax, the properties are statically analyzed to produce a more optimized build. So you should not define them at runtime. `::br`

For example, setting ``enforce: import.meta.server ? 'pre' : 'post'`` would defeat any future optimization Nuxt is able to do for your plugins.

Nuxt does statically pre-load any hook listeners when using object-syntax, allowing you to define hooks without needing to worry about order of plugin registration.

`::`

Registration Order

You can control the order in which plugins are registered by prefixing with 'alphabetical' numbering to the file names.

```
``bash [Directory structure]
plugins/
| - 01.myPlugin.ts
| - 02.myOtherPlugin.ts
``
```

In this example, ``02.myOtherPlugin.ts`` will be able to access anything that was injected by ``01.myPlugin.ts``.

This is useful in situations where you have a plugin that depends on another plugin.

`::note`

In case you're new to 'alphabetical' numbering, remember that filenames are sorted as strings, not as numeric values. For example, ``10.myPlugin.ts`` would come before ``2.myOtherPlugin.ts``. This is why the example prefixes single digit numbers with ``0``.

`::`

Loading Strategy

Parallel Plugins

By default, Nuxt loads plugins sequentially. You can define a plugin as ``parallel`` so Nuxt won't wait the end of the plugin's execution before loading the next plugin.

```
``ts twoslash [plugins/my-plugin.ts]
export default defineNuxtPlugin({
  name: 'my-plugin',
  parallel: true,
  async setup (nuxtApp) {
    // the next plugin will be executed immediately
  }
})
``
```

Plugins With Dependencies

If a plugin needs to wait for another plugin before it runs, you can add the plugin's name to the ``dependsOn`` array.

```
``ts twoslash [plugins/depending-on-my-plugin.ts]
export default defineNuxtPlugin({
  name: 'depends-on-my-plugin',
  dependsOn: ['my-plugin'],
  async setup (nuxtApp) {
    // this plugin will wait for the end of `my-plugin`'s execution before it runs
  }
})
``
```

Using Composables

You can use `[composables] (/docs/guide/directory-structure/composables)` as well as `[utils] (/docs/guide/directory-structure/utils)` within Nuxt plugins:

```
``ts [plugins/hello.ts]
export default defineNuxtPlugin((nuxtApp) => {
  const foo = useFoo()
})
``
```

However, keep in mind there are some limitations and differences:

`::important`

`**If a composable depends on another plugin registered later, it might not work.**` `::br`

Plugins are called in order sequentially and before everything else. You might use a composable that depends on another plugin which has not been called yet.

`::`

`::important`

`**If a composable depends on the Vue.js lifecycle, it won't work.**` `::br`

Normally, Vue.js composables are bound to the current component instance while plugins are only bound to `[`nuxtApp`]`

(/docs/api/composables/use-nuxt-app) instance.
::

Providing Helpers

If you would like to provide a helper on the [`NuxtApp`](/docs/api/composables/use-nuxt-app) instance, return it from the plugin under a `provide` key.

```
::code-group
```ts twoslash [plugins/hello.ts]
export default defineNuxtPlugin(() => {
 return {
 provide: {
 hello: (msg: string) => `Hello ${msg}!`
 }
 }
})
```
```ts twoslash [plugins/hello-object-syntax.ts]
export default defineNuxtPlugin({
 name: 'hello',
 setup () {
 return {
 provide: {
 hello: (msg: string) => `Hello ${msg}!`
 }
 }
 }
})
```
::
```

You can then use the helper in your components:

```
```vue [components/Hello.vue]
<script setup lang="ts">
// alternatively, you can also use it here
const { $hello } = useNuxtApp()
</script>

<template>
 <div>
 {{ $hello('world') }}
 </div>
</template>
```
```

::important

Note that we highly recommend using [`composables`](/docs/guide/directory-structure/composables) instead of providing helpers to avoid polluting the global namespace and keep your main bundle entry small.

::

::warning

If your plugin provides a `ref` or `computed`, it will not be unwrapped in a component `<template>`.
This is due to how Vue works with refs that aren't top-level to the template. You can read more about it [in the Vue documentation](https://vuejs.org/guide/essentials/reactivity-fundamentals.html#caveat-when-unwrapping-in-templates).
::

Typing Plugins

If you return your helpers from the plugin, they will be typed automatically; you'll find them typed for the return of `useNuxtApp()` and within your templates.

::note

If you need to use a provided helper `_within_` another plugin, you can call [`useNuxtApp()`](/docs/api/composables/use-nuxt-app) to get the typed version. But in general, this should be avoided unless you are certain of the plugins' order.
::

For advanced use-cases, you can declare the type of injected properties like this:

```
```ts [index.d.ts]
declare module '#app' {
 interface NuxtApp {
 $hello (msg: string): string
 }
}

declare module 'vue' {
 interface ComponentCustomProperties {
 $hello (msg: string): string
 }
}
```

```
export {}
````
```

:::note

If you are using WebStorm, you may need to augment `@vue/runtime-core` until [this issue] (<https://youtrack.jetbrains.com/issue/WEB-59818/VUE-TypeScript-WS-PS-does-not-correctly-display-type-of-globally-injected-properties>) is resolved.

::

Vue Plugins

If you want to use Vue plugins, like [vue-gtag](<https://github.com/MatteoGabriele/vue-gtag>) to add Google Analytics tags, you can use a Nuxt plugin to do so.

First, install the Vue plugin dependency:

```
:::code-group
```bash [yarn]
yarn add --dev vue-gtag-next
```
```bash [npm]
npm install --save-dev vue-gtag-next
```
```bash [pnpm]
pnpm add -D vue-gtag-next
```
```bash [bun]
bun add -D vue-gtag-next
```
::
```

Then create a plugin file:

```
```ts [plugins/vue-gtag.client.ts]
import VueGtag, { trackRouter } from 'vue-gtag-next'

export default defineNuxtPlugin((nuxtApp) => {
 nuxtApp.vueApp.use(VueGtag, {
 property: {
 id: 'GA_MEASUREMENT_ID'
 }
 })
 trackRouter(useRouter())
})
```
```

Vue Directives

Similarly, you can register a custom Vue directive in a plugin.

```
```ts twoslash [plugins/my-directive.ts]
export default defineNuxtPlugin((nuxtApp) => {
 nuxtApp.vueApp.directive('focus', {
 mounted (el) {
 el.focus()
 },
 getSSRProps (binding, vnode) {
 // you can provide SSR-specific props here
 return {}
 }
 })
})
```
```

:::warning

If you register a Vue directive, you must register it on both client and server side unless you are only using it when rendering one side. If the directive only makes sense from a client side, you can always move it to `~/plugins/my-directive.client.ts` and provide a 'stub' directive for the server in `~/plugins/my-directive.server.ts`.

::

:read-more{icon="i-simple-icons-vuedotjs" title="Custom Directives on Vue Docs" to="https://vuejs.org/guide/reusability/custom-directives.html" target="_blank"}

```
---
title: SEO and Meta
description: Improve your Nuxt app's SEO with powerful head config, composables and components.
navigation.icon: i-ph-file-search-duotone
---
```

Defaults

Out-of-the-box, Nuxt provides sensible defaults, which you can override if needed.

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 app: {
 head: {
 charset: 'utf-8',
 viewport: 'width=device-width, initial-scale=1',
 }
 }
})
```
```

Providing an `[`app.head`](/docs/api/nuxt-config#head)` property in your `[`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config)` allows you to customize the head for your entire app.

::important

This method does not allow you to provide reactive data. We recommend to use ``useHead()`` in ``app.vue``.

::

Shortcuts are available to make configuration easier: ``charset`` and ``viewport``. You can also provide any of the keys listed below in `[Types](#types)`.

`useHead`

The `[`useHead`](/docs/api/composables/use-head)` composable function allows you to manage your head tags programmatically and reactively, powered by `[Unhead](https://unhead.unjs.io)`.

As with all composables, it can only be used with a components ``setup`` and lifecycle hooks.

```
```:vue twoslash [app.vue]
<script setup lang="ts">
useHead({
 title: 'My App',
 meta: [
 { name: 'description', content: 'My amazing site.' }
],
 bodyAttrs: {
 class: 'test'
 },
 script: [{ innerHTML: 'console.log(`Hello world`)' }]
})
</script>
```
```

We recommend to take a look at the `[`useHead`](/docs/api/composables/use-head)` and `[`useHeadSafe`](/docs/api/composables/use-head-safe)` composables.

`useSeoMeta`

The `[`useSeoMeta`](/docs/api/composables/use-seo-meta)` composable lets you define your site's SEO meta tags as a flat object with full TypeScript support.

This helps you avoid typos and common mistakes, such as using ``name`` instead of ``property``.

```
```:vue twoslash [app.vue]
<script setup lang="ts">
useSeoMeta({
 title: 'My Amazing Site',
 ogTitle: 'My Amazing Site',
 description: 'This is my amazing site, let me tell you all about it.',
 ogDescription: 'This is my amazing site, let me tell you all about it.',
 ogImage: 'https://example.com/image.png',
 twitterCard: 'summary_large_image',
})
</script>
```
```

:read-more{to="/docs/api/composables/use-seo-meta"}

Components

Nuxt provides `<Title>`, `<Base>`, `<NoScript>`, `<Style>`, `<Meta>`, `<Link>`, `<Body>`, `<Html>` and `<Head>` components so that you can interact directly with your metadata within your component's template.

Because these component names match native HTML elements, they must be capitalized in the template.

`<Head>` and `<Body>` can accept nested meta tags (for aesthetic reasons) but this does not affect `_where_` the nested meta tags are rendered in the final HTML.

```
<!-- @case-police-ignore html -->

```vue [app.vue]
<script setup lang="ts">
const title = ref('Hello World')
</script>

<template>
 <div>
 <Head>
 <Title>{{ title }}</Title>
 <Meta name="description" :content="title" />
 <Style type="text/css" children="body { background-color: green; }" ></Style>
 </Head>

 <h1>{{ title }}</h1>
 </div>
</template>
```
```

Types

Below are the non-reactive types used for `[`useHead`]`(/docs/api/composables/use-head), `[`app.head`]`(/docs/api/nuxt-config#head) and components.

```
```ts
interface MetaObject {
 title?: string
 titleTemplate?: string | ((title?: string) => string)
 templateParams?: Record<string, string | Record<string, string>>
 base?: Base
 link?: Link[]
 meta?: Meta[]
 style?: Style[]
 script?: Script[]
 noscript?: Noscript[];
 htmlAttrs?: HtmlAttributes;
 bodyAttrs?: BodyAttributes;
}
```
```

See `[@unhead/schema]`(<https://github.com/unjs/unhead/blob/main/packages/schema/src/schema.ts>) for more detailed types.

Features

Reactivity

Reactivity is supported on all properties, by providing a computed value, a getter, or a reactive object.

::code-group

```
```vue twoslash [useHead]
<script setup lang="ts">
const description = ref('My amazing site.')

useHead({
 meta: [
 { name: 'description', content: description }
],
})
</script>
```

```vue twoslash [useSeoMeta]
<script setup lang="ts">
const description = ref('My amazing site.')

useSeoMeta({
 description
})
</script>
```

```vue [Components]
<script setup lang="ts">
const description = ref('My amazing site.')
</script>
```

```

<template>
 <div>
 <Meta name="description" :content="description" />
 </div>
</template>
````

```

::

Title Template

You can use the `titleTemplate` option to provide a dynamic template for customizing the title of your site. For example, you could add the name of your site to the title of every page.

The `titleTemplate` can either be a string, where `%s` is replaced with the title, or a function.

If you want to use a function (for full control), then this cannot be set in your `nuxt.config`. It is recommended instead to set it within your `app.vue` file where it will apply to all pages on your site:

::code-group

```

````vue twoslash [useHead]
<script setup lang="ts">
useHead({
 titleTemplate: (titleChunk) => {
 return titleChunk ? `${titleChunk} - Site Title` : 'Site Title';
 }
})
</script>
````

```

::

Now, if you set the title to `My Page` with `[`useHead`](/docs/api/composables/use-head)` on another page of your site, the title would appear as `My Page - Site Title` in the browser tab. You could also pass `null` to default to `Site Title`.

Body Tags

You can use the `tagPosition: 'bodyClose'` option on applicable tags to append them to the end of the `` tag.

For example:

```

````vue twoslash
<script setup lang="ts">
useHead({
 script: [
 {
 src: 'https://third-party-script.com',
 // valid options are: 'head' | 'bodyClose' | 'bodyOpen'
 tagPosition: 'bodyClose'
 }
]
})
</script>
````

```

Examples

With `definePageMeta`

Within your `[`pages/` directory](/docs/guide/directory-structure/pages)`, you can use `definePageMeta` along with `[`useHead`](/docs/api/composables/use-head)` to set metadata based on the current route.

For example, you can first set the current page title (this is extracted at build time via a macro, so it can't be set dynamically):

```

````vue twoslash [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
 title: 'Some Page'
})
</script>
````

```

And then in your layout file, you might use the route's metadata you have previously set:

```

````vue twoslash [layouts/default.vue]
<script setup lang="ts">
const route = useRoute()

useHead({
 meta: [{ property: 'og:title', content: `App Name - ${route.meta.title}` }]
})

```

```

})
</script>
```

```

```

:link-example{to="/docs/examples/features/meta-tags"}

```

```

:read-more{to="/docs/guide/directory-structure/pages/#page-metadata"}

```

Dynamic Title

In the example below, `titleTemplate` is set either as a string with the `%s` placeholder or as a `function`, which allows greater flexibility in setting the page title dynamically for each route of your Nuxt app:

```

```vue twoslash [app.vue]
<script setup lang="ts">
useHead({
 // as a string,
 // where `%s` is replaced with the title
 titleTemplate: '%s - Site Title',
})
</script>
```

```

```

```vue twoslash [app.vue]
<script setup lang="ts">
useHead({
 // or as a function
 titleTemplate: (productCategory) => {
 return productCategory
 ? `${productCategory} - Site Title`
 : 'Site Title'
 }
})
</script>
```

```

`nuxt.config` is also used as an alternative way of setting the page title. However, `nuxt.config` does not allow the page title to be dynamic. Therefore, it is recommended to use `titleTemplate` in the `app.vue` file to add a dynamic title, which is then applied to all routes of your Nuxt app.

External CSS

The example below shows how you might enable Google Fonts using either the `link` property of the [`useHead`](/docs/api/composables/use-head) composable or using the `` component:

```

::code-group

```

```

```vue twoslash [useHead]
<script setup lang="ts">
useHead({
 link: [
 {
 rel: 'preconnect',
 href: 'https://fonts.googleapis.com'
 },
 {
 rel: 'stylesheet',
 href: 'https://fonts.googleapis.com/css2?family=Roboto&display=swap',
 crossorigin: ''
 }
]
})
</script>
```

```

```

```vue [Components]
<template>
 <div>
 <Link rel="preconnect" href="https://fonts.googleapis.com" />
 <Link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" crossorigin="" />
 </div>
</template>
```

```

```

::

```

```
---
title: "pages"
description: "Nuxt provides file-based routing to create routes within your web application."
head.title: "pages/"
navigation.icon: i-ph-folder-duotone
---
```

```
::note
To reduce your application's bundle size, this directory is optional, meaning that [vue-router]
(https://router.vuejs.org) won't be included if you only use [app.vue](/docs/guide/directory-structure/app). To force the
pages system, set pages: true in nuxt.config or have a [app/router.options.ts](/docs/guide/recipes/custom-
routing#using-approuteroptions).
::
```

Usage

Pages are Vue components and can have any [valid extension](/docs/api/configuration/nuxt-config#extensions) that Nuxt supports (by default `.vue`, `.js`, `.jsx`, `.mjs`, `.ts` or `.tsx`).

Nuxt will automatically create a route for every page in your `~/pages/` directory.

::code-group

```
```vue [pages/index.vue]
<template>
 <h1>Index page</h1>
</template>
```
```

```
```ts twoslash [pages/index.ts]
// https://vuejs.org/guide/extras/render-function.html
export default defineComponent({
 render () {
 return h('h1', 'Index page')
 }
})
```
```

```
```tsx twoslash [pages/index.tsx]
// https://nuxt.com/docs/examples/advanced/jsx
// https://vuejs.org/guide/extras/render-function.html#jsx-tsx
export default defineComponent({
 render () {
 return <h1>Index page</h1>
 }
})
```
```

::

The `pages/index.vue` file will be mapped to the `/` route of your application.

If you are using [`app.vue`](/docs/guide/directory-structure/app), make sure to use the [`<NuxtPage/>`](/docs/api/components/nuxt-page) component to display the current page:

```
```vue [app.vue]
<template>
 <div>
 <!-- Markup shared across all pages, ex: NavBar -->
 <NuxtPage />
 </div>
</template>
```
```

Pages **must** have a single root element to allow [route transitions](/docs/getting-started/transitions) between pages. HTML comments are considered elements as well.

This means that when the route is server-rendered, or statically generated, you will be able to see its contents correctly, but when you navigate towards that route during client-side navigation the transition between routes will fail and you'll see that the route will not be rendered.

Here are some examples to illustrate what a page with a single root element looks like:

::code-group

```
```vue [pages/working.vue]
<template>
 <div>
 <!-- This page correctly has only one single root element -->
 Page content
 </div>
</template>
```
```



```
```vue [pages/bad-1.vue]
<template>
 <!-- This page will not render when route changes during client side navigation, because of this comment -->
 <div>Page content</div>
</template>
```
```

```
```vue [pages/bad-2.vue]
<template>
 <div>This page</div>
 <div>Has more than one root element</div>
 <div>And will not render when route changes during client side navigation</div>
</template>
```
```

::

Dynamic Routes

If you place anything within square brackets, it will be turned into a [dynamic route] (<https://router.vuejs.org/guide/essentials/dynamic-matching.html>) parameter. You can mix and match multiple parameters and even non-dynamic text within a file name or directory.

If you want a parameter to be `_optional_`, you must enclose it in double square brackets - for example, `~/pages/[[slug]]/index.vue` or ~/pages/[[slug]].vue` will match both `/` and `/test`.`

```
```bash [Directory Structure]
-| pages/
---| index.vue
---| users-[group]/
----| [id].vue
```
```

Given the example above, you can access `group/id` within your component via the ``$route`` object:

```
```vue [pages/users-[group\\]/[id\\].vue]
<template>
 <p>{{ $route.params.group }} - {{ $route.params.id }}</p>
</template>
```
```

Navigating to ``/users-admins/123`` would render:

```
```html
<p>admins - 123</p>
```
```

If you want to access the route using Composition API, there is a global [``useRoute``] ([/docs/api/composables/use-route](https://router.vuejs.org/docs/api/composables/use-route)) function that will allow you to access the route just like ``this.$route`` in the Options API.

```
```vue twoslash
<script setup lang="ts">
const route = useRoute()

if (route.params.group === 'admins' && !route.params.id) {
 console.log('Warning! Make sure user is authenticated!')
}
</script>
```
```

::note
Named parent routes will take priority over nested dynamic routes. For the ``/foo/hello`` route, `~/pages/foo.vue` will take priority over ~/pages/foo/[slug].vue`. :br Use ~/pages/foo/index.vue` and ~/pages/foo/[slug].vue` to match `/foo` and `/foo/hello` with different pages,.
::`

Catch-all Route

If you need a catch-all route, you create it by using a file named like ``[...slug].vue``. This will match `_all_` routes under that path.

```
```vue [pages/[...slug\\].vue]
<template>
 <p>{{ $route.params.slug }}</p>
</template>
```
```

Navigating to ``/hello/world`` would render:

```
```html
<p>["hello", "world"]</p>
```
```

Nested Routes

It is possible to display [nested routes](https://next.router.vuejs.org/guide/essentials/nested-routes.html) with ``.

Example:

```
```bash [Directory Structure]
-| pages/
---| parent/
-----| child.vue
---| parent.vue
```
```

This file tree will generate these routes:

```
```js
[
 {
 path: '/parent',
 component: '~/pages/parent.vue',
 name: 'parent',
 children: [
 {
 path: 'child',
 component: '~/pages/parent/child.vue',
 name: 'parent-child'
 }
]
 }
]
```
```

To display the `child.vue` component, you have to insert the `` component inside `pages/parent.vue`:

```
```vue {[pages/parent.vue]
<template>
 <div>
 <h1>I am the parent view</h1>
 <NuxtPage :foobar="123" />
 </div>
</template>
```
```

```
```vue {[pages/parent/child.vue]
<script setup lang="ts">
const props = defineProps(['foobar'])

console.log(props.foobar)
</script>
```
```

Child Route Keys

If you want more control over when the `` component is re-rendered (for example, for transitions), you can either pass a string or function via the `pageKey` prop, or you can define a `key` value via `definePageMeta`:

```
```vue {[pages/parent.vue]
<template>
 <div>
 <h1>I am the parent view</h1>
 <NuxtPage :page-key="route => route.fullPath" />
 </div>
</template>
```
```

Or alternatively:

```
```vue twoslash {[pages/parent/child.vue]
<script setup lang="ts">
definePageMeta({
 key: route => route.fullPath
})
</script>
```
```

```
:link-example{to="/docs/examples/routing/pages"}
```

Route Groups

In some cases, you may want to group a set of routes together in a way which doesn't affect file-based routing. For this purpose, you can put files in a folder which is wrapped in parentheses - `(` and `)`.

For example:

```
```bash [Directory structure]
-| pages/
---| index.vue
---| (marketing)/
----| about.vue
----| contact.vue
```
```

This will produce `/`, `/about` and `/contact` pages in your app. The `marketing` group is ignored for purposes of your URL structure.

Page Metadata

You might want to define metadata for each route in your app. You can do this using the `definePageMeta` macro, which will work both in `

`layout`

You can define the layout used to render the route. This can be either `false` (to disable any layout), a string or a `ref/computed`, if you want to make it reactive in some way. [More about layouts](/docs/guide/directory-structure/layouts).

`layoutTransition` and `pageTransition`

You can define transition properties for the `` component that wraps your pages and layouts, or pass `false` to disable the `` wrapper for that route. You can see a list of options that can be passed [here] (<https://vuejs.org/api/built-in-components.html#transition>) or read [more about how transitions work] (<https://vuejs.org/guide/built-ins/transition.html#transition>).

You can set default values for these properties [in your `nuxt.config`](/docs/api/nuxt-config#layouttransition).

`middleware`

You can define middleware to apply before loading this page. It will be merged with all the other middleware used in any matching parent/child routes. It can be a string, a function (an anonymous/inlined middleware function following [the global before guard pattern] (<https://router.vuejs.org/guide/advanced/navigation-guards.html#global-before-guards>)), or an array of strings/functions. [More about named middleware](/docs/guide/directory-structure/middleware).

`name`

You may define a name for this page's route.

`path`

You may define a path matcher, if you have a more complex pattern than can be expressed with the file name. See [the `vue-router` docs] (<https://router.vuejs.org/guide/essentials/route-matching-syntax.html#custom-regex-in-params>) for more information.

Typing Custom Metadata

If you add custom metadata for your pages, you may wish to do so in a type-safe way. It is possible to augment the type of the object accepted by `definePageMeta`:

```
``ts [index.d.ts]
declare module '#app' {
  interface PageMeta {
    pageType?: string
  }
}

// It is always important to ensure you import/export something when augmenting a type
export {}
````
```

### ## Navigation

To navigate between pages of your app, you should use the [`<NuxtLink>`](/docs/api/components/nuxt-link) component.

This component is included with Nuxt and therefore you don't have to import it as you do with other components.

A simple link to the `index.vue` page in your `pages` folder:

```
``vue
<template>
 <NuxtLink to="/">Home page</NuxtLink>
</template>
````

::read-more{to="/docs/api/components/nuxt-link"}
Learn more about `<NuxtLink>` usage.
::
```

Programmatic Navigation

Nuxt allows programmatic navigation through the `navigateTo()` utility method. Using this utility method, you will be able to programmatically navigate the user in your app. This is great for taking input from the user and navigating them dynamically throughout your application. In this example, we have a simple method called `navigate()` that gets called when the user submits a search form.

```
::note
Ensure to always `await` on `navigateTo` or chain its result by returning from functions.
::

``vue twoslash
<script setup lang="ts">
const name = ref('');
const type = ref(1);

function navigate(){
```

```

return navigateTo({
  path: '/search',
  query: {
    name: name.value,
    type: type.value
  }
})
}
</script>
`

```

Client-Only Pages

You can define a page as [client only](/docs/guide/directory-structure/components#client-components) by giving it a `.client.vue` suffix. None of the content of this page will be rendered on the server.

Server-Only Pages

You can define a page as [server only](/docs/guide/directory-structure/components#server-components) by giving it a `.server.vue` suffix. While you will be able to navigate to the page using client-side navigation, controlled by `vue-router`, it will be rendered with a server component automatically, meaning the code required to render the page will not be in your client-side bundle.

```

::alert{type=warning}
Server-only pages must have a single root element. (HTML comments are considered elements as well.)
::

```

Custom Routing

As your app gets bigger and more complex, your routing might require more flexibility. For this reason, Nuxt directly exposes the router, routes and router options for customization in different ways.

```

:read-more{to="/docs/guide/recipes/custom-routing"}

```

Multiple Pages Directories

By default, all your pages should be in one `pages` directory at the root of your project.

However, you can use [Nuxt Layers](/docs/getting-started/layers) to create groupings of your app's pages:

```

` ` ` bash [Directory Structure]
- | some-app/
--- | nuxt.config.ts
--- | pages
---- | app-page.vue
- | nuxt.config.ts
` ` `

` ` ` ts twoslash [some-app/nuxt.config.ts]
// some-app/nuxt.config.ts
export default defineNuxtConfig({
})
` ` `

```

```

` ` ` ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  extends: ['./some-app'],
})
` ` `

```

```

:read-more{to="/docs/guide/going-further/layers"}

```

```

---
title: 'Transitions'
description: Apply transitions between pages and layouts with Vue or native browser View Transitions.
navigation.icon: i-ph-exclude-square-duotone
---

::note
Nuxt leverages Vue's [<Transition>](https://vuejs.org/guide/built-ins/transition.html#the-transition-component) component
to apply transitions between pages and layouts.
::

## Page transitions

You can enable page transitions to apply an automatic transition for all your [pages](/docs/guide/directory-structure/pages).

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 app: {
 pageTransition: { name: 'page', mode: 'out-in' }
 },
})
```

::note
If you are changing layouts as well as page, the page transition you set here will not run. Instead, you should set a [layout
transition](/docs/getting-started/transitions#layout-transitions).
::

To start adding transition between your pages, add the following CSS to your [app.vue](/docs/guide/directory-
structure/app):

::code-group

```vue [app.vue]
<template>
 <NuxtPage />
</template>

<style>
.page-enter-active,
.page-leave-active {
 transition: all 0.4s;
}
.page-enter-from,
.page-leave-to {
 opacity: 0;
 filter: blur(1rem);
}
</style>
```

```vue [pages/index.vue]
<template>
 <div>
 <h1>Home page</h1>
 <NuxtLink to="/about">About page</NuxtLink>
 </div>
</template>
```

```vue [pages/about.vue]
<template>
 <div>
 <h1>About page</h1>
 <NuxtLink to="/">Home page</NuxtLink>
 </div>
</template>
```

::

This produces the following result when navigating between pages:

<video controls class="rounded" poster="https://res.cloudinary.com/nuxt/video/upload/v1665061349/nuxt3/nuxt3-page-
transitions_umwvmh.jpg">
  <source src="https://res.cloudinary.com/nuxt/video/upload/v1665061349/nuxt3/nuxt3-page-transitions_umwvmh.mp4"
type="video/mp4">
</video>

To set a different transition for a page, set the pageTransition key in [definePageMeta](/docs/api/utils/define-page-
meta) of the page:

::code-group

```

```

```vue twoslash [pages/about.vue]
<script setup lang="ts">
definePageMeta({
 pageTransition: {
 name: 'rotate'
 }
})
</script>
```

```

```

```vue [app.vue]
<template>
 <NuxtPage />
</template>

<style>
/* ... */
.rotate-enter-active,
.rotate-leave-active {
 transition: all 0.4s;
}
.rotate-enter-from,
.rotate-leave-to {
 opacity: 0;
 transform: rotate3d(1, 1, 1, 15deg);
}
</style>
```

```

::

Moving to the about page will add the 3d rotation effect:

```

<video controls class="rounded" poster="https://res.cloudinary.com/nuxt/video/upload/v1665063233/nuxt3/nuxt3-page-transitions-cutom.jpg">
  <source src="https://res.cloudinary.com/nuxt/video/upload/v1665063233/nuxt3/nuxt3-page-transitions-cutom.mp4"
  type="video/mp4">
</video>

```

Layout transitions

You can enable layout transitions to apply an automatic transition for all your [layouts](/docs/guide/directory-structure/layouts).

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 app: {
 layoutTransition: { name: 'layout', mode: 'out-in' }
 },
})
```

```

To start adding transition between your pages and layouts, add the following CSS to your [`app.vue`](/docs/guide/directory-structure/app):

::code-group

```

```vue [app.vue]
<template>
 <NuxtLayout>
 <NuxtPage />
 </NuxtLayout>
</template>

<style>
.layout-enter-active,
.layout-leave-active {
 transition: all 0.4s;
}
.layout-enter-from,
.layout-leave-to {
 filter: grayscale(1);
}
</style>
```

```

```

```vue [layouts/default.vue]
<template>
 <div>
 <pre>default layout</pre>
 <slot />
 </div>

```

```

</template>

<style scoped>
div {
 background-color: lightgreen;
}
</style>
```

```vue [layouts/orange.vue]
<template>
 <div>
 <pre>orange layout</pre>
 <slot />
 </div>
</template>

<style scoped>
div {
 background-color: #eebb90;
 padding: 20px;
 height: 100vh;
}
</style>
```

```vue [pages/index.vue]
<template>
 <div>
 <h1>Home page</h1>
 <NuxtLink to="/about">About page</NuxtLink>
 </div>
</template>
```

```vue [pages/about.vue]
<script setup lang="ts">
definePageMeta({
 layout: 'orange'
})
</script>

<template>
 <div>
 <h1>About page</h1>
 <NuxtLink to="/">Home page</NuxtLink>
 </div>
</template>
```

```

::

This produces the following result when navigating between pages:

```

<video controls class="rounded" poster="https://res.cloudinary.com/nuxt/video/upload/v1665065289/nuxt3/nuxt3-layouts-transitions_c9hwlx.jpg">
  <source src="https://res.cloudinary.com/nuxt/video/upload/v1665065289/nuxt3/nuxt3-layouts-transitions_c9hwlx.mp4"
  type="video/mp4">
</video>

```

Similar to `pageTransition`, you can apply a custom `layoutTransition` to the page component using `definePageMeta`:

```

```vue twoslash [pages/about.vue]
<script setup lang="ts">
definePageMeta({
 layout: 'orange',
 layoutTransition: {
 name: 'slide-in'
 }
})
</script>
```

```

Global settings

You can customize these default transition names globally using `nuxt.config`.

Both `pageTransition` and `layoutTransition` keys accept [`TransitionProps`](https://vuejs.org/api/built-in-components.html#transition) as JSON serializable values where you can pass the `name`, `mode` and other valid transition-props of the custom CSS transition.

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({

```



```

app: {
 pageTransition: {
 name: 'fade',
 mode: 'out-in' // default
 },
 layoutTransition: {
 name: 'slide',
 mode: 'out-in' // default
 }
}
})
```

```

::warning
 If you change the `name` property, you also have to rename the CSS classes accordingly.
 ::

To override the global transition property, use the `definePageMeta` to define page or layout transitions for a single Nuxt page and override any page or layout transitions that are defined globally in `nuxt.config` file.

```

```vue twoslash [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
 pageTransition: {
 name: 'bounce',
 mode: 'out-in' // default
 }
})
</script>
```

```

Disable Transitions

`pageTransition` and `layoutTransition` can be disabled for a specific route:

```

```vue twoslash [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
 pageTransition: false,
 layoutTransition: false
})
</script>
```

```

Or globally in the `nuxt.config`:

```

```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 app: {
 pageTransition: false,
 layoutTransition: false
 }
})
```

```

JavaScript Hooks

For advanced use-cases, you can use JavaScript hooks to create highly dynamic and custom transitions for your Nuxt pages.

This way presents perfect use-cases for JavaScript animation libraries such as [GSAP](https://gsap.com).

```

```vue twoslash [pages/some-page.vue]
<script setup lang="ts">
definePageMeta({
 pageTransition: {
 name: 'custom-flip',
 mode: 'out-in',
 onBeforeEnter: (el) => {
 console.log('Before enter...')
 },
 onEnter: (el, done) => {},
 onAfterEnter: (el) => {}
 }
})
</script>
```

```

::tip
 Learn more about additional [JavaScript hooks](https://vuejs.org/guide/built-ins/transition.html#javascript-hooks) available in the `Transition` component.
 ::

Dynamic Transitions

To apply dynamic transitions using conditional logic, you can leverage inline [middleware](/docs/guide/directory-structure/middleware) to assign a different transition name to `to.meta.pageTransition`.

::code-group

```
```\vue twoslash [pages/[id\]].vue]
<script setup lang="ts">
definePageMeta({
 pageTransition: {
 name: 'slide-right',
 mode: 'out-in'
 },
 middleware (to, from) {
 if (to.meta.pageTransition && typeof to.meta.pageTransition !== 'boolean')
 to.meta.pageTransition.name = +to.params.id! > +from.params.id! ? 'slide-left' : 'slide-right'
 }
})
</script>

<template>
 <h1>#{ $route.params.id }</h1>
</template>

<style>
.slide-left-enter-active,
.slide-left-leave-active,
.slide-right-enter-active,
.slide-right-leave-active {
 transition: all 0.2s;
}
.slide-left-enter-from {
 opacity: 0;
 transform: translate(50px, 0);
}
.slide-left-leave-to {
 opacity: 0;
 transform: translate(-50px, 0);
}
.slide-right-enter-from {
 opacity: 0;
 transform: translate(-50px, 0);
}
.slide-right-leave-to {
 opacity: 0;
 transform: translate(50px, 0);
}
</style>
```\`
```

```
```\vue [layouts/default.vue]
<script setup lang="ts">
const route = useRoute()
const id = computed(() => Number(route.params.id || 1))
const prev = computed(() => '/' + (id.value - 1))
const next = computed(() => '/' + (id.value + 1))
</script>
```

```
<template>
 <div>
 <slot />
 <div v-if="$route.params.id">
 <NuxtLink :to="prev">âˆ’ï</NuxtLink> |
 <NuxtLink :to="next">âž‰ï</NuxtLink>
 </div>
 </div>
</template>
```\`
```

::

The page now applies the `slide-left` transition when going to the next id and `slide-right` for the previous:

```
<video controls class="rounded" poster="https://res.cloudinary.com/nuxt/video/upload/v1665069410/nuxt3/nuxt-dynamic-page-transitions.jpg">
  <source src="https://res.cloudinary.com/nuxt/video/upload/v1665069410/nuxt3/nuxt-dynamic-page-transitions.mp4"
  type="video/mp4">
</video>
```

Transition with NuxtPage

When `

```

``vue [app.vue]
<template>
  <div>
    <NuxtLayout>
      <NuxtPage :transition="{
        name: 'bounce',
        mode: 'out-in'
      }" />
    </NuxtLayout>
  </div>
</template>
``

```

```

::note
Remember, this page transition cannot be overridden with `definePageMeta` on individual pages.
::

```

View Transitions API (experimental)

Nuxt ships with an experimental implementation of the [**View Transitions API**](https://developer.chrome.com/docs/web-platform/view-transitions) (see [MDN](https://developer.mozilla.org/en-US/docs/Web/API/View_Transitions_API)). This is an exciting new way to implement native browser transitions which (among other things) have the ability to transition between unrelated elements on different pages.

You can check a demo on <https://nuxt-view-transitions.surge.sh> and the [source on StackBlitz](https://stackblitz.com/edit/nuxt-view-transitions).

The Nuxt integration is under active development, but can be enabled with the `experimental.viewTransition` option in your configuration file:

```

``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  experimental: {
    viewTransition: true
  }
})
``

```

The possible values are: `false`, `true`, or `always`.

If set to true, Nuxt will not apply transitions if the user's browser matches `prefers-reduced-motion: reduce` (recommended). If set to `always`, Nuxt will always apply the transition and it is up to you to respect the user's preference.

By default, view transitions are enabled for all [pages](/docs/guide/directory-structure/pages), but you can set a different global default.

```

``ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  app: {
    // Disable view transitions globally, and opt-in on a per page basis
    viewTransition: false
  },
})
``

```

It is possible to override the default `viewTransition` value for a page by setting the `viewTransition` key in [definePageMeta](/docs/api/utils/define-page-meta) of the page:

```

``vue twoslash [pages/about.vue]
<script setup lang="ts">
definePageMeta({
  viewTransition: false
})
</script>
``

```

```

::alert{type="warning"}
Overriding view transitions on a per-page basis will only have an effect if you have enabled the
`experimental.viewTransition` option.
::

```

If you are also using Vue transitions like `pageTransition` and `layoutTransition` (see above) to achieve the same result as the new View Transitions API, then you may wish to disable Vue transitions if the user's browser supports the newer, native web API. You can do this by creating `~/middleware/disable-vue-transitions.global.ts` with the following contents:

```

``ts
export default defineNuxtRouteMiddleware(to => {
  if (import.meta.server || !document.startViewTransition) { return }

  // Disable built-in Vue transitions
  to.meta.pageTransition = false
  to.meta.layoutTransition = false

```

```
})  
``,`
```

Known issues

- If you perform data fetching within your page setup functions, that you may wish to reconsider using this feature for the moment. (By design, View Transitions completely freeze DOM updates whilst they are taking place.) We're looking at restricting the View Transition to the final moments before ``<Suspense>`` resolves, but in the interim you may want to consider carefully whether to adopt this feature if this describes you.

```
---
title: Authoring Nuxt Layers
description: Nuxt provides a powerful system that allows you to extend the default files, configs, and much more.
---
```

Nuxt layers are a powerful feature that you can use to share and reuse partial Nuxt applications within a monorepo, or from a git repository or npm package. The layers structure is almost identical to a standard Nuxt application, which makes them easy to author and maintain.

```
:read-more{to="/docs/getting-started/layers"}
```

A minimal Nuxt layer directory should contain a [`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config) file to indicate it is a layer.

```
``ts [base/nuxt.config.ts]
export default defineNuxtConfig({})
````
```

Additionally, certain other files in the layer directory will be auto-scanned and used by Nuxt for the project extending this layer.

- [`components/*`](/docs/guide/directory-structure/components) - Extend the default components
- [`composables/*`](/docs/guide/directory-structure/composables) - Extend the default composables
- [`layouts/*`](/docs/guide/directory-structure/layouts) - Extend the default layouts
- [`pages/*`](/docs/guide/directory-structure/pages) - Extend the default pages
- [`plugins/*`](/docs/guide/directory-structure/plugins) - Extend the default plugins
- [`server/*`](/docs/guide/directory-structure/server) - Extend the default server endpoints & middleware
- [`utils/*`](/docs/guide/directory-structure/utils) - Extend the default utils
- [`nuxt.config.ts`](/docs/guide/directory-structure/nuxt-config) - Extend the default nuxt config
- [`app.config.ts`](/docs/guide/directory-structure/app-config) - Extend the default app config

## ## Basic Example

```
::code-group
```

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
 extends: [
 './base'
]
})
````
```

```
``vue [app.vue]
<template>
  <BaseComponent/>
</template>
````
```

```
``ts [base/nuxt.config.ts]
export default defineNuxtConfig({
 // Extending from base nuxt.config.ts!
 app: {
 head: {
 title: 'Extending Configs is Fun!',
 meta: [
 { name: 'description', content: 'I am using the extends feature in nuxt 3!' }
],
 }
 }
})
````
```

```
``vue [base/components/BaseComponent.vue]
<template>
  <h1>Extending Components is Fun!</h1>
</template>
````
```

```
::
```

## ## Starter Template

To get started you can initialize a layer with the [nuxt/starter/layer template](https://github.com/nuxt/starter/tree/layer). This will create a basic structure you can build upon. Execute this command within the terminal to get started:

```
``bash [Terminal]
npx nuxi init --template layer nuxt-layer
````
```

Follow up on the README instructions for the next steps.

Publishing Layers

You can publish and share layers by either using a remote source or an npm package.

Git Repository

You can use a git repository to share your Nuxt layer. Some examples:

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  extends: [
    'github:username/repoName',          // GitHub Remote Source
    'github:username/repoName/base',    // GitHub Remote Source within /base directory
    'github:username/repoName#dev',     // GitHub Remote Source from dev branch
    'github:username/repoName#v1.0.0',  // GitHub Remote Source from v1.0.0 tag
    'gitlab:username/repoName',         // GitLab Remote Source example
    'bitbucket:username/repoName',      // Bitbucket Remote Source example
  ]
})
``
```

::tip

If you want to extend a private remote source, you need to add the environment variable ``GIGET_AUTH=<token>`` to provide a token.

::

::tip

If you want to extend a remote source from a self-hosted GitHub or GitLab instance, you need to supply its URL with the ``GIGET_GITHUB_URL=<url>`` or ``GIGET_GITLAB_URL=<url>`` environment variable - or directly configure it with [the ``auth`` option] (<https://github.com/unjs/c12#extending-config-layer-from-remote-sources>) in your ``nuxt.config``.

::

::note

When using git remote sources, if a layer has npm dependencies and you wish to install them, you can do so by specifying ``install: true`` in your layer options.

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  extends: [
    ['github:username/repoName', { install: true }]
  ]
})
``
::
```

npm Package

You can publish Nuxt layers as an npm package that contains the files and dependencies you want to extend. This allows you to share your config with others, use it in multiple projects or use it privately.

To extend from an npm package, you need to make sure that the module is published to npm and installed in the user's project as a devDependency. Then you can use the module name to extend the current nuxt config:

```
``ts [nuxt.config.ts]
export default defineNuxtConfig({
  extends: [
    // Node Module with scope
    '@scope/moduleName',
    // or just the module name
    'moduleName'
  ]
})
``
```

To publish a layer directory as an npm package, you want to make sure that the ``package.json`` has the correct properties filled out. This will make sure that the files are included when the package is published.

```
``json [package.json]
{
  "name": "my-theme",
  "version": "1.0.0",
  "type": "module",
  "main": "./nuxt.config.ts",
  "dependencies": {},
  "devDependencies": {
    "nuxt": "^3.0.0"
  }
}
``
```

::important

Make sure any dependency imported in the layer is **explicitly added** to the ``dependencies``. The ``nuxt`` dependency, and anything only used for testing the layer before publishing, should remain in the ``devDependencies`` field.

::

Now you can proceed to publish the module to npm, either publicly or privately.

::important

When publishing the layer as a private npm package, you need to make sure you log in, to authenticate with npm to download the node module.

::

Tips

Relative Paths and Aliases

When importing using aliases (such as `~/` and `@/`) in a layer components and composables, note that aliases are resolved relative to the user's project paths. As a workaround, you can **use relative paths** to import them. We are working on a better solution for named layer aliases.

Also when using relative paths in `nuxt.config` file of a layer, (with exception of nested `extends`) they are resolved relative to user's project instead of the layer. As a workaround, use full resolved paths in `nuxt.config`:

```
```js [nuxt.config.ts]
import { fileURLToPath } from 'url'
import { dirname, join } from 'path'

const currentDir = dirname(fileURLToPath(import.meta.url))

export default defineNuxtConfig({
 css: [
 join(currentDir, './assets/main.css')
]
})
```
```

Multi-Layer Support for Nuxt Modules

You can use the internal array `nuxt.options._layers` to support custom multi-layer handling for your modules.

```
```ts [modules/my-module.ts]
export default defineNuxtModule({
 setup(options, nuxt) {
 for (const layer of nuxt.options._layers) {
 // You can check for a custom directory existence to extend for each layer
 console.log('Custom extension for', layer.cwd, layer.config)
 }
 }
})
```
```

****Notes:****

- Earlier items in the `_layers` array have higher priority and override later ones
- The user's project is the first item in the `_layers` array

Going Deeper

Configuration loading and extends support is handled by [unjs/c12](https://github.com/unjs/c12), merged using [unjs/defu](https://github.com/unjs/defu) and remote git sources are supported using [unjs/giget](https://github.com/unjs/giget). Check the docs and source code to learn more.

::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/issues/13367" target="_blank"}
Checkout our ongoing development to bring more improvements for layers support on GitHub.
::

```
---
title: "<NuxtIsland>"
description: "Nuxt provides the <NuxtIsland> component to render a non-interactive component without any client JS."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/components/nuxt-island.ts
    size: xs
---
```

When rendering an island component, the content of the island component is static, thus no JS is downloaded client-side.

Changing the island component props triggers a refetch of the island component to re-render it again.

```
::note
Global styles of your application are sent with the response.
::
```

```
::tip
Server only components use `<NuxtIsland>` under the hood
::
```

Props

- `name` : Name of the component to render.
 - **type**: `string`
 - **required**
- `lazy` : Make the component non-blocking.
 - **type**: `boolean`
 - **default**: `false`
- `props` : Props to send to the component to render.
 - **type**: `Record<string, any>`
- `source` : Remote source to call the island to render.
 - **type**: `string`
- **dangerouslyLoadClientComponents**: Required to load components from a remote source.
 - **type**: `boolean`
 - **default**: `false`

```
::note
Remote islands need `experimental.componentIslands` to be `'local+remote'` in your `nuxt.config`.
It is strongly discouraged to enable `dangerouslyLoadClientComponents` as you can't trust a remote server's javascript.
::
```

Slots

Slots can be passed to an island component if declared.

Every slot is interactive since the parent component is the one providing it.

Some slots are reserved to `NuxtIsland` for special cases.

- `#fallback` : Specify the content to be rendered before the island loads (if the component is lazy) or if `NuxtIsland` fails to fetch the component.

Ref

- `refresh()`
 - **type**:`() => Promise<void>`
 - **description**: force refetch the server component by refetching it.

Events

- `error`
 - **parameters**:
 - **error**:
 - **type**: `unknown`
 - **description**: emitted when when `NuxtIsland` fails to fetch the new island.


```

---
title: 'useLoadingIndicator'
description: This composable gives you access to the loading state of the app page.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/loading-indicator.ts
    size: xs
---

```

Description

A composable which returns the loading state of the page. Used by [`<NuxtLoadingIndicator>`](/docs/api/components/nuxt-loading-indicator) and controllable.

It hooks into [`page:loading:start`](/docs/api/advanced/hooks#app-hooks-runtime) and [`page:loading:end`](/docs/api/advanced/hooks#app-hooks-runtime) to change its state.

Parameters

- `duration`: Duration of the loading bar, in milliseconds (default `2000`).
- `throttle`: Throttle the appearing and hiding, in milliseconds (default `200`).
- `estimatedProgress`: By default Nuxt will back off as it approaches 100%. You can provide a custom function to customize the progress estimation, which is a function that receives the duration of the loading bar (above) and the elapsed time. It should return a value between 0 and 100.

Properties

`isLoading`

- **type**: `Ref<boolean>`
- **description**: The loading state

`error`

- **type**: `Ref<boolean>`
- **description**: The error state

`progress`

- **type**: `Ref<number>`
- **description**: The progress state. From `0` to `100`.

Methods

`start()`

Set `isLoading` to true and start to increase the `progress` value.

`finish()`

Set the `progress` value to `100`, stop all timers and intervals then reset the loading state `500` ms later. `finish` accepts a `{ force: true }` option to skip the interval before the state is reset, and `{ error: true }` to change the loading bar color and set the error property to true.

`clear()`

Used by `finish()`. Clear all timers and intervals used by the composable.

Example

```

```vue
<script setup lang="ts">
 const { progress, isLoading, start, finish, clear } = useLoadingIndicator({
 duration: 2000,
 throttle: 200,
 // This is how progress is calculated by default
 estimatedProgress: (duration, elapsed) => (2 / Math.PI * 100) * Math.atan(elapsed / duration * 100 / 50)
 })
</script>
```

```

```
---
title: "<NuxtImg>"
description: "Nuxt provides a <NuxtImg> component to handle automatic image optimization."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/image/blob/main/src/runtime/components/nuxt-img.ts
    size: xs
---
```

`<NuxtImg>` is a drop-in replacement for the native `` tag.

- Uses built-in provider to optimize local and remote images
- Converts `src` to provider-optimized URLs
- Automatically resizes images based on `width` and `height`
- Generates responsive sizes when providing `sizes` option
- Supports native lazy loading as well as other `` attributes

Setup

In order to use `<NuxtImg>` you should install and enable the Nuxt Image module:

```
```bash [Terminal]
npx nuxi@latest module add image
```
```

Usage

`<NuxtImg>` outputs a native `img` tag directly (without any wrapper around it). Use it like you would use the `` tag:

```
```html
<NuxtImg src="/nuxt-icon.png" />
```
```

Will result in:

```
```html

```
```

```
::read-more{to="https://image.nuxt.com/usage/nuxt-img" target="_blank"}
Read more about the <NuxtImg> component.
::
```

```
---
title: "public"
description: "The public/ directory is used to serve your website's static assets."
head.title: "public/"
navigation.icon: i-ph-folder-duotone
---
```

Files contained within the `public/` directory are served at the root and are not modified by the build process. This is suitable for files that have to keep their names (e.g. `robots.txt`) `_or_` likely won't change (e.g. `favicon.ico`).

```
```bash [Directory structure]
-| public/
---| favicon.ico
---| og-image.png
---| robots.txt
```
```

```
```vue [app.vue]
<script setup lang="ts">
useSeoMeta({
 ogImage: '/og-image.png'
})
</script>
```
```

```
:::tip{to="https://v2.nuxt.com/docs/directory-structure/static" target="_blank"}
This is known as the [static/] directory in Nuxt 2.
:::
```

```

---
title: 'useNuxtApp'
description: 'Access the shared runtime context of the Nuxt Application.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts
    size: xs
---

```

`useNuxtApp` is a built-in composable that provides a way to access shared runtime context of Nuxt, also known as the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context), which is available on both client and server side (but not within Nitro routes). It helps you access the Vue app instance, runtime hooks, runtime config variables and internal states, such as `ssrContext` and `payload`.

```

```vue [app.vue]
<script setup lang="ts">
const nuxtApp = useNuxtApp()
</script>
```

```

If runtime context is unavailable in your scope, `useNuxtApp` will throw an exception when called. You can use `tryUseNuxtApp`(#tryusenuxtapp) instead for composables that do not require `nuxtApp`, or to simply check if context is available or not without an exception.

```

<!--
note
By default, the shared runtime context of Nuxt is namespaced under the [buildId](/docs/api/nuxt-config#buildid) option. It allows the support of multiple runtime contexts.

```

Params

- `appName`: an optional application name. If you do not provide it, the Nuxt `buildId` option is used. Otherwise, it must match with an existing `buildId`. -->

Methods

`provide (name, value)`

`nuxtApp` is a runtime context that you can extend using [Nuxt plugins](/docs/guide/directory-structure/plugins). Use the `provide` function to create Nuxt plugins to make values and helper methods available in your Nuxt application across all composables and components.

`provide` function accepts `name` and `value` parameters.

```

```js
const nuxtApp = useNuxtApp()
nuxtApp.provide('hello', (name) => `Hello ${name}!`)

// Prints "Hello name!"
console.log(nuxtApp.$hello('name'))
```

```

As you can see in the example above, `$hello` has become the new and custom part of `nuxtApp` context and it is available in all places where `nuxtApp` is accessible.

`hook(name, cb)`

Hooks available in `nuxtApp` allows you to customize the runtime aspects of your Nuxt application. You can use runtime hooks in Vue.js composables and [Nuxt plugins](/docs/guide/directory-structure/plugins) to hook into the rendering lifecycle.

`hook` function is useful for adding custom logic by hooking into the rendering lifecycle at a specific point. `hook` function is mostly used when creating Nuxt plugins.

See [Runtime Hooks](/docs/api/advanced/hooks#app-hooks-runtime) for available runtime hooks called by Nuxt.

```

```ts [plugins/test.ts]
export default defineNuxtPlugin((nuxtApp) => {
 nuxtApp.hook('page:start', () => {
 /* your code goes here */
 })
 nuxtApp.hook('vue:error', (..._args) => {
 console.log('vue:error')
 // if (import.meta.client) {
 // console.log(..._args)
 // }
 })
})
```

```

`callHook(name, ...args)`

`callHook` returns a promise when called with any of the existing hooks.

```
``ts
await nuxtApp.callHook('my-plugin:init')
``
```

Properties

``useNuxtApp()`` exposes the following properties that you can use to extend and customize your app and share state, data and variables.

``vueApp``

``vueApp`` is the global Vue.js [application instance](https://vuejs.org/api/application.html#application-api) that you can access through ``nuxtApp``.

Some useful methods:

- `[`component()`]`(https://vuejs.org/api/application.html#app-component) - Registers a global component if passing both a name string and a component definition, or retrieves an already registered one if only the name is passed.
- `[`directive()`]`(https://vuejs.org/api/application.html#app-directive) - Registers a global custom directive if passing both a name string and a directive definition, or retrieves an already registered one if only the name is passed[(example)](/docs/guide/directory-structure/plugins#vue-directives).
- `[`use()`]`(https://vuejs.org/api/application.html#app-use) - Installs a **[[Vue.js Plugin]]**(https://vuejs.org/guide/reusability/plugins.html)* [(example)](/docs/guide/directory-structure/plugins#vue-plugins).

:read-more{icon="i-simple-icons-vuedotjs" to="https://vuejs.org/api/application.html#application-api"}

``ssrContext``

``ssrContext`` is generated during server-side rendering and it is only available on the server side.

Nuxt exposes the following properties through ``ssrContext``:

- ``url`` (string) - Current request url.
- ``event`` ([unjs/h3](https://github.com/unjs/h3) request event) - Access the request & response of the current route.
- ``payload`` (object) - NuxtApp payload object.

``payload``

``payload`` exposes data and state variables from server side to client side. The following keys will be available on the client after they have been passed from the server side:

- ``serverRendered`` (boolean) - Indicates if response is server-side-rendered.
- ``data`` (object) - When you fetch the data from an API endpoint using either `[`useFetch`]`(/docs/api/composables/use-fetch) or `[`useAsyncData`]`(/docs/api/composables/use-async-data) , resulting payload can be accessed from the ``payload.data``. This data is cached and helps you prevent fetching the same data in case an identical request is made more than once.

```
::code-group
``vue [app.vue]
<script setup lang="ts">
const { data } = await useAsyncData('count', () => $fetch('/api/count'))
</script>
``
``ts [server/api/count.ts]
export default defineEventHandler(event => {
  return { count: 1 }
})
``
::
```

After fetching the value of ``count`` using `[`useAsyncData`]`(/docs/api/composables/use-async-data) in the example above, if you access ``payload.data``, you will see `{ count: 1 }` recorded there.

When accessing the same ``payload.data`` from `[`ssrcontext`]`(#ssrcontext), you can access the same value on the server side as well.

- ``state`` (object) - When you use `[`useState`]`(/docs/api/composables/use-state) composable in Nuxt to set shared state, this state data is accessed through ``payload.state.[name-of-your-state]``.

```
``ts [plugins/my-plugin.ts]
export const useColor = () => useState<string>('color', () => 'pink')

export default defineNuxtPlugin((nuxtApp) => {
  if (import.meta.server) {
    const color = useColor()
  }
})
``
```

It is also possible to use more advanced types, such as ``ref``, ``reactive``, ``shallowRef``, ``shallowReactive`` and ``NuxtError``.

Since [Nuxt v3.4](https://nuxt.com/blog/v3-4#payload-enhancements), it is possible to define your own reducer/reviver for types that are not supported by Nuxt.

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=8w6ffRBs8a4" target="_blank"}

Watch a video from Alexander Lichter about serializing payloads, especially with regards to classes.
::

In the example below, we define a reducer (or a serializer) and a reviver (or deserializer) for the [Luxon] (<https://moment.github.io/luxon/#/>) `DateTime` class, using a payload plugin.

```
``ts [plugins/date-time-payload.ts]
/**
 * This kind of plugin runs very early in the Nuxt lifecycle, before we revive the payload.
 * You will not have access to the router or other Nuxt-injected properties.
 *
 * Note that the "DateTime" string is the type identifier and must
 * be the same on both the reducer and the reviver.
 */
export default definePayloadPlugin((nuxtApp) => {
  definePayloadReducer('DateTime', (value) => {
    return value instanceof DateTime && value.toJSON()
  })
  definePayloadReviver('DateTime', (value) => {
    return DateTime.fromISO(value)
  })
})
``
```

`isHydrating`

Use `nuxtApp.isHydrating` (boolean) to check if the Nuxt app is hydrating on the client side.

```
``ts [components/nuxt-error-boundary.ts]
export default defineComponent({
  setup (_props, { slots, emit }) {
    const nuxtApp = useNuxtApp()
    onErrorCaptured((err) => {
      if (import.meta.client && !nuxtApp.isHydrating) {
        // ...
      }
    })
  }
})
``
```

`runWithContext`

::note

You are likely here because you got a "Nuxt instance unavailable" message. Please use this method sparingly, and report examples that are causing issues, so that it can ultimately be solved at the framework level.
::

The `runWithContext` method is meant to be used to call a function and give it an explicit Nuxt context. Typically, the Nuxt context is passed around implicitly and you do not need to worry about this. However, when working with complex `'async'/'await'` scenarios in middleware/plugins, you can run into instances where the current instance has been unset after an async call.

```
``ts [middleware/auth.ts]
export default defineNuxtRouteMiddleware(async (to, from) => {
  const nuxtApp = useNuxtApp()
  let user
  try {
    user = await fetchUser()
    // the Vue/Nuxt compiler loses context here because of the try/catch block.
  } catch (e) {
    user = null
  }
  if (!user) {
    // apply the correct Nuxt context to our `navigateTo` call.
    return nuxtApp.runWithContext(() => navigateTo('/auth'))
  }
})
``
```

Usage

```
``js
const result = nuxtApp.runWithContext(() => functionWithContext())
``
```

- `functionWithContext`: Any function that requires the context of the current Nuxt application. This context will be correctly applied automatically.

`runWithContext` will return whatever is returned by `functionWithContext`.

A Deeper Explanation of Context

Vue.js Composition API (and Nuxt composables similarly) work by depending on an implicit context. During the lifecycle, Vue sets the temporary instance of the current component (and Nuxt temporary instance of `nuxtApp`) to a global variable and unsets it in same tick. When rendering on the server side, there are multiple requests from different users and `nuxtApp` running in a same global context. Because of this, Nuxt and Vue immediately unset this global instance to avoid leaking a shared reference between two users or components.

What it does mean? The Composition API and Nuxt Composables are only available during lifecycle and in same tick before any async operation:

```
```js
// --- Vue internal ---
const _vueInstance = null
const getCurrentInstance = () => _vueInstance
// ---

// Vue / Nuxt sets a global variable referencing to current component in _vueInstance when calling setup()
async function setup() {
 getCurrentInstance() // Works
 await someAsyncOperation() // Vue unsets the context in same tick before async operation!
 getCurrentInstance() // null
}
```

The classic solution to this, is caching the current instance on first call to a local variable like `const instance = getCurrentInstance()` and use it in the next composable call but the issue is that any nested composable calls now needs to explicitly accept the instance as an argument and not depend on the implicit context of composition-api. This is design limitation with composables and not an issue per-se.

To overcome this limitation, Vue does some behind the scenes work when compiling our application code and restores context after each call for `<script setup>`:

```
```js
const __instance = getCurrentInstance() // Generated by Vue compiler
getCurrentInstance() // Works!
await someAsyncOperation() // Vue unsets the context
__restoreInstance(__instance) // Generated by Vue compiler
getCurrentInstance() // Still works!
```
```

For a better description of what Vue actually does, see [unjs/unctx#2 (comment)](<https://github.com/unjs/unctx/issues/2#issuecomment-942193723>).

#### #### Solution

This is where `runWithContext` can be used to restore context, similarly to how `<script setup>` works.

Nuxt internally uses [unjs/unctx](<https://github.com/unjs/unctx>) to support composables similar to Vue for plugins and middleware. This enables composables like `navigateTo()` to work without directly passing `nuxtApp` to them - bringing the DX and performance benefits of Composition API to the whole Nuxt framework.

Nuxt composables have the same design as the Vue Composition API and therefore need a similar solution to magically do this transform. Check out [unjs/unctx#2](<https://github.com/unjs/unctx/issues/2>) (proposal), [unjs/unctx#4](<https://github.com/unjs/unctx/pull/4>) (transform implementation), and [nuxt/framework#3884](<https://github.com/nuxt/framework/pull/3884>) (Integration to Nuxt).

Vue currently only supports async context restoration for `<script setup>` for `async/await` usage. In Nuxt 3, the transform support for `defineNuxtPlugin()` and `defineNuxtRouteMiddleware()` was added, which means when you use them Nuxt automatically transforms them with context restoration.

#### #### Remaining Issues

The `unjs/unctx` transformation to automatically restore context seems buggy with `try/catch` statements containing `await` which ultimately needs to be solved in order to remove the requirement of the workaround suggested above.

#### #### Native Async Context

Using a new experimental feature, it is possible to enable native async context support using [Node.js `AsyncLocalStorage`]([https://nodejs.org/api/async\\_context.html#class-asynclocalstorage](https://nodejs.org/api/async_context.html#class-asynclocalstorage)) and new unctx support to make async context available **natively** to **any nested async composable** without needing a transform or manual passing/calling with context.

```
:::tip
Native async context support works currently in Bun and Node.
::

:read-more{to="/docs/guide/going-further/experimental-features#asynccontext"}
```

#### ## tryUseNuxtApp

This function works exactly the same as `useNuxtApp`, but returns `null` if context is unavailable instead of throwing an exception.

You can use it for composables that do not require `nuxtApp`, or to simply check if context is available or not without an exception.

Example usage:

```
```ts [composable.ts]
export function useStandType() {
  // Always works on the client
  if (tryUseNuxtApp()) {
    return useRuntimeConfig().public.STAND_TYPE
  } else {
    return process.env.STAND_TYPE
  }
}
```
```

<!-- ### Params

- `appName`: an optional application name. If you do not provide it, the Nuxt `buildId` option is used. Otherwise, it must match with an existing `buildId`. -->



```

title: "Debugging"
description: "In Nuxt 3, you can get started with debugging your application directly in the browser as well as in your IDE."

```

## ## Sourcemaps

Sourcemaps are enabled for your server build by default, and for the client build in dev mode, but you can enable them more specifically in your configuration.

```
```:ts
export default defineNuxtConfig({
  // or sourcemap: true
  sourcemap: {
    server: true,
    client: true
  }
})
```:
```

## ## Debugging with Node Inspector

You can use [Node inspector](https://nodejs.org/en/learn/getting-started/debugging) to debug Nuxt server-side.

```
```:bash
nuxi dev --inspect
```:
```

This will start Nuxt in `dev` mode with debugger active. If everything is working correctly a Node.js icon will appear on your Chrome DevTools and you can attach to the debugger.

::important

Note that the Node.js and Chrome processes need to be run on the same platform. This doesn't work inside of Docker.

::

## ## Debugging in Your IDE

It is possible to debug your Nuxt app in your IDE while you are developing it.

### ### Example VS Code Debug Configuration

You may need to update the config below with a path to your web browser. For more information, visit the [VS Code documentation about debug configuration](https://go.microsoft.com/fwlink/?linkid=830387).

::important

If you use `pnpm`, you will need to have `nuxi` installed as a devDependency for the configuration below to work.

::

```
```:json5
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "client: chrome",
      "url": "http://localhost:3000",
      "webRoot": "${workspaceFolder}"
    },
    {
      "type": "node",
      "request": "launch",
      "name": "server: nuxt",
      "outputCapture": "std",
      "program": "${workspaceFolder}/node_modules/nuxi/bin/nuxi.mjs",
      "args": [
        "dev"
      ],
    }
  ],
  "compounds": [
    {
      "name": "fullstack: nuxt",
      "configurations": [
        "server: nuxt",
        "client: chrome"
      ]
    }
  ]
}
```:
```

If you prefer your usual browser extensions, add this to the `_chrome_` configuration above:

```
```json5
"userDataDir": false,
```
```

### ### Example JetBrains IDEs Debug Configuration

You can also debug your Nuxt app in JetBrains IDEs such as IntelliJ IDEA, WebStorm, or PhpStorm.

1. Create a new file in your project root directory and name it ``nuxt.run.xml``.
2. Open the ``nuxt.run.xml`` file and paste the following debug configuration:

```
```html
<component name="ProjectRunConfigurationManager">
  <configuration default="false" name="client: chrome" type="JavascriptDebugType" uri="http://localhost:3000"
useFirstLineBreakpoints="true">
    <method v="2" />
  </configuration>

  <configuration default="false" name="server: nuxt" type="NodeJSConfigurationType" application-parameters="dev" path-to-js-
file="$PROJECT_DIR$/node_modules/nuxi/bin/nuxi.mjs" working-dir="$PROJECT_DIR$">
    <method v="2" />
  </configuration>

  <configuration default="false" name="fullstack: nuxt" type="CompoundRunConfigurationType">
    <toRun name="client: chrome" type="JavascriptDebugType" />
    <toRun name="server: nuxt" type="NodeJSConfigurationType" />
    <method v="2" />
  </configuration>
</component>
```
```

### ### Other IDEs

If you have another IDE and would like to contribute sample configuration, feel free to [open a PR] (<https://github.com/nuxt/nuxt/edit/main/docs/2.guide/3.going-further/9.debugging.md>)!

---  
navigation: false  
redirect: /guide/going-further/tooling  
---

```

title: "navigateTo"
description: navigateTo is a helper function that programmatically navigates users.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
 size: xs

```

```
:::note
`navigateTo` is available on both client and server side (but not within Nitro routes).
::
```

## ## Usage

`navigateTo` is available on both server side and client side. It can be used within the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context), or directly, to perform page navigation.

```
:::tip
To send a redirect from a server endpoint, use [`sendRedirect`](https://h3.unjs.io/utls/response#sendredirectevent-location-code) instead.
::
```

## ### Within a Vue Component

```
```vue
<script setup lang="ts">
// passing 'to' as a string
await navigateTo('/search')

// ... or as a route object
await navigateTo({ path: '/search' })

// ... or as a route object with query parameters
await navigateTo({
  path: '/search',
  query: {
    page: 1,
    sort: 'asc'
  }
})
</script>
```
```

## ### Within Route Middleware

```
```ts
export default defineNuxtRouteMiddleware((to, from) => {
  if (to.path !== '/search') {
    // setting the redirect code to '301 Moved Permanently'
    return navigateTo('/search', { redirectCode: 301 })
  }
})
```
```

:read-more{to="/docs/guide/directory-structure/middleware"}

## ### External URL

The `external` parameter in `navigateTo` influences how navigating to URLs is handled:

- **Without `external: true`**:
  - Internal URLs navigate as expected.
  - External URLs throw an error.
- **With `external: true`**:
  - Internal URLs navigate with a full-page reload.
  - External URLs navigate as expected.

## #### Example

```
```vue
<script setup lang="ts">
// will throw an error;
// navigating to an external URL is not allowed by default
await navigateTo('https://nuxt.com')

// will redirect successfully with the 'external' parameter set to 'true'
await navigateTo('https://nuxt.com', {
  external: true
})
</script>
```

```
...
```

```
### Using open()
```

```
```vue
<script setup lang="ts">
// will open 'https://nuxt.com' in a new tab
await navigateTo('https://nuxt.com', {
 open: {
 target: '_blank',
 windowFeatures: {
 width: 500,
 height: 500
 }
 }
})
</script>
```
```

```
## Type
```

```
```ts
navigateTo(to: RouteLocationRaw | undefined | null, options?: NavigateToOptions) => Promise<void | NavigationFailure> |
RouteLocationRaw

interface NavigateToOptions {
 replace?: boolean
 redirectCode?: number
 external?: boolean
 open?: OpenOptions
}
```
```

```
:::warning
Make sure to always use `await` or `return` on result of `navigateTo` when calling it.
:::
```

```
## Parameters
```

```
### `to`

**Type**: [RouteLocationRaw](https://router.vuejs.org/api/interfaces/RouteLocationOptions.html#Interface-RouteLocationOptions) | undefined | null

**Default**: ''/''
```

`to` can be a plain string or a route object to redirect to. When passed as `undefined` or `null`, it will default to `''/''`.

```
### `options` (optional)
```

```
**Type**: NavigateToOptions
```

An object accepting the following properties:

- `replace` (optional)

```
**Type**: boolean
```

```
**Default**: false
```

By default, `navigateTo` pushes the given route into the Vue Router's instance on the client side.

This behavior can be changed by setting `replace` to `true`, to indicate that given route should be replaced.

- `redirectCode` (optional)

```
**Type**: number
```

```
**Default**: 302
```

`navigateTo` redirects to the given path and sets the redirect code to [`302 Found`](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302) by default when the redirection takes place on the server side.

This default behavior can be modified by providing different `redirectCode`. Commonly, [`301 Moved Permanently`](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301) can be used for permanent redirections.

- `external` (optional)

```
**Type**: boolean
```

```
**Default**: false
```

Allows navigating to an external URL when set to `true`. Otherwise, `navigateTo` will throw an error, as external

navigation is not allowed by default.

- ``open`` (optional)

****Type****: ``OpenOptions``

Allows navigating to the URL using the `[open()]`(<https://developer.mozilla.org/en-US/docs/Web/API/Window/open>) method of the window. This option is only applicable on the client side and will be ignored on the server side.

An object accepting the following properties:

- ``target``

****Type****: ``string``

****Default****: ``'_blank'``

A string, without whitespace, specifying the name of the browsing context the resource is being loaded into.

- ``windowFeatures`` (optional)

****Type****: ``OpenWindowFeatures``

An object accepting the following properties:

- ``popup`` (optional)

****Type****: ``boolean``

- ``width`` or ``innerWidth`` (optional)

****Type****: ``number``

- ``height`` or ``innerHeight`` (optional)

****Type****: ``number``

- ``left`` or ``screenX`` (optional)

****Type****: ``number``

- ``top`` or ``screenY`` (optional)

****Type****: ``number``

- ``noopener`` (optional)

****Type****: ``boolean``

- ``noreferrer`` (optional)

****Type****: ``boolean``

Refer to the [\[documentation\]](https://developer.mozilla.org/en-US/docs/Web/API/Window/open)(<https://developer.mozilla.org/en-US/docs/Web/API/Window/open>) for more detailed information on the ****windowFeatures**** properties.

```
---
title: "onBeforeRouteLeave"
description: The onBeforeRouteLeave composable allows registering a route guard within a component.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
    size: xs
---

:read-more{icon="i-simple-icons-vuedotjs" to="https://router.vuejs.org/api/#onBeforeRouteLeave" title="Vue Router Docs"
target="_blank"}
```

```
---
title: "onBeforeRouteUpdate"
description: The onBeforeRouteUpdate composible allows registering a route guard within a component.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
    size: xs
---

:read-more{icon="i-simple-icons-vuedotjs" to="https://router.vuejs.org/api/#onBeforeRouteUpdate" title="Vue Router Docs"
target="_blank"}
```



```

---
title: 'useNuxtData'
description: 'Access the current cached value of data fetching composables.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
    size: xs
---

::note
`useNuxtData` gives you access to the current cached value of [`useAsyncData`](/docs/api/composables/use-async-data) ,
`useLazyAsyncData`, [`useFetch`](/docs/api/composables/use-fetch) and [`useLazyFetch`](/docs/api/composables/use-lazy-fetch)
with explicitly provided key.
::

```

Usage

The example below shows how you can use cached data as a placeholder while the most recent data is being fetched from the server.

```

```vue [pages/posts.vue]
<script setup lang="ts">
// We can access same data later using 'posts' key
const { data } = await useFetch('/api/posts', { key: 'posts' })
</script>
```

```vue [pages/posts/[id\].vue]
<script setup lang="ts">
// Access to the cached value of useFetch in posts.vue (parent route)
const { id } = useRoute().params
const { data: posts } = useNuxtData('posts')
const { data } = useLazyFetch(`/api/posts/${id}`, {
 key: `post-${id}`,
 default() {
 // Find the individual post from the cache and set it as the default value.
 return posts.value.find(post => post.id === id)
 }
})
</script>
```

```

Optimistic Updates

We can leverage the cache to update the UI after a mutation, while the data is being invalidated in the background.

```

```vue [pages/todos.vue]
<script setup lang="ts">
// We can access same data later using 'todos' key
const { data } = await useAsyncData('todos', () => $fetch('/api/todos'))
</script>
```

```vue [components/NewTodo.vue]
<script setup lang="ts">
const newTodo = ref('')
const previousTodos = ref([])

// Access to the cached value of useFetch in todos.vue
const { data: todos } = useNuxtData('todos')

const { data } = await useFetch('/api/addTodo', {
 method: 'post',
 body: {
 todo: newTodo.value
 },
 onRequest () {
 previousTodos.value = todos.value // Store the previously cached value to restore if fetch fails.

 todos.value.push(newTodo.value) // Optimistically update the todos.
 },
 onRequestError () {
 todos.value = previousTodos.value // Rollback the data if the request failed.
 },
 async onResponse () {
 await refreshNuxtData('todos') // Invalidate todos in the background if the request succeeded.
 }
})
</script>
```

```

Type

```
``ts
useNuxtData<DataT = any> (key: string): { data: Ref<DataT | null> }
``
```

```
---
title: 'utils'
head.title: 'utils/'
description: Use the utils/ directory to auto-import your utility functions throughout your application.
navigation.icon: i-ph-folder-duotone
---
```

The main purpose of the [``utils/`` directory](/docs/guide/directory-structure/utils) is to allow a semantic distinction between your Vue composables and other auto-imported utility functions.

Usage

Method 1: Using named export

```
``ts twoslash [utils/index.ts]
export const { format: formatNumber } = Intl.NumberFormat('en-GB', {
  notation: 'compact',
  maximumFractionDigits: 1
})
```
```

**Method 2:** Using default export

```
``ts twoslash [utils/random-entry.ts or utils/randomEntry.ts]
// It will be available as randomEntry() (camelCase of file name without extension)
export default function (arr: Array<any>) {
 return arr[Math.floor(Math.random() * arr.length)]
}
```
```

You can now use auto imported utility functions in `.js`, `.ts` and `.vue` files

```
``vue [app.vue]
<template>
  <p>{{ formatNumber(1234) }}</p>
</template>
```
```

`:read-more{to="/docs/guide/concepts/auto-imports"}`

`:link-example{to="/docs/examples/features/auto-imports"}`

`::tip`

The way ``utils/`` auto-imports work and are scanned is identical to the [``composables/``](/docs/guide/directory-structure/composables) directory.

`::`

`::important`

These utils are only available within the Vue part of your app. `:br`

Only ``server/utils`` are auto-imported in the [``server/``](/docs/guide/directory-structure/server#server-utilities) directory.

`::`

```

title: 'State Management'
description: Nuxt provides powerful state management libraries and the useState composable to create a reactive and SSR-friendly shared state.
navigation.icon: i-ph-database-duotone

```

Nuxt provides the `useState` (</docs/api/composables/use-state>) composable to create a reactive and SSR-friendly shared state across components.

`useState` (</docs/api/composables/use-state>) is an SSR-friendly `ref` (<https://vuejs.org/api/reactivity-core.html#ref>) replacement. Its value will be preserved after server-side rendering (during client-side hydration) and shared across all components using a unique key.

`::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=mv0WcBABcIk" target="_blank"}`  
Watch a video from Alexander Lichter about why and when to use `useState`.  
`::`

`::important`  
Because the data inside `useState` (</docs/api/composables/use-state>) will be serialized to JSON, it is important that it does not contain anything that cannot be serialized, such as classes, functions or symbols.  
`::`

`::read-more{to="/docs/api/composables/use-state"}`  
Read more about `useState` composable.  
`::`

## ## Best Practices

`::warning`  
Never define `const state = ref()` outside of `<script setup>` or `setup()` function.  
For example, doing `export myState = ref({})` would result in state shared across requests on the server and can lead to memory leaks.  
`::`

`::tip{icon="i-ph-check-circle-duotone"}`  
Instead use `const useX = () => useState('x')`  
`::`

## ## Examples

### ### Basic Usage

In this example, we use a component-local counter state. Any other component that uses `useState('counter')` shares the same reactive state.

```
```:vue twoslash [app.vue]
<script setup lang="ts">
const counter = useState('counter', () => Math.round(Math.random() * 1000))
</script>
```

```
<template>
  <div>
    Counter: {{ counter }}
    <button @click="counter++">
      +
    </button>
    <button @click="counter--">
      -
    </button>
  </div>
</template>
```:
```

`:link-example{to="/docs/examples/features/state-management"}`

`::note`  
To globally invalidate cached state, see `clearNuxtState` (</docs/api/utils/clear-nuxt-state>) util.  
`::`

### ### Initializing State

Most of the time, you will want to initialize your state with data that resolves asynchronously. You can use the `app.vue` (</docs/guide/directory-structure/app>) component with the `callOnce` (</docs/api/utils/call-once>) util to do so.

```
```:vue twoslash [app.vue]
<script setup lang="ts">
const websiteConfig = useState('config')

await callOnce(async () => {
  websiteConfig.value = await $fetch('https://my-cms.com/api/website-config')
})
</script>
```

```
...
```

```
:::tip
```

This is similar to the [`nuxtServerInit` action`](https://v2.nuxt.com/docs/directory-structure/store/#the-nuxtserverinit-action) in Nuxt 2, which allows filling the initial state of your store server-side before rendering the page.

```
:::
```

```
:read-more{to="/docs/api/utils/call-once"}
```

Usage with Pinia

In this example, we leverage the [Pinia module](/modules/pinia) to create a global store and use it across the app.

```
:::important
```

Make sure to install the Pinia module with ``npx nuxi@latest module add pinia`` or follow the [module's installation steps](https://pinia.vuejs.org/ssr/nuxt.html#Installation).

```
:::
```

```
:::code-group
```

```
```ts [stores/website.ts]
export const useWebsiteStore = defineStore('websiteStore', {
 state: () => ({
 name: '',
 description: ''
 }),
 actions: {
 async fetch() {
 const infos = await $fetch('https://api.nuxt.com/modules/pinia')

 this.name = infos.name
 this.description = infos.description
 }
 }
})
```
```vue [app.vue]
<script setup lang="ts">
const website = useWebsiteStore()

await callOnce(website.fetch)
</script>
```

```
<template>
 <main>
 <h1>{{ website.name }}</h1>
 <p>{{ website.description }}</p>
 </main>
</template>
```
```

```
:::
```

Advanced Usage

```
:::code-group
```

```
```ts [composables/locale.ts]
import type { Ref } from 'vue'

export const useLocale = () => {
 return useState<string>('locale', () => useDefaultLocale().value)
}

export const useDefaultLocale = (fallback = 'en-US') => {
 const locale = ref(fallback)
 if (import.meta.server) {
 const reqLocale = useRequestHeaders()['accept-language']?.split(',')[0]
 if (reqLocale) {
 locale.value = reqLocale
 }
 } else if (import.meta.client) {
 const navLang = navigator.language
 if (navLang) {
 locale.value = navLang
 }
 }
 return locale
}
```

```
export const useLocales = () => {
 const locale = useLocale()
 const locales = ref([
 'en-US',
 'en-GB',
 ...
```

```

 'ja-JP-u-ca-japanese'
])
 if (!locales.value.includes(locale.value)) {
 locales.value.unshift(locale.value)
 }
 return locales
}

export const useLocaleDate = (date: Ref<Date> | Date, locale = useLocale()) => {
 return computed(() => new Intl.DateTimeFormat(locale.value, { dateStyle: 'full' }).format(unref(date)))
}

```

```

```vue [app.vue]
<script setup lang="ts">
const locales = useLocales()
const locale = useLocale()
const date = useLocaleDate(new Date('2016-10-26'))
</script>

<template>
  <div>
    <h1>Nuxt birthday</h1>
    <p>{{ date }}</p>
    <label for="locale-chooser">Preview a different locale</label>
    <select id="locale-chooser" v-model="locale">
      <option v-for="locale of locales" :key="locale" :value="locale">
        {{ locale }}
      </option>
    </select>
  </div>
</template>
```

```

```

::

:link-example{to="/docs/examples/advanced/locale"}

```

## ## Shared State

By using [auto-imported composables](/docs/guide/directory-structure/composables) we can define global type-safe states and import them across the app.

```

```ts twoslash [composables/states.ts]
export const useColor = () => useState<string>('color', () => 'pink')
```

```

```

```vue [app.vue]
<script setup lang="ts">
// ---cut-start---
const useColor = () => useState<string>('color', () => 'pink')
// ---cut-end---
const color = useColor() // Same as useState('color')
</script>

```

```

<template>
  <p>Current color: {{ color }}</p>
</template>
```

```

```

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=dZSNW07s0-A" target="_blank"}
Watch a video from Daniel Roe on how to deal with global state and SSR in Nuxt.
::

```

## ## Using third-party libraries

Nuxt **used to rely** on the Vuex library to provide global state management. If you are migrating from Nuxt 2, please head to [the migration guide](/docs/migration/configuration#vuex).

Nuxt is not opinionated about state management, so feel free to choose the right solution for your needs. There are multiple integrations with the most popular state management libraries, including:

- [Pinia](/modules/pinia) - the official Vue recommendation
- [Harlem](/modules/harlem) - immutable global state management
- [XState](/modules/xstate) - state machine approach with tools for visualizing and testing your state logic

```

title: server
head.title: 'server/'
description: The server/ directory is used to register API and server handlers to your application.
navigation.icon: i-ph-folder-duotone

```

Nuxt automatically scans files inside these directories to register API and server handlers with Hot Module Replacement (HMR) support.

```

```bash [Directory structure]
-| server/
---| api/
----| hello.ts      # /api/hello
---| routes/
----| bonjour.ts    # /bonjour
---| middleware/
----| log.ts        # log all requests
```

```

Each file should export a default function defined with `defineEventHandler()` or `eventHandler()` (alias).

The handler can directly return JSON data, a `Promise`, or use `event.node.res.end()` to send a response.

```

```ts twoslash [server/api/hello.ts]
export default defineEventHandler((event) => {
  return {
    hello: 'world'
  }
})
```

```

You can now universally call this API in your pages and components:

```

```vue [pages/index.vue]
<script setup lang="ts">
const { data } = await useFetch('/api/hello')
</script>

<template>
  <pre>{{ data }}</pre>
</template>
```

```

## ## Server Routes

Files inside the `~/server/api` are automatically prefixed with `/api` in their route.

To add server routes without `/api` prefix, put them into `~/server/routes` directory.

**Example:**

```

```ts [server/routes/hello.ts]
export default defineEventHandler(() => 'Hello World!')
```

```

Given the example above, the `/hello` route will be accessible at `<http://localhost:3000/hello>`.

**note**  
 Note that currently server routes do not support the full functionality of dynamic routes as `[pages] (/docs/guide/directory-structure/pages#dynamic-routes)` do.  
**note**

## ## Server Middleware

Nuxt will automatically read in any file in the `~/server/middleware` to create server middleware for your project.

Middleware handlers will run on every request before any other server route to add or check headers, log requests, or extend the event's request object.

**note**  
 Middleware handlers should not return anything (nor close or respond to the request) and only inspect or extend the request context or throw an error.  
**note**

**Examples:**

```

```ts [server/middleware/log.ts]
export default defineEventHandler((event) => {
  console.log('New request: ' + getRequestURL(event))
})
```

```

```
``ts [server/middleware/auth.ts]
export default defineEventHandler((event) => {
 event.context.auth = { user: 123 }
})
```
```

Server Plugins

Nuxt will automatically read any files in the `~/server/plugins` directory and register them as Nitro plugins. This allows extending Nitro's runtime behavior and hooking into lifecycle events.

****Example:****

```
``ts [server/plugins/nitroPlugin.ts]
export default defineNitroPlugin((nitroApp) => {
  console.log('Nitro plugin', nitroApp)
})
```
```

```
:read-more{to="https://nitro.unjs.io/guide/plugins" title="Nitro Plugins" target="_blank"}
```

## ## Server Utilities

Server routes are powered by [unjs/h3](https://github.com/unjs/h3) which comes with a handy set of helpers.

```
:read-more{to="https://www.jsdocs.io/package/h3#package-index-functions" title="Available H3 Request Helpers" target="_blank"}
```

You can add more helpers yourself inside the `~/server/utls` directory.

For example, you can define a custom handler utility that wraps the original handler and performs additional operations before returning the final response.

**\*\*Example:\*\***

```
``ts [server/utls/handler.ts]
import type { EventHandler, EventHandlerRequest } from 'h3'

export const defineWrappedResponseHandler = <T extends EventHandlerRequest, D> (
 handler: EventHandler<T, D>
): EventHandler<T, D> => {
 defineEventHandler<T>(async event => {
 try {
 // do something before the route handler
 const response = await handler(event)
 // do something after the route handler
 return { response }
 } catch (err) {
 // Error handling
 return { err }
 }
 })
})
```
```

Server Types

```
::tip
This feature is available from Nuxt >= 3.5
::
```

To improve clarity within your IDE between the auto-imports from 'nitro' and 'vue', you can add a `~/server/tsconfig.json` with the following content:

```
``json [server/tsconfig.json]
{
  "extends": "../.nuxt/tsconfig.server.json"
}
```
```

Currently, these values won't be respected when type checking ([`nuxi typecheck`](/docs/api/commands/typecheck)), but you should get better type hints in your IDE.

## ## Recipes

### ### Route Parameters

Server routes can use dynamic parameters within brackets in the file name like `~/api/hello/[name].ts` and be accessed via `event.context.params`.

```
``ts [server/api/hello/[name].ts]
export default defineEventHandler((event) => {
 const name = getRouterParam(event, 'name')
})
```



```

 return `Hello, ${name}!`
 })
 ...

::tip
Alternatively, use `getValidatedRouterParams` with a schema validator such as Zod for runtime and type safety.
::

```

You can now universally call this API on `/api/hello/nuxt` and get `Hello, nuxt!`.

### ### Matching HTTP Method

Handle file names can be suffixed with `.get`, `.post`, `.put`, `.delete`, ... to match request's [HTTP Method] (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>).

```

```ts [server/api/test.get.ts]
export default defineEventHandler(() => 'Test get handler')
```

```ts [server/api/test.post.ts]
export default defineEventHandler(() => 'Test post handler')
```

```

Given the example above, fetching `/test` with:

- **GET** method: Returns `Test get handler`
- **POST** method: Returns `Test post handler`
- Any other method: Returns 405 error

You can also use `index.[method].ts` inside a directory for structuring your code differently, this is useful to create API namespaces.

```

::code-group
```ts [server/api/foo/index.get.ts]
export default defineEventHandler((event) => {
  // handle GET requests for the `api/foo` endpoint
})
```

```ts [server/api/foo/index.post.ts]
export default defineEventHandler((event) => {
  // handle POST requests for the `api/foo` endpoint
})
```

```ts [server/api/foo/bar.get.ts]
export default defineEventHandler((event) => {
  // handle GET requests for the `api/foo/bar` endpoint
})
```
::

```

### ### Catch-all Route

Catch-all routes are helpful for fallback route handling.

For example, creating a file named `~/server/api/foo/[...].ts` will register a catch-all route for all requests that do not match any route handler, such as `/api/foo/bar/baz`.

```

```ts [server/api/foo/[...].ts]
export default defineEventHandler((event) => {
  // event.context.path to get the route path: '/api/foo/bar/baz'
  // event.context.params._ to get the route segment: 'bar/baz'
  return `Default foo handler`
})
```

```

You can set a name for the catch-all route by using `~/server/api/foo/[...slug].ts` and access it via `event.context.params.slug`.

```

```ts [server/api/foo/[...slug].ts]
export default defineEventHandler((event) => {
  // event.context.params.slug to get the route segment: 'bar/baz'
  return `Default foo handler`
})
```

```

### ### Body Handling

```

```ts [server/api/submit.post.ts]
export default defineEventHandler(async (event) => {
  const body = await readBody(event)
  return { body }
})
```

```

```
::tip{to="https://unjs.io/blog/2023-08-15-h3-towards-the-edge-of-the-web#runtime-type-safe-request-utils"}
Alternatively, use `readValidatedBody` with a schema validator such as Zod for runtime and type safety.
::
```

You can now universally call this API using:

```
``vue [app.vue]
<script setup lang="ts">
async function submit() {
 const { body } = await $fetch('/api/submit', {
 method: 'post',
 body: { test: 123 }
 })
}
</script>
``
```

```
::note
We are using `submit.post.ts` in the filename only to match requests with `POST` method that can accept the request body.
When using `readBody` within a GET request, `readBody` will throw a `405 Method Not Allowed` HTTP error.
::
```

### ### Query Parameters

Sample query `/api/query?foo=bar&baz=qux`

```
``ts [server/api/query.get.ts]
export default defineEventHandler((event) => {
 const query = getQuery(event)

 return { a: query.foo, b: query.baz }
})
``
```

```
::tip{to="https://unjs.io/blog/2023-08-15-h3-towards-the-edge-of-the-web#runtime-type-safe-request-utils"}
Alternatively, use `getValidatedQuery` with a schema validator such as Zod for runtime and type safety.
::
```

### ### Error Handling

If no errors are thrown, a status code of `200 OK` will be returned.

Any uncaught errors will return a `500 Internal Server Error` HTTP Error.

To return other error codes, throw an exception with [`createError`](/docs/api/utils/create-error):

```
``ts [server/api/validation/[id\\.].ts]
export default defineEventHandler((event) => {
 const id = parseInt(event.context.params.id) as number

 if (!Number.isInteger(id)) {
 throw createError({
 statusCode: 400,
 statusMessage: 'ID should be an integer',
 })
 }
 return 'All good'
})
``
```

### ### Status Codes

To return other status codes, use the [`setResponseStatus`](/docs/api/utils/set-response-status) utility.

For example, to return `202 Accepted`

```
``ts [server/api/validation/[id\\.].ts]
export default defineEventHandler((event) => {
 setResponseStatus(event, 202)
})
``
```

### ### Runtime Config

```
::code-group
``ts [server/api/foo.ts]
export default defineEventHandler(async (event) => {
 const config = useRuntimeConfig(event)

 const repo = await $fetch('https://api.github.com/repos/nuxt/nuxt', {
 headers: {
 Authorization: `token ${config.githubToken}`
 }
 })
})
``
```

```

 }
 })

 return repo
})
```
```ts [nuxt.config.ts]
export default defineNuxtConfig({
 runtimeConfig: {
 githubToken: ''
 }
})
```
```bash [.env]
NUXT_GITHUB_TOKEN='<my-super-token>'
```
::

```

::note
 Giving the `event` as argument to `useRuntimeConfig` is optional, but it is recommended to pass it to get the runtime config overwritten by [environment variables](/docs/guide/going-further/runtime-config#environment-variables) at runtime for server routes.

::

Request Cookies

```

```ts [server/api/cookies.ts]
export default defineEventHandler((event) => {
 const cookies = parseCookies(event)

 return { cookies }
})
```

```

Advanced Usage

Nitro Config

You can use `nitro` key in `nuxt.config` to directly set [Nitro configuration](https://nitro.unjs.io/config).

::warning
 This is an advanced option. Custom config can affect production deployments, as the configuration interface might change over time when Nitro is upgraded in semver-minor versions of Nuxt.

::

```

```ts [nuxt.config.ts]
export default defineNuxtConfig({
 // https://nitro.unjs.io/config
 nitro: {}
})
```

```

:read-more{to="/docs/guide/concepts/server-engine"}

Nested Router

```

```ts [server/api/hello/[...slug\].ts]
import { createRouter, defineEventHandler, useBase } from 'h3'

const router = createRouter()

router.get('/test', defineEventHandler(() => 'Hello World'))

export default useBase('/api/hello', router.handler)
```

```

Sending Streams

::tip
 This is an experimental feature and is available in all environments.

::

```

```ts [server/api/foo.get.ts]
import fs from 'node:fs'
import { sendStream } from 'h3'

export default defineEventHandler((event) => {
 return sendStream(event, fs.createReadStream('/path/to/file'))
})
```

```

Sending Redirect

```

```ts [server/api/foo.get.ts]
export default defineEventHandler(async (event) => {
 await sendRedirect(event, '/path/redirect/to', 302)
})
```

```

Legacy Handler or Middleware

```

```ts [server/api/legacy.ts]
export default fromNodeMiddleware((req, res) => {
 res.end('Legacy handler')
})
```

```

::important

Legacy support is possible using [unjs/h3](https://github.com/unjs/h3), but it is advised to avoid legacy handlers as much as you can.

::

```

```ts [server/middleware/legacy.ts]
export default fromNodeMiddleware((req, res, next) => {
 console.log('Legacy middleware')
 next()
})
```

```

::warning

Never combine `next()` callback with a legacy middleware that is `async` or returns a `Promise`.

::

Server Storage

Nitro provides a cross-platform [storage layer](https://nitro.unjs.io/guide/storage). In order to configure additional storage mount points, you can use `nitro.storage`, or [server plugins](#server-plugins).

Example of adding a Redis storage:

Using `nitro.storage`:

```

```ts [nuxt.config.ts]
export default defineNuxtConfig({
 nitro: {
 storage: {
 redis: {
 driver: 'redis',
 /* redis connector options */
 port: 6379, // Redis port
 host: "127.0.0.1", // Redis host
 username: "", // needs Redis >= 6
 password: "",
 db: 0, // Defaults to 0
 tls: {} // tls/ssl
 }
 }
 }
})
```

```

Then in your API handler:

```

```ts [server/api/storage/test.ts]
export default defineEventHandler(async (event) => {
 // List all keys with
 const keys = await useStorage('redis').getKeys()

 // Set a key with
 await useStorage('redis').setItem('foo', 'bar')

 // Remove a key with
 await useStorage('redis').removeItem('foo')

 return {}
})
```

```

::read-more{to="https://nitro.unjs.io/guide/storage" target="_blank"}

Read more about Nitro Storage Layer.

::

Alternatively, you can create a storage mount point using a server plugin and runtime config:

::code-group

```

```ts [server/plugins/storage.ts]

```

```

import redisDriver from 'unstorage/drivers/redis'

export default defineNitroPlugin(() => {
 const storage = useStorage()

 // Dynamically pass in credentials from runtime configuration, or other sources
 const driver = redisDriver({
 base: 'redis',
 host: useRuntimeConfig().redis.host,
 port: useRuntimeConfig().redis.port,
 /* other redis connector options */
 })

 // Mount driver
 storage.mount('redis', driver)
})
```


```

``` ts [nuxt.config.ts]
export default defineNuxtConfig({
  runtimeConfig: {
    redis: { // Default values
      host: '',
      port: 0,
      /* other redis connector options */
    }
  }
})
```
::

```


```

```
---
title: 'Data fetching'
description: Nuxt provides composables to handle data fetching within your application.
navigation.icon: i-ph-plugs-connected-duotone
---
```

Nuxt comes with two composables and a built-in library to perform data-fetching in browser or server environments: `useFetch`, `useAsyncData` (</docs/api/composables/use-async-data>) and `$fetch`.

In a nutshell:

- `useFetch` (</docs/api/composables/use-fetch>) is the most straightforward way to handle data fetching in a component setup function.
- `$fetch` (</docs/api/utis/dollarfetch>) is great to make network requests based on user interaction.
- `useAsyncData` (</docs/api/composables/use-async-data>), combined with `$fetch`, offers more fine-grained control.

Both `useFetch` and `useAsyncData` share a common set of options and patterns that we will detail in the last sections.

Before that, it's imperative to know why these composables exist in the first place.

Why use specific composables for data fetching?

Nuxt is a framework which can run isomorphic (or universal) code in both server and client environments. If the `$fetch` function (</docs/api/utis/dollarfetch>) is used to perform data fetching in the setup function of a Vue component, this may cause data to be fetched twice, once on the server (to render the HTML) and once again on the client (when the HTML is hydrated). This is why Nuxt offers specific data fetching composables so data is fetched only once.

Network calls duplication

The `useFetch` (</docs/api/composables/use-fetch>) and `useAsyncData` (</docs/api/composables/use-async-data>) composables ensure that once an API call is made on the server, the data is properly forwarded to the client in the payload.

The payload is a JavaScript object accessible through `useNuxtApp().payload` (</docs/api/composables/use-nuxt-app#payload>). It is used on the client to avoid refetching the same data when the code is executed in the browser [during hydration] (</docs/guide/concepts/rendering#universal-rendering>).

```
::tip
Use the [Nuxt DevTools](https://devtools.nuxt.com) to inspect this data in the Payload tab.
::
```

Suspense

Nuxt uses Vue's `<Suspense>` (<https://vuejs.org/guide/built-ins/suspense>) component under the hood to prevent navigation before every async data is available to the view. The data fetching composables can help you leverage this feature and use what suits best on a per-calls basis.

```
::note
You can add the <NuxtLoadingIndicator> (/docs/api/components/nuxt-loading-indicator) to add a progress bar between page navigations.
::
```

useFetch

The `useFetch` (</docs/api/composables/use-fetch>) composable is the most straightforward way to perform data fetching.

```
``vue twoslash [app.vue]
<script setup lang="ts">
const { data: count } = await useFetch('/api/count')
</script>

<template>
  <p>Page visits: {{ count }}</p>
</template>
``
```

This composable is a wrapper around the `useAsyncData` (</docs/api/composables/use-async-data>) composable and `$fetch` utility.

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=njsGVmcWviY" target="_blank"}
Watch the video from Alexander Lichter to avoid using useFetch the wrong way!
::
```

```
:read-more{to="/docs/api/composables/use-fetch"}
```

```
:link-example{to="/docs/examples/features/data-fetching"}
```

\$fetch

Nuxt includes the `ofetch` (<https://github.com/unjs/ofetch>) library, and is auto-imported as the `$fetch` alias globally across your application. It's what `useFetch` uses behind the scenes.

```
``vue twoslash [pages/todos.vue]
<script setup lang="ts">
```

```

async function addTodo() {
  const todo = await $fetch('/api/todos', {
    method: 'POST',
    body: {
      // My todo data
    }
  })
}
</script>
`

```

::warning
 Beware that using only `$fetch` will not provide [network calls de-duplication and navigation prevention](#why-use-specific-composables-for-data-fetching). :br
 It is recommended to use `$fetch` for client-side interactions (event based) or combined with `useAsyncData` (#useasyncdata) when fetching the initial component data.
::

::read-more{to="/docs/api/utils/dollarfetch"}
 Read more about `$fetch`.
::

`useAsyncData`

The `useAsyncData` composable is responsible for wrapping async logic and returning the result once it is resolved.

::tip
`useFetch(url)` is nearly equivalent to `useAsyncData(url, () => $fetch(url))`. :br
 It's developer experience sugar for the most common use case.
::

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=0X-a0pSGabA" target="_blank"}
 Watch a video from Alexander Lichter to dig deeper into the difference between `useFetch` and `useAsyncData`.
::

There are some cases when using the `useFetch` (/docs/api/composables/use-fetch) composable is not appropriate, for example when a CMS or a third-party provide their own query layer. In this case, you can use `useAsyncData` (/docs/api/composables/use-async-data) to wrap your calls and still keep the benefits provided by the composable.

```

`vue [pages/users.vue]
<script setup lang="ts">
const { data, error } = await useAsyncData('users', () => myGetFunction('users'))

// This is also possible:
const { data, error } = await useAsyncData(() => myGetFunction('users'))
</script>
`

```

::note
 The first argument of `useAsyncData` (/docs/api/composables/use-async-data) is a unique key used to cache the response of the second argument, the querying function. This key can be ignored by directly passing the querying function, the key will be auto-generated.
 :br :br
 Since the autogenerated key only takes into account the file and line where `useAsyncData` is invoked, it is recommended to always create your own key to avoid unwanted behavior, like when you are creating your own custom composable wrapping `useAsyncData`.
 :br :br
 Setting a key can be useful to share the same data between components using `useNuxtData` (/docs/api/composables/use-nuxt-data) or to [refresh specific data](/docs/api/utils/refresh-nuxt-data#refresh-specific-data).
::

```

`vue [pages/users/[id].vue]
<script setup lang="ts">
const { id } = useRoute().params

const { data, error } = await useAsyncData(`user:${id}`, () => {
  return myGetFunction('users', { id })
})
</script>
`

```

The `useAsyncData` composable is a great way to wrap and wait for multiple `$fetch` requests to be completed, and then process the results.

```

`vue
<script setup lang="ts">
const { data: discounts, status } = await useAsyncData('cart-discount', async () => {
  const [coupons, offers] = await Promise.all([
    $fetch('/cart/coupons'),
    $fetch('/cart/offers')
  ])

  return { coupons, offers }
})

```

```

})
// discounts.value.coupons
// discounts.value.offers
</script>
```

```

```

::read-more{to="/docs/api/composables/use-async-data"}
Read more about `useAsyncData`.
::

```

## ## Return Values

`useFetch` and `useAsyncData` have the same return values listed below.

- `data`: the result of the asynchronous function that is passed in.
- `refresh`/`execute`: a function that can be used to refresh the data returned by the `handler` function.
- `clear`: a function that can be used to set `data` to `undefined`, set `error` to `null`, set `status` to `idle`, and mark any currently pending requests as cancelled.
- `error`: an error object if the data fetching failed.
- `status`: a string indicating the status of the data request (`"idle"`, `"pending"`, `"success"`, `"error"`).

```

::note
`data`, `error` and `status` are Vue refs accessible with `.value` in `
```



By default, data fetching composables will perform their asynchronous function on both client and server environments. Set the ``server`` option to ``false`` to only perform the call on the client-side. On initial load, the data will not be fetched before hydration is complete so you have to handle a pending state, though on subsequent client-side navigation the data will be awaited before loading the page.

Combined with the ``lazy`` option, this can be useful for data that is not needed on the first render (for example, non-SEO sensitive data).

```
````ts twoslash
/* This call is performed before hydration */
const articles = await useFetch('/api/article')

/* This call will only be performed on the client */
const { status, data: comments } = useFetch('/api/comments', {
  lazy: true,
  server: false
})
```
```

The ``useFetch`` composable is meant to be invoked in setup method or called directly at the top level of a function in lifecycle hooks, otherwise you should use [``$fetch`` method](#fetch).

### ### Minimize payload size

The ``pick`` option helps you to minimize the payload size stored in your HTML document by only selecting the fields that you want returned from the composables.

```
````vue
<script setup lang="ts">
/* only pick the fields used in your template */
const { data: mountain } = await useFetch('/api/mountains/everest', {
  pick: ['title', 'description']
})
</script>

<template>
  <h1>{{ mountain.title }}</h1>
  <p>{{ mountain.description }}</p>
</template>
```
```

If you need more control or map over several objects, you can use the ``transform`` function to alter the result of the query.

```
````ts
const { data: mountains } = await useFetch('/api/mountains', {
  transform: (mountains) => {
    return mountains.map(mountain => ({ title: mountain.title, description: mountain.description }))
  }
})
```
```

:::note

Both ``pick`` and ``transform`` don't prevent the unwanted data from being fetched initially. But they will prevent unwanted data from being added to the payload transferred from server to client.

:::

### ### Caching and refetching

#### #### Keys

[``useFetch``](/docs/api/composables/use-fetch) and [``useAsyncData``](/docs/api/composables/use-async-data) use keys to prevent refetching the same data.

- [``useFetch``](/docs/api/composables/use-fetch) uses the provided URL as a key. Alternatively, a ``key`` value can be provided in the ``options`` object passed as a last argument.

- [``useAsyncData``](/docs/api/composables/use-async-data) uses its first argument as a key if it is a string. If the first argument is the handler function that performs the query, then a key that is unique to the file name and line number of the instance of ``useAsyncData`` will be generated for you.

:::tip

To get the cached data by key, you can use [``useNuxtData``](/docs/api/composables/use-nuxt-data)

:::

#### #### Refresh and execute

If you want to fetch or refresh data manually, use the ``execute`` or ``refresh`` function provided by the composables.

```
````vue twoslash
<script setup lang="ts">
const { data, error, execute, refresh } = await useFetch('/api/users')
</script>

<template>
```

```

<div>
  <p>{{ data }}</p>
  <button @click="() => refresh()">Refresh data</button>
</div>
</template>
```

```

The ``execute`` function is an alias for ``refresh`` that works in exactly the same way but is more semantic for cases when the fetch is [not immediate](#not-immediate).

```

::tip
To globally refetch or invalidate cached data, see [`clearNuxtData`](/docs/api/utils/clear-nuxt-data) and [`refreshNuxtData`](/docs/api/utils/refresh-nuxt-data).
::

```

#### #### Clear

If you want to clear the data provided, for whatever reason, without needing to know the specific key to pass to ``clearNuxtData``, you can use the ``clear`` function provided by the composables.

```

```vue twoslash
<script setup lang="ts">
const { data, clear } = await useFetch('/api/users')

const route = useRoute()
watch(() => route.path, (path) => {
  if (path === '/') clear()
})
</script>
```

```

#### #### Watch

To re-run your fetching function each time other reactive values in your application change, use the ``watch`` option. You can use it for one or multiple `_watchable_` elements.

```

```vue twoslash
<script setup lang="ts">
const id = ref(1)

const { data, error, refresh } = await useFetch('/api/users', {
  /* Changing the id will trigger a refetch */
  watch: [id]
})
</script>
```

```

Note that **watching a reactive value won't change the URL fetched**. For example, this will keep fetching the same initial ID of the user because the URL is constructed at the moment the function is invoked.

```

```vue
<script setup lang="ts">
const id = ref(1)

const { data, error, refresh } = await useFetch(`/api/users/${id.value}`, {
  watch: [id]
})
</script>
```

```

If you need to change the URL based on a reactive value, you may want to use a [computed URL](#computed-url) instead.

#### #### Computed URL

Sometimes you may need to compute an URL from reactive values, and refresh the data each time these change. Instead of juggling your way around, you can attach each param as a reactive value. Nuxt will automatically use the reactive value and re-fetch each time it changes.

```

```vue
<script setup lang="ts">
const id = ref(null)

const { data, status } = useLazyFetch('/api/user', {
  query: {
    user_id: id
  }
})
</script>
```

```

In the case of more complex URL construction, you may use a callback as a [computed getter](https://vuejs.org/guide/essentials/computed.html) that returns the URL string.

Every time a dependency changes, the data will be fetched using the newly constructed URL. Combine this with [not-immediate] (#not-immediate), and you can wait until the reactive element changes before fetching.

```
```vue
<script setup lang="ts">
const id = ref(null)

const { data, status } = useLazyFetch(() => `/api/users/${id.value}`, {
  immediate: false
})

const pending = computed(() => status.value === 'pending');
</script>

<template>
  <div>
    <!-- disable the input while fetching -->
    <input v-model="id" type="number" :disabled="pending"/>

    <div v-if="status === 'idle'">
      Type an user ID
    </div>

    <div v-else-if="pending">
      Loading ...
    </div>

    <div v-else>
      {{ data }}
    </div>
  </div>
</template>
```
```

If you need to force a refresh when other reactive values change, you can also [watch other values] (#watch).

### ### Not immediate

The `useFetch` composable will start fetching data the moment is invoked. You may prevent this by setting `immediate: false`, for example, to wait for user interaction.

With that, you will need both the `status` to handle the fetch lifecycle, and `execute` to start the data fetch.

```
```vue
<script setup lang="ts">
const { data, error, execute, status } = await useLazyFetch('/api/comments', {
  immediate: false
})
</script>

<template>
  <div v-if="status === 'idle'">
    <button @click="execute">Get data</button>
  </div>

  <div v-else-if="status === 'pending'">
    Loading comments...
  </div>

  <div v-else>
    {{ data }}
  </div>
</template>
```
```

For finer control, the `status` variable can be:

- `idle` when the fetch hasn't started
- `pending` when a fetch has started but not yet completed
- `error` when the fetch fails
- `success` when the fetch is completed successfully

### ## Passing Headers and cookies

When we call `\$fetch` in the browser, user headers like `cookie` will be directly sent to the API. But during server-side-rendering, since the `\$fetch` request takes place 'internally' within the server, it doesn't include the user's browser cookies, nor does it pass on cookies from the fetch response.

### ### Pass Client Headers to the API

We can use [`useRequestHeaders`](/docs/api/composables/use-request-headers) to access and proxy cookies to the API from server-side.

The example below adds the request headers to an isomorphic ``$fetch`` call to ensure that the API endpoint has access to the same ``cookie`` header originally sent by the user.

```
``vue
<script setup lang="ts">
const headers = useRequestHeaders(['cookie'])

const { data } = await useFetch('/api/me', { headers })
</script>
``
```

**::caution**  
Be very careful before proxying headers to an external API and just include headers that you need. Not all headers are safe to be bypassed and might introduce unwanted behavior. Here is a list of common headers that are NOT to be proxied:

```
- `host`, `accept`
- `content-length`, `content-md5`, `content-type`
- `x-forwarded-host`, `x-forwarded-port`, `x-forwarded-proto`
- `cf-connecting-ip`, `cf-ray`
::
```

### ### Pass Cookies From Server-side API Calls on SSR Response

If you want to pass on/proxy cookies in the other direction, from an internal request back to the client, you will need to handle this yourself.

```
``ts [composables/fetch.ts]
import { appendResponseHeader } from 'h3'
import type { H3Event } from 'h3'

export const fetchWithCookie = async (event: H3Event, url: string) => {
 /* Get the response from the server endpoint */
 const res = await $fetch.raw(url)
 /* Get the cookies from the response */
 const cookies = res.headers.getSetCookie()
 /* Attach each cookie to our incoming Request */
 for (const cookie of cookies) {
 appendResponseHeader(event, 'set-cookie', cookie)
 }
 /* Return the data of the response */
 return res._data
}
``
```

```
``vue
<script setup lang="ts">
// This composable will automatically pass cookies to the client
const event = useRequestEvent()

const { data: result } = await useAsyncData(() => fetchWithCookie(event!, '/api/with-cookie'))

onMounted(() => console.log(document.cookie))
</script>
``
```

### ## Options API support

Nuxt provides a way to perform ``asyncData`` fetching within the Options API. You must wrap your component definition within ``defineNuxtComponent`` for this to work.

```
``vue
<script>
export default defineNuxtComponent({
 /* Use the fetchKey option to provide a unique key */
 fetchKey: 'hello',
 async asyncData () {
 return {
 hello: await $fetch('/api/hello')
 }
 }
})
</script>
``
```

**::note**  
Using ``<script setup>`` or ``<script setup lang="ts">`` are the recommended way of declaring Vue components in Nuxt 3.  
**::**

`:read-more{to="/docs/api/utils/define-nuxt-component"}`

### ## Serializing Data From Server to Client

When using ``useAsyncData`` and ``useLazyAsyncData`` to transfer data fetched on server to the client (as well as anything else

that utilizes [the Nuxt payload](/docs/api/composables/use-nuxt-app#payload)), the payload is serialized with [devalue](https://github.com/Rich-Harris/devalue). This allows us to transfer not just basic JSON but also to serialize and revive/deserialize more advanced kinds of data, such as regular expressions, Dates, Map and Set, `ref`, `reactive`, `shallowRef`, `shallowReactive` and `NuxtError` - and more.

It is also possible to define your own serializer/deserializer for types that are not supported by Nuxt. You can read more in the [useNuxtApp](/docs/api/composables/use-nuxt-app#payload) docs.


:::note

Note that this `_does not apply_` to data passed from your server routes when fetched with `$fetch` or `useFetch` - see the next section for more information.

::

## ## Serializing Data From API Routes

When fetching data from the `server` directory, the response is serialized using `JSON.stringify`. However, since serialization is limited to only JavaScript primitive types, Nuxt does its best to convert the return type of `$fetch` and `useFetch`(/docs/api/composables/use-fetch) to match the actual value.

 [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/JSON/stringify#description](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify#description) Learn more about `JSON.stringify` limitations.

::

### ### Example

```
``ts [server/api/foo.ts]
export default defineEventHandler(() => {
 return new Date()
})
```

```vue [app.vue]
<script setup lang="ts">
// Type of `data` is inferred as string even though we returned a Date object
const { data } = await useFetch('/api/foo')
</script>
```
```

Custom serializer function

To customize the serialization behavior, you can define a `toJSON` function on your returned object. If you define a `toJSON` method, Nuxt will respect the return type of the function and will not try to convert the types.

```
``ts [server/api/bar.ts]
export default defineEventHandler(() => {
  const data = {
    createdAt: new Date(),

    toJSON() {
      return {
        createdAt: {
          year: this.createdAt.getFullYear(),
          month: this.createdAt.getMonth(),
          day: this.createdAt.getDate(),
        },
      },
    },
  },
  return data
})
```

```vue [app.vue]
<script setup lang="ts">
// Type of `data` is inferred as
// {
//   createdAt: {
//     year: number
//     month: number
//     day: number
//   }
// }
const { data } = await useFetch('/api/bar')
</script>
```
```

### ### Using an alternative serializer

Nuxt does not currently support an alternative serializer to `JSON.stringify`. However, you can return your payload as a normal string and utilize the `toJSON` method to maintain type safety.

In the example below, we use [superjson](https://github.com/blitz-js/superjson) as our serializer.

```
``ts [server/api/superjson.ts]
import superjson from 'superjson'

export default defineEventHandler(() => {
 const data = {
 createdAt: new Date(),

 // Workaround the type conversion
 toJSON() {
 return this
 }
 }

 // Serialize the output to string, using superjson
 return superjson.stringify(data) as unknown as typeof data
})
```

```vue [app.vue]
<script setup lang="ts">
import superjson from 'superjson'

// `date` is inferred as { createdAt: Date } and you can safely use the Date object methods
const { data } = await useFetch('/api/superjson', {
 transform: (value) => {
 return superjson.parse(value as unknown as string)
 },
})
</script>
```
```

Recipes

Consuming SSE (Server Sent Events) via POST request

::tip
If you're consuming SSE via GET request, you can use [`EventSource`](https://developer.mozilla.org/en-US/docs/Web/API/EventSource) or VueUse composable [`useEventSource`](https://vueuse.org/core/useEventSource/).
::

When consuming SSE via POST request, you need to handle the connection manually. Here's how you can do it:

```
``ts
// Make a POST request to the SSE endpoint
const response = await $fetch<ReadableStream>('/chats/ask-ai', {
  method: 'POST',
  body: {
    query: "Hello AI, how are you?",
  },
  responseType: 'stream',
})

// Create a new ReadableStream from the response with TextDecoderStream to get the data as text
const reader = response.pipeThrough(new TextDecoderStream()).getReader()

// Read the chunk of data as we get it
while (true) {
  const { value, done } = await reader.read()

  if (done)
    break

  console.log('Received:', value)
}
```
```

```

title: 'Error Handling'
description: 'Learn how to catch and handle errors in Nuxt.'
navigation.icon: i-ph-bug-beetle-duotone

```

Nuxt is a full-stack framework, which means there are several sources of unpreventable user runtime errors that can happen in different contexts:

- Errors during the Vue rendering lifecycle (SSR & CSR)
- Server and client startup errors (SSR + CSR)
- Errors during Nitro server lifecycle ([`server/`](/docs/guide/directory-structure/server) directory)
- Errors downloading JS chunks

```
::tip
SSR stands for **Server-Side Rendering** and **CSR** for **Client-Side Rendering**.
::
```

## ## Vue Errors

You can hook into Vue errors using [`onErrorCaptured`](https://vuejs.org/api/composition-api-lifecycle.html#onerrorcaptured).

In addition, Nuxt provides a [`vue:error`](/docs/api/advanced/hooks#app-hooks-runtime) hook that will be called if any errors propagate up to the top level.

If you are using an error reporting framework, you can provide a global handler through [`vueApp.config.errorHandler`](https://vuejs.org/api/application.html#app-config-errorhandler). It will receive all Vue errors, even if they are handled.

```
``ts twoslash [plugins/error-handler.ts]
export default defineNuxtPlugin((nuxtApp) => {
 nuxtApp.vueApp.config.errorHandler = (error, instance, info) => {
 // handle error, e.g. report to a service
 }

 // Also possible
 nuxtApp.hook('vue:error', (error, instance, info) => {
 // handle error, e.g. report to a service
 })
})
``
```

```
::note
Note that the `vue:error` hook is based on `onErrorCaptured` (https://vuejs.org/api/composition-api-lifecycle.html#onerrorcaptured) lifecycle hook.
::
```

## ## Startup Errors

Nuxt will call the `app:error` hook if there are any errors in starting your Nuxt application.

This includes:

- running [Nuxt plugins](/docs/guide/directory-structure/plugins)
- processing `app:created` and `app:beforeMount` hooks
- rendering your Vue app to HTML (during SSR)
- mounting the app (on client-side), though you should handle this case with `onErrorCaptured` or with `vue:error`
- processing the `app:mounted` hook

## ## Nitro Server Errors

You cannot currently define a server-side handler for these errors, but can render an error page, see the [Render an Error Page](#error-page) section.

## ## Errors with JS chunks

You might encounter chunk loading errors due to a network connectivity failure or a new deployment (which invalidates your old, hashed JS chunk URLs). Nuxt provides built-in support for handling chunk loading errors by performing a hard reload when a chunk fails to load during route navigation.

You can change this behavior by setting `experimental.emitRouteChunkError` to `false` (to disable hooking into these errors at all) or to `manual` if you want to handle them yourself. If you want to handle chunk loading errors manually, you can check out the [the automatic implementation](https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/plugins/chunk-reload.client.ts) for ideas.

## ## Error Page

```
::note
When Nuxt encounters a fatal error (any unhandled error on the server, or an error created with `fatal: true` on the client) it will either render a JSON response (if requested with `Accept: application/json` header) or trigger a full-screen error page.
::
```

An error may occur during the server lifecycle when:

- processing your Nuxt plugins

- rendering your Vue app into HTML
- a server API route throws an error

It can also occur on the client side when:

- processing your Nuxt plugins
- before mounting the application (`app:beforeMount` hook)
- mounting your app if the error was not handled with `onErrorCaptured` or `vue:error` hook
- the Vue app is initialized and mounted in browser (`app:mounted`).

```
::read-more{to="/docs/api/advanced/hooks"}
Discover all the Nuxt lifecycle hooks.
::
```

Customize the default error page by adding `~/error.vue` in the source directory of your application, alongside `app.vue`.

```
<!-- TODO:twoslash: Twoslash does not support tsconfig paths yet -->
```

```
```vue [error.vue]
<script setup lang="ts">
import type { NuxtError } from '#app'

const props = defineProps({
  error: Object as () => NuxtError
})

const handleError = () => clearError({ redirect: '/' })
</script>

<template>
  <div>
    <h2>{{ error.statusCode }}</h2>
    <button @click="handleError">Clear errors</button>
  </div>
</template>
```

::read-more{to="/docs/guide/directory-structure/error"}
Read more about `error.vue` and its uses.
::
```

For custom errors we highly recommend to use `onErrorCaptured` composable that can be called in a page/component setup function or `vue:error` runtime nuxt hook that can be configured in a nuxt plugin.

```
```ts twoslash [plugins/error-handler.ts]
export default defineNuxtPlugin(nuxtApp => {
  nuxtApp.hook('vue:error', (err) => {
    //
  })
})
```
```

When you are ready to remove the error page, you can call the [`clearError`](/docs/api/utils/clear-error) helper function, which takes an optional path to redirect to (for example, if you want to navigate to a 'safe' page).

```
::important
Make sure to check before using anything dependent on Nuxt plugins, such as `$route` or `useRouter`, as if a plugin threw an error, then it won't be re-run until you clear the error.
::
```

```
::note
Rendering an error page is an entirely separate page load, meaning any registered middleware will run again. You can use [`useError`](#useerror) in middleware to check if an error is being handled.
::
```

```
::note
If you are running on Node 16 and you set any cookies when rendering your error page, they will [overwrite cookies previously set](https://github.com/nuxt/nuxt/pull/20585). We recommend using a newer version of Node as Node 16 reached end-of-life in September 2023.
::
```

## ## Error Utils

### ### `useError`

```
```ts [TS Signature]
function useError (): Ref<Error | { url, statusCode, statusMessage, message, description, data }>
```
```

This function will return the global Nuxt error that is being handled.

```
::read-more{to="/docs/api/composables/use-error"}
Read more about `useError` composable.
::
```



### ### `createError`

```
``ts [TS Signature]
function createError (err: string | { cause, data, message, name, stack, statusCode, statusMessage, fatal }): Error
```
```

Create an error object with additional metadata. You can pass a string to be set as the error `message` or an object containing error properties. It is usable in both the Vue and Server portions of your app, and is meant to be thrown.

If you throw an error created with `createError`:

- on server-side, it will trigger a full-screen error page which you can clear with [`clearError`](#clearerror).
- on client-side, it will throw a non-fatal error for you to handle. If you need to trigger a full-screen error page, then you can do this by setting `fatal: true`.

```
```vue twoslash [pages/movies/[slug\].vue]
<script setup lang="ts">
const route = useRoute()
const { data } = await useFetch(`/api/movies/${route.params.slug}`)

if (!data.value) {
 throw createError({
 statusCode: 404,
 statusMessage: 'Page Not Found'
 })
}
</script>
```
```

[::read-more{to="/docs/api/utils/create-error"}](#)
Read more about `createError` util.
::

`showError`

```
``ts [TS Signature]
function showError (err: string | Error | { statusCode, statusMessage }): Error
```
```

You can call this function at any point on client-side, or (on server side) directly within middleware, plugins or `setup()` functions. It will trigger a full-screen error page which you can clear with [`clearError`](#clearerror).

It is recommended instead to use `throw createError()`.

[::read-more{to="/docs/api/utils/show-error"}](#)  
Read more about `showError` util.  
::

### ### `clearError`

```
``ts [TS Signature]
function clearError (options?: { redirect?: string }): Promise<void>
```
```

This function will clear the currently handled Nuxt error. It also takes an optional path to redirect to (for example, if you want to navigate to a 'safe' page).

[::read-more{to="/docs/api/utils/clear-error"}](#)
Read more about `clearError` util.
::

Render Error in Component

Nuxt also provides a [`<NuxtErrorBoundary>`](/docs/api/components/nuxt-error-boundary) component that allows you to handle client-side errors within your app, without replacing your entire site with an error page.

This component is responsible for handling errors that occur within its default slot. On client-side, it will prevent the error from bubbling up to the top level, and will render the `#error` slot instead.

The `#error` slot will receive `error` as a prop. (If you set `error = null` it will trigger re-rendering the default slot; you'll need to ensure that the error is fully resolved first or the error slot will just be rendered a second time.)

[::tip](#)
If you navigate to another route, the error will be cleared automatically.
::

```
```vue [pages/index.vue]
<template>
 <!-- some content -->
 <NuxtErrorBoundary @error="someErrorLogger">
 <!-- You use the default slot to render your content -->
 <template #error="{ error, clearError }">
 You can display the error locally here: {{ error }}
 </template>
 </NuxtErrorBoundary>
</template>
```
```

```
      <button @click="clearError">
        This will clear the error.
      </button>
    </template>
  </NuxtErrorBoundary>
</template>
``,`

:link-example{to="/docs/examples/advanced/error-handling"}
```

```
---
title: "nuxi analyze"
description: "Analyze the production bundle or your Nuxt application."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/analyze.ts
    size: xs
---
```

```
```bash [Terminal]
npx nuxi analyze [--log-level] [rootDir]
```
```

The `analyze` command builds Nuxt and analyzes the production bundle (experimental).

| Option | Default | Description |
|-------------------|---------|--|
| ----- ----- ----- | | |
| `rootDir` | `.` | The directory of the target application. |

```
::note
This command sets `process.env.NODE_ENV` to `production`.
::
```

```
---
title: 'nuxi build-module'
description: 'Nuxt command to build your Nuxt module before publishing.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/build-module.ts
    size: xs
---
```

```
```bash [Terminal]
npx nuxi build-module [--stub] [rootDir]
```
```

The `build-module` command runs `@nuxt/module-builder` to generate `dist` directory within your `rootDir` that contains the full build for your `nuxt-module`.

| Option | Default | Description |
|----------------------|--------------------|--|
| ----- ----- ----- | | |
| <code>rootDir</code> | <code>`.`</code> | The root directory of the module to bundle. |
| <code>--stub</code> | <code>false</code> | Stub out your module for development using [jiti](https://github.com/unjs/jiti#jiti). (**note:** This is mainly for development purposes.) |

```
::read-more{to="https://github.com/nuxt/module-builder" icon="i-simple-icons-github" color="gray" target="_blank"}
Read more about '@nuxt/module-builder'.
::
```

```
---
title: "nuxi add"
description: "Scaffold an entity into your Nuxt application."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/add.ts
    size: xs
---
```

```
```bash [Terminal]
npx nuxi add [--cwd] [--force] <TEMPLATE> <NAME>
```
```

| Option | Default | Description |
|-------------------|---------|--|
| ----- ----- ----- | | |
| `TEMPLATE` | - | Specify a template of the file to be generated. |
| `NAME` | - | Specify a name of the file that will be created. |
| `--cwd` | `.` | The directory of the target application. |
| `--force` | `false` | Force override file if it already exists. |

****Modifiers:****

Some templates support additional modifier flags to add a suffix (like ``.client`` or ``.get``) to their name.

```
```bash [Terminal]
Generates `/plugins/sockets.client.ts`
npx nuxi add plugin sockets --client
```
```

`nuxi add component`

* Modifier flags: `--mode client|server`` or `--client`` or `--server``

```
```bash [Terminal]
Generates `components/TheHeader.vue`
npx nuxi add component TheHeader
```
```

`nuxi add composable`

```
```bash [Terminal]
Generates `composables/foo.ts`
npx nuxi add composable foo
```
```

`nuxi add layout`

```
```bash [Terminal]
Generates `layouts/custom.vue`
npx nuxi add layout custom
```
```

`nuxi add plugin`

* Modifier flags: `--mode client|server`` or `--client`` or `--server``

```
```bash [Terminal]
Generates `plugins/analytics.ts`
npx nuxi add plugin analytics
```
```

`nuxi add page`

```
```bash [Terminal]
Generates `pages/about.vue`
npx nuxi add page about
```
```

```
```bash [Terminal]
Generates `pages/category/[id].vue`
npx nuxi add page "category/[id]"
```
```

`nuxi add middleware`

* Modifier flags: `--global``

```
```bash [Terminal]
Generates `middleware/auth.ts`
npx nuxi add middleware auth
```
```

```
## `nuxi add api`
```

```
* Modifier flags: `--method` (can accept `connect`, `delete`, `get`, `head`, `options`, `patch`, `post`, `put` or `trace`) or  
alternatively you can directly use `--get`, `--post`, etc.
```

```
```bash [Terminal]
```

```
Generates `server/api/hello.ts`
```

```
npx nuxi add api hello
```

```
```
```

```

---
title: "usePreviewMode"
description: "Use usePreviewMode to check and control preview mode in Nuxt"
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/preview.ts
    size: xs
---

```

`usePreviewMode`

Preview mode allows you to see how your changes would be displayed on a live site without revealing them to users.

You can use the built-in `usePreviewMode` composable to access and control preview state in Nuxt. If the composable detects preview mode it will automatically force any updates necessary for [`useAsyncData`](/docs/api/composables/use-async-data) and [`useFetch`](/docs/api/composables/use-fetch) to rerender preview content.

```

```js
const { enabled, state } = usePreviewMode()
```

```

Options

Custom `enable` check

You can specify a custom way to enable preview mode. By default the `usePreviewMode` composable will enable preview mode if there is a `preview` param in url that is equal to `true` (for example, `http://localhost:3000?preview=true`). You can wrap the `usePreviewMode` into custom composable, to keep options consistent across usages and prevent any errors.

```

```js
export function useMyPreviewMode () {
 return usePreviewMode({
 shouldEnable: () => {
 return !!route.query.customPreview
 }
 });
}
```

```

Modify default state

`usePreviewMode` will try to store the value of a `token` param from url in state. You can modify this state and it will be available for all [`usePreviewMode`](/docs/api/composables/use-preview-mode) calls.

```

```js
const data1 = ref('data1')

const { enabled, state } = usePreviewMode({
 getState: (currentState) => {
 return { data1, data2: 'data2' }
 }
})
```

```

:::note

The `getState` function will append returned values to current state, so be careful not to accidentally overwrite important state.

:::

Customize the `onEnable` and `onDisable` callbacks

By default, when `usePreviewMode` is enabled, it will call `refreshNuxtData()` to re-fetch all data from the server.

When preview mode is disabled, the composable will attach a callback to call `refreshNuxtData()` to run after a subsequent router navigation.

You can specify custom callbacks to be triggered by providing your own functions for the `onEnable` and `onDisable` options.

```

```js
const { enabled, state } = usePreviewMode({
 onEnable: () => {
 console.log('preview mode has been enabled')
 },
 onDisable: () => {
 console.log('preview mode has been disabled')
 }
})
```

```

Example

The example below creates a page where part of a content is rendered only in preview mode.

```

```vue [pages/some-page.vue]
<script setup>
const { enabled, state } = usePreviewMode()

const { data } = await useFetch('/api/preview', {
 query: {
 apiKey: state.token
 }
})
</script>

<template>
 <div>
 Some base content
 <p v-if="enabled">
 Only preview content: {{ state.token }}

 <button @click="enabled = false">
 disable preview mode
 </button>
 </p>
 </div>
</template>
```

```

Now you can generate your site and serve it:

```

```bash [Terminal]
npx nuxi generate
npx nuxi preview
```

```

Then you can see your preview page by adding the query param `preview` to the end of the page you want to see once:

```

```js
?preview=true
```

```

```

::note
`usePreviewMode` should be tested locally with `nuxi generate` and then `nuxi preview` rather than `nuxi dev`. (The [preview command](/docs/api/commands/preview) is not related to preview mode.)
::

```



```
---
title: 'useRequestEvent'
description: 'Access the incoming request event with the useRequestEvent composable.'
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
    size: xs
---
```

Within the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context) you can use `useRequestEvent` to access the incoming request.

```
``ts
// Get underlying request event
const event = useRequestEvent()

// Get the URL
const url = event?.path
``
```

```
:::tip
In the browser, `useRequestEvent` will return `undefined`.
::
```

```

---
title: ".env"
description: "A .env file specifies your build/dev-time environment variables."
head.title: ".env"
navigation.icon: i-ph-file-duotone
---

::important
This file should be added to your [.gitignore](/docs/guide/directory-structure/gitignore) file to avoid pushing secrets to your repository.
::

## Dev, Build and Generate Time

Nuxt CLI has built-in [dotenv](https://github.com/motdotla/dotenv) support in development mode and when running [nuxi build](/docs/api/commands/build) and [nuxi generate](/docs/api/commands/generate).

In addition to any process environment variables, if you have a `.env` file in your project root directory, it will be automatically loaded at dev, build and generate time. Any environment variables set there will be accessible within your `nuxt.config` file and modules.

```bash [.env]
MY_ENV_VARIABLE=hello
```

::note
Note that removing a variable from `.env` or removing the `.env` file entirely will not unset values that have already been set.
::

## Custom File

If you want to use a different file - for example, to use `.env.local` or `.env.production` - you can do so by passing the `--dotenv` flag when using `nuxi`.

```bash [Terminal]
npx nuxi dev --dotenv .env.local
```

When updating `.env` in development mode, the Nuxt instance is automatically restarted to apply new values to the `process.env`.

::important
In your application code, you should use [Runtime Config](/docs/guide/going-further/runtime-config) instead of plain env variables.
::

## Production

After your server is built, you are responsible for setting environment variables when you run the server.

Your `.env` files will not be read at this point. How you do this is different for every environment.

This design decision was made to ensure compatibility across various deployment environments, some of which may not have a traditional file system available, such as serverless platforms or edge networks like Cloudflare Workers.

Since `.env` files are not used in production, you must explicitly set environment variables using the tools and methods provided by your hosting environment. Here are some common approaches:

* You can pass the environment variables as arguments using the terminal:

  ```$ DATABASE_HOST=mydatabaseconnectionstring node .output/server/index.mjs```

* You can set environment variables in shell configuration files like `.bashrc` or `.profile`.

* Many cloud service providers, such as Vercel, Netlify, and AWS, provide interfaces for setting environment variables via their dashboards, CLI tools or configuration files.

Production Preview

For local production preview purpose, we recommend using [nuxi preview](/docs/api/commands/preview) since using this command, the `.env` file will be loaded into `process.env` for convenience. Note that this command requires dependencies to be installed in the package directory.

Or you could pass the environment variables as arguments using the terminal. For example, on Linux or macOS:

```bash [Terminal]
DATABASE_HOST=mydatabaseconnectionstring node .output/server/index.mjs
```

Note that for a purely static site, it is not possible to set runtime configuration config after your project is prerendered.

:read-more{to="/docs/guide/going-further/runtime-config"}

```

:::note

If you want to use environment variables set at build time but do not care about updating these down the line (or only need to update them reactively within your app) then `appConfig` may be a better choice. You can define `appConfig` both within your `nuxt.config` (using environment variables) and also within an `~/app.config.ts` file in your project.

:read-more{to="/docs/guide/directory-structure/app-config"}

:::

```

title: "onNuxtReady"
description: The onNuxtReady composable allows running a callback after your app has finished initializing.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ready.ts
 size: xs

```

```
::important
`onNuxtReady` only runs on the client-side. :br
It is ideal for running code that should not block the initial rendering of your app.
::
```

```
```ts [plugins/ready.client.ts]
export default defineNuxtPlugin(() => {
  onNuxtReady(async () => {
    const myAnalyticsLibrary = await import('my-big-analytics-library')
    // do something with myAnalyticsLibrary
  })
})
```
```

It is 'safe' to run even after your app has initialized. In this case, then the code will be registered to run in the next idle callback.

```

title: 'prefetchComponents'
description: Nuxt provides utilities to give you control over prefetching components.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/preload.ts
 size: xs

```

Prefetching component downloads the code in the background, this is based on the assumption that the component will likely be used for rendering, enabling the component to load instantly if and when the user requests it. The component is downloaded and cached for anticipated future use without the user making an explicit request for it.

Use ``prefetchComponents`` to manually prefetch individual components that have been registered globally in your Nuxt app. By default Nuxt registers these as async components. You must use the Pascal-cased version of the component name.

```
```:ts
await prefetchComponents('MyGlobalComponent')

await prefetchComponents(['MyGlobalComponent1', 'MyGlobalComponent2'])
```:
```

```
:::note
Current implementation behaves exactly the same as [`preloadComponents`](/docs/api/utis/preload-components) by preloading components instead of just prefetching we are working to improve this behavior.
::
```

```
:::note
On server, `prefetchComponents` will have no effect.
::
```

```

title: 'preloadComponents'
description: Nuxt provides utilities to give you control over preloading components.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/preload.ts
 size: xs

```

Preloading components loads components that your page will need very soon, which you want to start loading early in rendering lifecycle. This ensures they are available earlier and are less likely to block the page's render, improving performance.

Use ``preloadComponents`` to manually preload individual components that have been registered globally in your Nuxt app. By default Nuxt registers these as async components. You must use the Pascal-cased version of the component name.

```
```js
await preloadComponents('MyGlobalComponent')

await preloadComponents(['MyGlobalComponent1', 'MyGlobalComponent2'])
```
```

```
:::note
On server, `preloadComponents` will have no effect.
:::
```

```

title: "useRequestHeader"
description: "Use useRequestHeader to access a certain incoming request header."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
 size: xs

```

You can use the built-in [`useRequestHeader`](/docs/api/composables/use-request-header) composable to access any incoming request header within your pages, components, and plugins.

```
```:ts
// Get the authorization request header
const authorization = useRequestHeader('authorization')
```

:::tip
In the browser, useRequestHeader will return undefined.
:::
```

## ## Example

We can use `useRequestHeader` to easily figure out if a user is authorized or not.

The example below reads the `authorization` request header to find out if a person can access a restricted resource.

```
```:ts [middleware/authorized-only.ts]
export default defineNuxtRouteMiddleware((to, from) => {
  if (!useRequestHeader('authorization')) {
    return navigateTo('/not-authorized')
  }
})
```
```

```

title: ".gitignore"
description: "A .gitignore file specifies intentionally untracked files that git should ignore."
head.title: ".gitignore"
navigation.icon: i-ph-file-duotone

```

A `.gitignore` file specifies intentionally untracked files that git should ignore.

```
:read-more{icon="i-simple-icons-git" color="gray" title="the git documentation" to="https://git-scm.com/docs/gitignore"
target="_blank"}
```

We recommend having a `.gitignore` file that has **at least** the following entries present:

```
```bash [.gitignore]
# Nuxt dev/build outputs
.output
.data
.nuxt
.nitro
.cache
dist

# Node dependencies
node_modules

# Logs
logs
*.log

# Misc
.DS_Store

# Local env files
.env
.env.*
!.env.example
```
```



```

title: 'Server'
description: Build full-stack applications with Nuxt's server framework. You can fetch data from your database or another
server, create APIs, or even generate static server-side content like a sitemap or a RSS feed - all from a single codebase.
navigation.icon: i-ph-computer-tower-duotone

```

`:read-more{to="/docs/guide/directory-structure/server"}`

## ## Powered by Nitro

`![Server engine](/assets/docs/getting-started/server.svg)`

Nuxt's server is [Nitro](https://github.com/unjs/nitro). It was originally created for Nuxt but is now part of [UnJS](https://unjs.io) and open for other frameworks - and can even be used on its own.

Using Nitro gives Nuxt superpowers:

- Full control of the server-side part of your app
- Universal deployment on any provider (many zero-config)
- Hybrid rendering

Nitro is internally using [h3](https://github.com/unjs/h3), a minimal H(TTP) framework built for high performance and portability.

`::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=DkvgJa-X3lk" target="_blank"}`  
Watch a video from Alexander Lichter to understand the responsibilities of Nuxt and Nitro in your application.  
`::`

## ## Server Endpoints & Middleware

You can easily manage the server-only part of your Nuxt app, from API endpoints to middleware.

Both endpoints and middleware can be defined like this:

```
```:ts twoslash [server/api/test.ts]
export default defineEventHandler(async (event) => {
  // ... Do whatever you want here
})
```
```

And you can directly return ``text``, ``json``, ``html`` or even a ``stream``.

Out-of-the-box, it supports **hot module replacement** and **auto-import** like the other parts of your Nuxt application.

`:read-more{to="/docs/guide/directory-structure/server"}`

## ## Universal Deployment

Nitro offers the ability to deploy your Nuxt app anywhere, from a bare metal server to the edge network, with a start time of just a few milliseconds. That's fast!

`:read-more{to="/blog/nuxt-on-the-edge"}`

There are more than 15 presets to build your Nuxt app for different cloud providers and servers, including:

- [Cloudflare Workers](https://workers.cloudflare.com)
- [Netlify Functions](https://www.netlify.com/products/functions)
- [Vercel Edge Network](https://vercel.com/docs/edge-network/overview)

Or for other runtimes:

```
::card-group
:card{icon="i-logos-deno" title="Deno" to="https://deno.land" target="_blank"}
:card{icon="i-logos-bun" title="Bun" to="https://bun.sh" target="_blank"}
::
```

`:read-more{to="/docs/getting-started/deployment"}`

## ## Hybrid Rendering

Nitro has a powerful feature called ``routeRules`` which allows you to define a set of rules to customize how each route of your Nuxt app is rendered (and more).

```
```:ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  routeRules: {
    // Generated at build time for SEO purpose
    '/': { prerender: true },
    // Cached for 1 hour
    '/api/*': { cache: { maxAge: 60 * 60 } },
    // Redirection to avoid 404
    '/old-page': {
```

```
      redirect: { to: '/new-page', statusCode: 302 }
    }
    // ...
  }
})
``,`
```

`::read-more{to="/docs/guide/concepts/rendering#hybrid-rendering"}`

Learn about all available route rules are available to customize the rendering mode of your routes.

`::`

In addition, there are some route rules (for example, ``ssr``, ``appMiddleware``, and ``experimentalNoScripts``) that are Nuxt specific to change the behavior when rendering your pages to HTML.

Some route rules (``appMiddleware``, ``redirect`` and ``prerender``) also affect client-side behavior.

Nitro is used to build the app for server side rendering, as well as pre-rendering.

`::read-more{to="/docs/guide/concepts/rendering"}`

```
---
title: .nuxtignore
head.title: '.nuxtignore'
description: The .nuxtignore file lets Nuxt ignore files in your project's root directory during the build phase.
navigation.icon: i-ph-file-duotone
---
```

The `.nuxtignore` file tells Nuxt to ignore files in your project's root directory (`[`rootDir`](/docs/api/nuxt-config#rootdir)`) during the build phase.

It is subject to the same specification as `[`.gitignore`](/docs/guide/directory-structure/gitignore)` and `.eslintignore` files, in which each line is a glob pattern indicating which files should be ignored.

```
::tip
You can also configure [`ignoreOptions`](/docs/api/nuxt-config#ignoreoptions), [`ignorePrefix`](/docs/api/nuxt-config#ignoreprefix) and [`ignore`](/docs/api/nuxt-config#ignore) in your nuxt.config file.
::
```

Usage

```
```bash [.nuxtignore]
ignore layout foo.vue
layouts/foo.vue
ignore layout files whose name ends with -ignore.vue
layouts/*-ignore.vue

ignore page bar.vue
pages/bar.vue
ignore page inside ignore folder
pages/ignore/*.vue

ignore route middleware files under foo folder except foo/bar.js
middleware/foo/*.js
!middleware/foo/bar.js
```
```

```
::read-more{icon="i-simple-icons-git" color="gray" title="the git documentation" to="https://git-scm.com/docs/gitignore"
target="_blank"}
More details about the spec are in the gitignore documentation.
::
```

```
---
title: 'Layers'
description: Nuxt provides a powerful system that allows you to extend the default files, configs, and much more.
navigation.icon: i-ph-stack-duotone
---
```

One of the core features of Nuxt is the layers and extending support. You can extend a default Nuxt application to reuse components, utils, and configuration. The layers structure is almost identical to a standard Nuxt application which makes them easy to author and maintain.

Use Cases

- Share reusable configuration presets across projects using `nuxt.config` and `app.config`
- Create a component library using [`components/`](/docs/guide/directory-structure/components) directory
- Create utility and composable library using [`composables/`](/docs/guide/directory-structure/composables) and [`utils/`](/docs/guide/directory-structure/utils) directories
- Create Nuxt module presets
- Share standard setup across projects
- Create Nuxt themes
- Enhance code organization by implementing a modular architecture and support Domain-Driven Design (DDD) pattern in large scale projects.

Usage

By default, any layers within your project in the `~/layers` directory will be automatically registered as layers in your project

```
::note
Layer auto-registration was introduced in Nuxt v3.12.0
::
```

In addition, you can extend from a layer by adding the [extends](/docs/api/nuxt-config#extends) property to your `nuxt.config` file.

```
```:ts [nuxt.config.ts]
export default defineNuxtConfig({
 extends: [
 '../base', // Extend from a local layer
 '@my-themes/awesome', // Extend from an installed npm package
 'github:my-themes/awesome#v1', // Extend from a git repository
]
})
```:
```

You can also pass an authentication token if you are extending from a private GitHub repository:

```
```:ts [nuxt.config.ts]
export default defineNuxtConfig({
 extends: [
 // per layer configuration
 ['github:my-themes/private-awesome', { auth: process.env.GITHUB_TOKEN }]
]
})
```:
```

Nuxt uses [unjs/c12](https://c12.unjs.io) and [unjs/giget](https://giget.unjs.io) for extending remote layers. Check the documentation for more information and all available options.

```
::read-more{to="/docs/guide/going-further/layers"}
Read more about layers in the Layer Author Guide.
::
```

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=lnFCM7c9f7I" target="_blank"}
Watch a video from Learn Vue about Nuxt Layers.
::
```

```
::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=fr5yo3aVkfA" target="_blank"}
Watch a video from Alexander Lichter about Nuxt Layers.
::
```

Examples

```
::card-group
::card
---
icon: i-simple-icons-github
title: Content Wind Theme
to: https://github.com/Atinux/content-wind
target: _blank
ui.icon.base: text-black dark:text-white
---
A lightweight Nuxt theme to build a Markdown driven website. Powered by Nuxt Content, TailwindCSS and Iconify.
::
```



```
---
title: "Prerendering"
description: Nuxt allows pages to be statically rendered at build time to improve certain performance or SEO metrics
navigation.icon: i-ph-code-block-duotone
---
```

Nuxt allows for select pages from your application to be rendered at build time. Nuxt will serve the prebuilt pages when requested instead of generating them on the fly.

```
:read-more{title="Nuxt rendering modes" to="/docs/guide/concepts/rendering"}
```

Crawl-based Pre-rendering

Use the [`nuxi generate` command](/docs/api/commands/generate) to build and pre-render your application using the [Nitro](/docs/guide/concepts/server-engine) crawler. This command is similar to `nuxt build` with the `nitro.static` option set to `true`, or running `nuxt build --prerender`.

This will build your site, stand up a nuxt instance, and, by default, prerender the root page `/` along with any of your site's pages it links to, any of your site's pages they link to, and so on.

```
::code-group
```

```
```bash [npm]
npx nuxi generate
```
```

```
```bash [yarn]
yarn dlx nuxi generate
```
```

```
```bash [pnpm]
pnpm dlx nuxi generate
```
```

```
```bash [bun]
bun x nuxi generate
```
```

```
::
```

You can now deploy the `.output/public` directory to any static hosting service or preview it locally with `npx serve .output/public`.

Working of the Nitro crawler:

1. Load the HTML of your application's root route (`/`), any non-dynamic pages in your `~/pages` directory, and any other routes in the `nitro.prerender.routes` array.
2. Save the HTML and `payload.json` to the `~/output/public/` directory to be served statically.
3. Find all anchor tags (``) in the HTML to navigate to other routes.
4. Repeat steps 1-3 for each anchor tag found until there are no more anchor tags to crawl.

This is important to understand since pages that are not linked to a discoverable page can't be pre-rendered automatically.

```
:read-more{to="/docs/api/commands/generate#nuxi-generate"}
Read more about the `nuxi generate` command.
::
```

Selective Pre-rendering

You can manually specify routes that [Nitro](/docs/guide/concepts/server-engine) will fetch and pre-render during the build or ignore routes that you don't want to pre-render like `/dynamic` in the `nuxt.config` file:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 nitro: {
 prerender: {
 routes: ["/user/1", "/user/2"],
 ignore: ["/dynamic"],
 },
 },
});
```
```

You can combine this with the `crawlLinks` option to pre-render a set of routes that the crawler can't discover like your `/sitemap.xml` or `/robots.txt`:

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
 nitro: {
 prerender: {
 crawlLinks: true,
 routes: ["/sitemap.xml", "/robots.txt"],
 },
 },
});
```
```

```
  },
});
````
```

Setting ``nitro.prerender`` to ``true`` is similar to ``nitro.prerender.crawlLinks`` to ``true``.

```
::read-more{to="https://nitro.unjs.io/config#prerender"}
Read more about pre-rendering in the Nitro documentation.
::
```

Lastly, you can manually configure this using `routeRules`.

```
````ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  routeRules: {
    // Set prerender to true to configure it to be prerendered
    "/rss.xml": { prerender: true },
    // Set it to false to configure it to be skipped for prerendering
    "/this-DOES-NOT-get-prerendered": { prerender: false },
    // Everything under /blog gets prerendered as long as it
    // is linked to from another page
    "/blog/**": { prerender: true },
  },
});
````
```

```
::read-more{to="https://nitro.unjs.io/config/#routerules"}
Read more about Nitro's `routeRules` configuration.
::
```

As a shorthand, you can also configure this in a page file using `[`defineRouteRules`](/docs/api/utils/define-route-rules)`.

```
::read-more{to="/docs/guide/going-further/experimental-features#inlinerrouterules" icon="i-ph-star-duotone"}
This feature is experimental and in order to use it you must enable the `experimental.inlineRouteRules` option in your
`nuxt.config`.
::
```

```
````vue [pages/index.vue]
<script setup>
// Or set at the page level
defineRouteRules({
  prerender: true,
});
</script>

<template>
  <div>
    <h1>Homepage</h1>
    <p>Pre-rendered at build time</p>
  </div>
</template>
````
```

This will be translated to:

```
````ts [nuxt.config.ts]
export default defineNuxtConfig({
  routeRules: {
    "/": { prerender: true },
  },
});
````
```

## Runtime prerender configuration

### ``prerenderRoutes``

You can use this at runtime within a `[Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context)` to add more routes for Nitro to prerender.

```
````vue [pages/index.vue]
<script setup>
prerenderRoutes(["/some/other/url"]);
</script>

<template>
  <div>
    <h1>This will register other routes for prerendering when prerendered</h1>
  </div>
</template>
````
```

```
:read-more{title="prerenderRoutes" to="/docs/api/utils/prerender-routes"}
```

### `prerender:routes` Nuxt hook

This is called before prerendering for additional routes to be registered.

```
``ts [nitro.config.ts]
export default defineNuxtConfig({
 hooks: {
 async "prerender:routes"(ctx) {
 const { pages } = await fetch("https://api.some-cms.com/pages").then(
 (res) => res.json(),
);
 for (const page of pages) {
 ctx.routes.add(`/${page.name}`);
 }
 },
 },
});
```

### `prerender:generate` Nitro hook

This is called for each route during prerendering. You can use this for fine grained handling of each route that gets prerendered.

```
``ts [nitro.config.ts]
export default defineNuxtConfig({
 nitro: {
 hooks: {
 "prerender:generate"(route) {
 if (route.route?.includes("private")) {
 route.skip = true;
 }
 },
 },
 },
});
```



```

title: 'nuxi cleanup'
description: "Remove common generated Nuxt files and caches."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/cleanup.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi cleanup [rootDir]
```
```

The `cleanup` command removes common generated Nuxt files and caches, including:

- `.nuxt`
- `.output`
- `node\_modules/.vite`
- `node\_modules/.cache`

| Option            | Default | Description                        |
|-------------------|---------|------------------------------------|
| ----- ----- ----- |         |                                    |
| `rootDir`         | `.`     | The root directory of the project. |

```

title: "useRequestHeaders"
description: "Use useRequestHeaders to access the incoming request headers."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
 size: xs

```

You can use built-in [`useRequestHeaders`](/docs/api/composables/use-request-headers) composable to access the incoming request headers within your pages, components, and plugins.

```
```js
// Get all request headers
const headers = useRequestHeaders()

// Get only cookie request header
const headers = useRequestHeaders(['cookie'])
```
```

```
:::tip
In the browser, useRequestHeaders will return an empty object.
:::
```

### ## Example

We can use `useRequestHeaders` to access and proxy the initial request's `authorization` header to any future internal requests during SSR.

The example below adds the `authorization` request header to an isomorphic `$fetch` call.

```
```vue [pages/some-page.vue]
<script setup lang="ts">
const { data } = await useFetch('/api/confidential', {
  headers: useRequestHeaders(['authorization'])
})
</script>
```
```

```

title: "nuxi build"
description: "Build your Nuxt application."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/build.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi build [--prerender] [--preset] [--dotenv] [--log-level] [rootDir]
```
```

The `build` command creates a `.output` directory with all your application, server and dependencies ready for production.

| Option                   | Default            | Description                                                                                                          |
|--------------------------|--------------------|----------------------------------------------------------------------------------------------------------------------|
| ----- ----- -----        |                    |                                                                                                                      |
| <code>rootDir</code>     | <code>`.`</code>   | The root directory of the application to bundle.                                                                     |
| <code>--prerender</code> | <code>false</code> | Pre-render every route of your application. (**note:** This is an experimental flag. The behavior might be changed.) |
| <code>--preset</code>    | <code>-</code>     | Set a [Nitro preset](https://nitro.unjs.io/deploy#changing-the-deployment-preset)                                    |
| <code>--dotenv</code>    | <code>`.`</code>   | Point to another <code>.env</code> file to load, <b>relative</b> to the root directory.                              |
| <code>--log-level</code> | <code>info</code>  | Specify build-time logging level, allowing <code>silent</code> \  <code>info</code> \  <code>verbose</code> .        |

**::note**  
This command sets `process.env.NODE_ENV` to `production`.  
**::**

**::note**  
`--prerender` will always set the `preset` to `static`  
**::**

```

title: 'nuxi dev'
description: The dev command starts a development server with hot module replacement at http://localhost:3000
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/dev.ts
 size: xs

```bash [Terminal]
npx nuxi dev [rootDir] [--dotenv] [--log-level] [--clipboard] [--open, -o] [--no-clear] [--port, -p] [--host, -h] [--https]
[--ssl-cert] [--ssl-key] [--tunnel]
```

The `dev` command starts a development server with hot module replacement at [http://localhost:3000](https://localhost:3000)

Option	Default	Description
`rootDir`	`.`	The root directory of the application to serve.
`--dotenv`	`.`	Point to another`.env` file to load, relative to the root directory.
`--open, -o`	`false`	Open URL in browser.
`--clipboard`	`false`	Copy URL to clipboard.
`--no-clear`	`false`	Does not clear the console after startup.
`--port, -p`	`3000`	Port to listen.
`--host, -h`	`localhost`	Hostname of the server.
`--https`	`false`	Listen with`https` protocol with a self-signed certificate by default.
`--ssl-cert`	`null`	Specify a certificate for https.
`--ssl-key`	`null`	Specify the key for the https certificate.
`--tunnel`	`false`	Tunnel your local server to the internet with [unjs/untun](https://github.com/unjs/untun)

The port and host can also be set via NUXT_PORT, PORT, NUXT_HOST or HOST environment variables.

Additionally to the above options, `nuxi` can pass options through to `listen`, e.g. `--no-qr` to turn off the dev server QR code. You can find the list of `listen` options in the [unjs/listen](https://github.com/unjs/listen) docs.

This command sets `process.env.NODE_ENV` to `development`.

::note
If you are using a self-signed certificate in development, you will need to set `NODE_TLS_REJECT_UNAUTHORIZED=0` in your environment.
::
```

```

title: 'useRequestURL'
description: 'Access the incoming request URL with the useRequestURL composable.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/url.ts
 size: xs

```

`useRequestURL` is a helper function that returns an [URL object](https://developer.mozilla.org/en-US/docs/Web/API/URL/URL) working on both server-side and client-side.

**::important**  
When utilizing [Hybrid Rendering](/docs/guide/concepts/rendering#hybrid-rendering) with cache strategies, all incoming request headers are dropped when handling the cached responses via the [Nitro caching layer](https://nitro.unjs.io/guide/cache) (meaning `useRequestURL` will return `localhost` for the `host`).

You can define the [`cache.varies` option](https://nitro.unjs.io/guide/cache#options) to specify headers that will be considered when caching and serving the responses, such as `host` and `x-forwarded-host` for multi-tenant environments.

**::**

**::code-group**

```
```vue [pages/about.vue]
<script setup lang="ts">
const url = useRequestURL()
</script>

<template>
  <p>URL is: {{ url }}</p>
  <p>Path is: {{ url.pathname }}</p>
</template>
```

```html [Result in development]
<p>URL is: http://localhost:3000/about</p>
<p>Path is: /about</p>
```
```

**::**

**::tip**{icon="i-simple-icons-mdnwebdocs" color="gray" to="https://developer.mozilla.org/en-US/docs/Web/API/URL#instance\_properties" target="\_blank"}  
Read about the URL instance properties on the MDN documentation.

**::**

```

title: app.config.ts
head.title: 'app.config.ts'
description: Expose reactive configuration within your application with the App Config file.
navigation.icon: i-ph-file-duotone

```

Nuxt provides an `app.config` config file to expose reactive configuration within your application with the ability to update it at runtime within lifecycle or using a nuxt plugin and editing it with HMR (hot-module-replacement).

You can easily provide runtime app configuration using `app.config.ts` file. It can have either of `.ts`, `.js`, or `.mjs` extensions.

```

```ts twoslash [app.config.ts]
export default defineAppConfig({
  foo: 'bar'
})
```

```

```

::caution
Do not put any secret values inside `app.config` file. It is exposed to the user client bundle.
::

```

## ## Usage

To expose config and environment variables to the rest of your app, you will need to define configuration in `app.config` file.

```

```ts twoslash [app.config.ts]
export default defineAppConfig({
  theme: {
    primaryColor: '#ababab'
  }
})
```

```

When adding `theme` to the `app.config`, Nuxt uses Vite or webpack to bundle the code. We can universally access `theme` both when server-rendering the page and in the browser using `[useAppConfig] (/docs/api/composables/use-app-config)` composable.

```

```vue [pages/index.vue]
<script setup lang="ts">
const appConfig = useAppConfig()

console.log(appConfig.theme)
</script>
```

```

When configuring a custom `[srcDir] (/docs/api/nuxt-config#srcdir)`, make sure to place the `app.config` file at the root of the new `srcDir` path.

## ## Typing App Config

Nuxt tries to automatically generate a TypeScript interface from provided app config so you won't have to type it yourself.

However, there are some cases where you might want to type it yourself. There are two possible things you might want to type.

### ### App Config Input

`AppConfigInput` might be used by module authors who are declaring what valid `_input_` options are when setting app config. This will not affect the type of `useAppConfig()`.

```

```ts [index.d.ts]
declare module 'nuxt/schema' {
  interface AppConfigInput {
    /** Theme configuration */
    theme?: {
      /** Primary app color */
      primaryColor?: string
    }
  }
}

// It is always important to ensure you import/export something when augmenting a type
export {}
```

```

### ### App Config Output

If you want to type the result of calling `[useAppConfig] (/docs/api/composables/use-app-config)`, then you will want to extend `AppConfig`.

```

::warning
Be careful when typing `AppConfig` as you will overwrite the types Nuxt infers from your actually defined app config.

```

```

::

```ts [index.d.ts]
declare module 'nuxt/schema' {
  interface AppConfig {
    // This will entirely replace the existing inferred `theme` property
    theme: {
      // You might want to type this value to add more specific types than Nuxt can infer,
      // such as string literal types
      primaryColor?: 'red' | 'blue'
    }
  }
}

// It is always important to ensure you import/export something when augmenting a type
export {}
```

```

## ## Merging Strategy

Nuxt uses a custom merging strategy for the `AppConfig` within [the layers](/docs/getting-started/layers) of your application.

This strategy is implemented using a [Function Merger](https://github.com/unjs/defu#function-merger), which allows defining a custom merging strategy for every key in `app.config` that has an array as value.

```

::note
The function merger can only be used in the extended layers and not the main `app.config` in project.
::

```

Here's an example of how you can use:

```

::code-group

```ts twoslash [layer/app.config.ts]
export default defineAppConfig({
  // Default array value
  array: ['hello'],
})
```

```ts twoslash [app.config.ts]
export default defineAppConfig({
  // Overwrite default array value by using a merger function
  array: () => ['bonjour'],
})
```

::

```

```

title: 'preloadRouteComponents'
description: preloadRouteComponents allows you to manually preload individual pages in your Nuxt app.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/preload.ts
 size: xs

```

Preloading routes loads the components of a given route that the user might navigate to in future. This ensures that the components are available earlier and less likely to block the navigation, improving performance.

```

::tip{icon="i-ph-rocket-launch-duotone" color="gray"}
Nuxt already automatically preloads the necessary routes if you're using the `NuxtLink` component.
::

:read-more{to="/docs/api/components/nuxt-link"}

```

### ## Example

Preload a route when using `navigateTo`.

```

```ts
// we don't await this async function, to avoid blocking rendering
// this component's setup function
preloadRouteComponents('/dashboard')

const submit = async () => {
  const results = await $fetch('/api/authentication')

  if (results.token) {
    await navigateTo('/dashboard')
  }
}
```

:read-more{to="/docs/api/utils/navigate-to"}

::note
On server, `preloadRouteComponents` will have no effect.
::

```



```

title: 'prerenderRoutes'
description: prerenderRoutes hints to Nitro to prerender an additional route.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
 size: xs

```

When prerendering, you can hint to Nitro to prerender additional paths, even if their URLs do not show up in the HTML of the generated page.

```
::important
`prerenderRoutes` can only be called within the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context).
::

::note
`prerenderRoutes` has to be executed during prerendering. If the `prerenderRoutes` is used in dynamic pages/routes which are not prerendered, then it will not be executed.
::

```js
const route = useRoute()

prerenderRoutes('/')
prerenderRoutes(['/ ', '/about'])
```

::note
In the browser, or if called outside prerendering, `prerenderRoutes` will have no effect.
::
```

```

title: "refreshCookie"
description: "Refresh useCookie values manually when a cookie has changed"
navigation:
 badge: New
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/cookie.ts
 size: xs

```

```
::important
This utility is available since [Nuxt v3.10](/blog/v3-10).
::
```

## ## Purpose

The `refreshCookie` function is designed to refresh cookie value returned by `useCookie`.

This is useful for updating the `useCookie` ref when we know the new cookie value has been set in the browser.

## ## Usage

```
``vue [app.vue]
<script setup lang="ts">
const tokenCookie = useCookie('token')

const login = async (username, password) => {
 const token = await $fetch('/api/token', { ... }) // Sets `token` cookie on response
 refreshCookie('token')
}

const loggedIn = computed(() => !!tokenCookie.value)
</script>
``
```

```
::note{to="/docs/guide/going-further/experimental-features#cookiestore"}
You can enable experimental cookieStore option to automatically refresh useCookie value when cookie changes in the browser.
::
```

## ## Type

```
``ts
refreshCookie(name: string): void
``
```

```

title: 'useRouteAnnouncer'
description: This composable observes the page title changes and updates the announcer message accordingly.
navigation:
 badge: New
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/route-announcer.ts
 size: xs

::important
This composable is available in Nuxt v3.12+.
::
```

## Description

A composable which observes the page title changes and updates the announcer message accordingly. Used by [`<NuxtRouteAnnouncer>`](/docs/api/components/nuxt-route-announcer) and controllable. It hooks into Unhead's [`dom:rendered`](https://unhead.unjs.io/api/core/hooks#dom-hooks) to read the page's title and set it as the announcer message.

## Parameters

- `politeness`: Sets the urgency for screen reader announcements: `off` (disable the announcement), `polite` (waits for silence), or `assertive` (interrupts immediately). (default `polite`).

## Properties

### `message`

- **type**: `Ref<string>`
- **description**: The message to announce

### `politeness`

- **type**: `Ref<string>`
- **description**: Screen reader announcement urgency level `off`, `polite`, or `assertive`

## Methods

### `set(message, politeness = "polite")`

Sets the message to announce with its urgency level.

### `polite(message)`

Sets the message with `politeness = "polite"`

### `assertive(message)`

Sets the message with `politeness = "assertive"`

## Example

```
``vue [pages/index.vue]
<script setup lang="ts">
 const { message, politeness, set, polite, assertive } = useRouteAnnouncer({
 politeness: 'assertive'
 })
</script>
``
```

```

title: "app.vue"
description: "The app.vue file is the main component of your Nuxt application."
head.title: "app.vue"
navigation.icon: i-ph-file-duotone

```

## ## Minimal Usage

With Nuxt 3, the `[`pages/`](/docs/guide/directory-structure/pages)` directory is optional. If not present, Nuxt won't include `[vue-router](https://router.vuejs.org)` dependency. This is useful when working on a landing page or an application that does not need routing.

```
``vue [app.vue]
<template>
 <h1>Hello World!</h1>
</template>
``
```

```
:link-example{to="/docs/examples/hello-world"}
```

## ## Usage with Pages

If you have a `[`pages/`](/docs/guide/directory-structure/pages)` directory, to display the current page, use the `[`<NuxtPage>`](/docs/api/components/nuxt-page)` component:

```
``vue [app.vue]
<template>
 <div>
 <NuxtLayout>
 <NuxtPage/>
 </NuxtLayout>
 </div>
</template>
``
```

```
::warning
Since [`${<NuxtPage>}`](/docs/api/components/nuxt-page) internally uses Vue's [`${<Suspense>}`](https://vuejs.org/guide/built-ins/suspense.html#suspense) component, it cannot be set as a root element.
::
```

```
::note
Remember that `app.vue` acts as the main component of your Nuxt application. Anything you add to it (JS and CSS) will be global and included in every page.
::
```

```
::read-more{to="/docs/guide/directory-structure/layouts"}
If you want to have the possibility to customize the structure around the page between pages, check out the `layouts/` directory.
::
```

```

title: "error.vue"
description: "The error.vue file is the error page in your Nuxt application."
head.title: "error.vue"
navigation.icon: i-ph-file-duotone

```

During the lifespan of your application, some errors may appear unexpectedly at runtime. In such case, we can use the `error.vue` file to override the default error files and display the error nicely.

```

```vue [error.vue]
<script setup lang="ts">
import type { NuxtError } from '#app'

const props = defineProps({
  error: Object as () => NuxtError
})
</script>

<template>
  <div>
    <h1>{{ error.statusCode }}</h1>
    <NuxtLink to="/">Go back home</NuxtLink>
  </div>
</template>
```

```

::note  
 Although it is called an 'error page' it's not a route and shouldn't be placed in your `~/pages` directory. For the same reason, you shouldn't use `definePageMeta` within this page. That being said, you can still use layouts in the error file, by utilizing the [`NuxtLayout`](/docs/api/components/nuxt-layout) component and specifying the name of the layout.  
 ::

The error page has a single prop - `error` which contains an error for you to handle.

The `error` object provides the following fields:

```

```ts
{
  statusCode: number
  fatal: boolean
  unhandled: boolean
  statusMessage?: string
  data?: unknown
  cause?: unknown
}
```

```

If you have an error with custom fields they will be lost; you should assign them to `data` instead:

```

```ts
throw createError({
  statusCode: 404,
  statusMessage: 'Page Not Found',
  data: {
    myCustomField: true
  }
})
```

```

```

title: 'Reporting Bugs'
description: 'One of the most valuable roles in open source is taking the time to report bugs helpfully.'
navigation.icon: i-ph-bug-duotone

```

Try as we might, we will never completely eliminate bugs.

Even if you can't fix the underlying code, reporting a bug well can enable someone else with a bit more familiarity with the codebase to spot a pattern or make a quick fix.

Here are a few key steps.

## ## Is It Really a Bug?

Consider if you're looking to get help with something, or whether you think there's a bug with Nuxt itself. If it's the former, we'd love to help you - but the best way to do that is through [asking for help](/docs/community/getting-help) rather than reporting a bug.

## ## Search the Issues

Search through the [open issues](https://github.com/nuxt/nuxt/issues) and [discussions](https://github.com/nuxt/nuxt/discussions) first. If you find anything that seems like the same bug, it's much better to comment on an existing thread than create a duplicate.

## ## Create a Minimal Reproduction

It's important to be able to reproduce the bug reliably - in a minimal way and apart from the rest of your project. This narrows down what could be causing the issue and makes it possible for someone not only to find the cause, but also to test a potential solution.

Start with the Nuxt sandbox and add the **minimum** amount of code necessary to reproduce the bug you're experiencing.

```
::note
If your issue concerns Vue or Vite, please try to reproduce it first with the Vue SSR starter.
::
```

```
Nuxt:

::card-group
 :card{title="Nuxt on StackBlitz" icon="i-simple-icons-stackblitz" to="https://nuxt.new/s/v3" target="_blank"}
 :card{title="Nuxt on CodeSandbox" icon="i-simple-icons-codesandbox" to="https://nuxt.new/c/v3" target="_blank"}
::

Vue:

::card-group
 :card{title="Vue SSR on StackBlitz" icon="i-simple-icons-stackblitz" to="https://stackblitz.com/github/nuxt-contrib/vue3-ssr-starter/tree/main?terminal=dev" target="_blank"}
 :card{title="Vue SSR on CodeSandbox" icon="i-simple-icons-codesandbox" to="https://codesandbox.io/s/github/nuxt-contrib/vue3-ssr-starter/main" target="_blank"}
 :card{title="Vue SSR Template on GitHub" icon="i-simple-icons-github" to="https://github.com/nuxt-contrib/vue3-ssr-starter/generate" target="_blank"}
::
```

Once you've reproduced the issue, remove as much code from your reproduction as you can (while still recreating the bug). The time spent making the reproduction as minimal as possible will make a huge difference to whoever sets out to fix the issue.

## ## Figure Out What the Cause Might Be

With a Nuxt project, there are lots of moving pieces - from [Nuxt modules](/modules) to [other JavaScript libraries](https://www.npmjs.com). Try to report the bug at the most relevant and specific place. That will likely be the Nuxt module causing an issue, or the upstream library that Nuxt is depending on.

```

title: Getting Help
description: We're a friendly community of developers and we'd love to help.
navigation:
 icon: i-ph-lifebuoy-duotone

```

At some point, you may find that there's an issue you need some help with.

But don't worry! We're a friendly community of developers and we'd love to help.

```
"I can't figure out how to (...)."
```

You've read through these docs and you think it should be possible, but it's not clear how. The best thing is to [open a GitHub Discussion](https://github.com/nuxt/nuxt/discussions).

Please don't feel embarrassed about asking a question that you think is easy - we've all been there! â€œ,

Everyone you'll encounter is helping out because they care, not because they are paid to do so. The kindest thing to do is make it easy for them to help you. Here are some ideas:

- Explain what your objective is, not just the problem you're facing. "I need to ensure my form inputs are accessible, so I'm trying to get the ids to match between server and client."
- Make sure you've first read the docs and used your favorite search engine. Let people know by saying something like "I've Googled for 'nuxt script setup' but I couldn't find code examples anywhere."
- Explain what you've tried. Tell people the kind of solutions you've experimented with, and why. Often this can make people's advice more relevant to your situation.
- Share your code. People probably won't be able to help if they just see an error message or a screenshot - but that all changes if you share your code in a copy/pasteable format - preferably in the form of a minimal reproduction like a CodeSandbox.

And finally, just ask the question! There's no need to [ask permission to ask a question](https://dontasktoask.com) or [wait for someone to reply to your 'hello'](https://www.nohello.com). If you do, you might not get a response because people are waiting for the whole question before engaging.

```
"Could there be a bug?"
```

Something isn't working the way that the docs say that it should. You're not sure if it's a bug. You've searched through the [open issues](https://github.com/nuxt/nuxt/issues) and [discussions](https://github.com/nuxt/nuxt/discussions) but you can't find anything. (if there is a closed issue, please create a new one)

We recommend taking a look at [how to report bugs](/docs/community/reporting-bugs). Nuxt is still in active development, and every issue helps make it better.

```
"I need professional help"
```

If the community couldn't provide the help you need in the time-frame you have, NuxtLabs offers professional support with the [Nuxt Experts](https://nuxt.com/enterprise/support).

The objective of the Nuxt Expert is to provide support to the Vue ecosystem, while also creating freelance opportunities for those contributing to open-source solutions, thus helping to maintain the sustainability of the ecosystem.

The Nuxt experts are Vue, Nuxt and Vite chosen contributors providing professional support and consulting services.

```

title: 'Contribution'
description: 'Nuxt is a community project - and so we love contributions of all kinds! â€¦,'
navigation.icon: i-ph-git-pull-request-duotone

```

There is a range of different ways you might be able to contribute to the Nuxt ecosystem.

## ## Ecosystem

The Nuxt ecosystem includes many different projects and organizations:

- \* [nuxt/](https://github.com/nuxt) - core repositories for the Nuxt framework itself. [**nuxt/nuxt**](https://github.com/nuxt/nuxt) contains the Nuxt framework (both versions 2 and 3).
- \* [nuxt-modules/](https://github.com/nuxt-modules) - community-contributed and maintained modules and libraries. There is a [process to migrate a module](/docs/guide/going-further/modules/#joining-nuxt-modules-and-nuxtjs) to `nuxt-modules`. While these modules have individual maintainers, they are not dependent on a single person.
- \* [unjs/](https://github.com/unjs) - many of these libraries are used throughout the Nuxt ecosystem. They are designed to be universal libraries that are framework- and environment-agnostic. We welcome contributions and usage by other frameworks and projects.

## ## How To Contribute

### ### Triage Issues and Help Out in Discussions

Check out the issues and discussions for the project you want to help. For example, here are [the issues board](https://github.com/nuxt/nuxt/issues) and [discussions](https://github.com/nuxt/nuxt/discussions) for Nuxt 3. Helping other users, sharing workarounds, creating reproductions, or even poking into a bug a little bit and sharing your findings makes a huge difference.

### ### Creating an Issue

Thank you for taking the time to create an issue! â€¦,

- \* **Reporting bugs**: Check out [our guide](/docs/community/reporting-bugs) for some things to do before opening an issue.

- \* **Feature requests**: Check that there is not an existing issue or discussion covering the scope of the feature you have in mind. If the feature is to another part of the Nuxt ecosystem (such as a module), please consider raising a feature request there first. If the feature you have in mind is general or the API is not entirely clear, consider opening a discussion in the **Ideas** section to discuss with the community first.

We'll do our best to follow our [internal issue decision making flowchart](https://mermaid.live/view#pako:eNqFLE1v2zAMhv8K4UuToslhx2Bo0TZt12Edhm7YMCAXWqJtorLk6q0pkfS\_j7KdfpyWQ-BQR8mHL6nsCuU0FauIMm6rGvQRfq03FuRzvvvTYIQHthpcBT\_uqQNwPHUzjheLxf4i1VDx8x4udrf5EBC0QvSsYg4ffS79KS9pmX9QALTgyid2KYB7Ih-4bmKwbDk2YB0E1grUVaRi-FDmmjAmT3u4nB3DmoNKIUA1BsGSohA49jnVMQhHbDh\_EZQUImyxh-gAtfaiG-KWSJ-N8nt6YtpCdgeE5rXP0dav5YwWJIJU7zrvNADV9C7JBIyIC07Wxupkx3LFQ5vCkguRno5f9fP2qnUko0Y2dk9rGdvHAA9IIhVGLCp5FFNPN-ce4DKeXBd53xMli0Lp9IZtyORQVsnrGm-WJzejtUu5ffQdr5FGQ3bLslYvGthjZbJTLpReZG5\_lLYw7XQ\_CbPVT92ws9gnEJj-v84dk-PiaXnmF1XGAaPs0sMKywNvYmG80ZohV8k4wDR9\_N3KN\_dHm5mh1lnkM5FsYzRfNiTvJoT5gnQsl6uxjqXLhkNq9syHJ0UZZ8ERUIlNShr6N8gzDEliR-ow7QZa0fhY4LoHLRo-8N7ZxPwjRj5ZZYXpv0SNs9v3Jjs8NXB4ets92xan3zydXZHvj64lKMayh4-gZC1bjASW2ipLeWuzIuToiXfImu5rbucclMIc0ubYiWPGv3DptjYF9Fhiu5nb1Wxij7RSZE6jZHWjLXHtlhVaIJESXN0\_m68\_s0\_wMs\_o09gyg) when responding to issues.

### ### Send a Pull Request

We always welcome pull requests! â€¦,

### #### Before You Start

Before you fix a bug, we recommend that you check whether **there's an issue that describes it**, as it's possible it's a documentation issue or that there is some context that would be helpful to know.

If you're working on a feature, then we ask that you **open a feature request issue first** to discuss with the maintainers whether the feature is desired - and the design of those features. This helps save time for both the maintainers and the contributors and means that features can be shipped faster. The issue **should be confirmed** by a framework team member before building out a feature in a pull request.

For typo fixes, it's recommended to batch multiple typo fixes into one pull request to maintain a cleaner commit history.

For bigger changes to Nuxt itself, we recommend that you first [create a Nuxt module](#create-a-module) and implement the feature there. This allows for quick proof-of-concept. You can then [create an RFC](#make-an-rfc) in the form of a discussion. As users adopt it and you gather feedback, it can then be refined and either added to Nuxt core or continue as a standalone module.

### #### Commit Conventions

We use [Conventional Commits](https://www.conventionalcommits.org) for commit messages, which [allows a changelog to be auto-generated](https://github.com/unjs/changelogen) based on the commits. Please read the guide through if you aren't familiar with it already.

Note that `fix:` and `feat:` are for **actual code changes** (that might affect logic). For typo or document changes, use `docs:` or `chore:` instead:

- \* `~~fix: typo~~ -> docs: fix typo``



If you are working in a project with a monorepo, like `nuxt/nuxt`, ensure that you specify the main scope of your commit in brackets. For example: `feat(nuxi): add 'do-magic' command`.

#### #### Making the Pull Request

If you don't know how to send a pull request, we recommend reading [the guide](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request).

When sending a pull request, make sure your PR's title also follows the [Commit Convention](#commit-conventions).

If your PR fixes or resolves existing issues, please make sure you mention them in the PR description.

It's ok to have multiple commits in a single PR; you don't need to rebase or force push for your changes as we will use `Squash and Merge` to squash the commits into one commit when merging.

We do not add any commit hooks to allow for quick commits. But before you make a pull request, you should ensure that any lint/test scripts are passing.

In general, please also make sure that there are no unrelated changes in a PR. For example, if your editor has made any changes to whitespace or formatting elsewhere in a file that you edited, please revert these so it is more obvious what your PR changes. And please avoid including multiple unrelated features or fixes in a single PR. If it is possible to separate them, it is better to have multiple PRs to review and merge separately. In general, a PR should do one thing only.

#### #### Once You've Made a Pull Request

Once you've made a pull request, we'll do our best to review it promptly.

If we assign it to a maintainer, then that means that person will take special care to review it and implement any changes that may be required.

If we request changes on a PR, please ignore the red text! It doesn't mean we think it's a bad PR - it's just a way of easily telling the status of a list of pull requests at a glance.

If we mark a PR as 'pending', that means we likely have another task to do in reviewing the PR - it's an internal note-to-self, and not necessarily a reflection on whether the PR is a good idea or not. We will do our best to explain via a comment the reason for the pending status.

We'll do our best to follow [our PR decision making flowchart](https://mermaid.live/view#pako:eNp9VE1v2kAQ\_SsjXzBSEqlALLaUisSh0ACK2l4qcVm8Y9hi7672Iwly-0-ZtYPt5FA0CHbee\_PmzdpVLcmOURLlhXrJ9sw4-JNuJNBnWs1UQafIQVjrERyWumA0v58-AJExT29\_0b7BXbWwwL0uRPa1vLZvcB\_fF8oiMMmB2QM4BXkt3Uo0N7Lh3LWaDz2SVkK6Qgt7DHvw0CKt5sxCKaQoWQEGtVHcZ04oGdw04LTVngW\_LHOeFcURGgZ97mw6PSv-iJdsi0UCA4nI7SfNwc3W3JZit3eQ1SZFDlKB15yswQ2Mgb0jbYeatY3n8bcr-IWlekYYaJRCyB04I9g0B1CEfKF5dAVTzmFAtnqn4-bUYAiMMmHZgWhNPRhgus5mW2BATxq0NkIZ4Y4NbNjzE2ZchBzcHmGLE\_ZMSKCCyRXyLrVFfa\_5n\_PBK2xKy3kk9e0jULUdltk6C8kI-7NFD8f4EVGDoqlp-wa4sJm3ltIMiuZ\_mTQXJyTSkQZtunPqsKxShV9GKdkBYe1fHXjpbclvONL09Kqx\_M7YHm0mav\_luxfE5zKwVs09hM5DLSupgYDlr5fLDkwo7ykixKG-xDsUly1LZ-uY32dgDc7LG7YqwbNp0msJwmIUivjWFtfd-xRrEcJ70mydz37qFplH0txEp4GskI2qB5dRCWakgl0z3oV8JuITJa4iRL6yZk5bKKNPBG0ead-H2UWJc54vIiaW53SPgwrz4fIhVNmbw76lfI6R2\_MW21) when responding and reviewing to pull requests.

#### ### Create a Module

If you've built something with Nuxt that's cool, why not [extract it into a module](/docs/guide/going-further/modules), so it can be shared with others? We have [many excellent modules already](/modules), but there's always room for more.

If you need help while building it, feel free to [check in with us](/docs/community/getting-help).

#### ### Make an RFC

We highly recommend [creating a module](#create-a-module) first to test out big new features and gain community adoption.

If you have done this already, or it's not appropriate to create a new module, then please start by creating a new discussion. Make sure it explains your thinking as clearly as possible. Include code examples or function signatures for new APIs. Reference existing issues or pain points with examples.

If we think this should be an RFC, we'll change the category to RFC and broadcast it more widely for feedback.

An RFC will then move through the following stages:

- \* `rfc: active` - currently open for comment
- \* `rfc: approved` - approved by the Nuxt team
- \* `rfc: ready to implement` - an issue has been created and assigned to implement
- \* `rfc: shipped` - implemented
- \* `rfc: archived` - not approved, but archived for future reference

#### ### Conventions Across Ecosystem

The following conventions are required within the `nuxt/` organization and recommended for other maintainers in the ecosystem.

#### #### Module Conventions

Modules should follow the [Nuxt module template](https://github.com/nuxt/starter/tree/module). See [module guide](/docs/guide/going-further/modules) for more information.

#### #### Use Core `unjs/` Libraries

We recommend the following libraries which are used throughout the ecosystem:

- \* [pathe](https://github.com/unjs/pathe) - universal path utilities (replacement for node `path`)
- \* [ufo](https://github.com/unjs/ufo) - URL parsing and joining utilities
- \* [unbuild](https://github.com/unjs/unbuild) - rollup-powered build system
- \* ... check out the rest of the [unjs/](https://github.com/unjs) organization for many more!

#### #### Use ESM Syntax and Default to `type: module`

Most of the Nuxt ecosystem can consume ESM directly. In general we advocate that you avoid using CJS-specific code, such as `\_\_dirname` and `require` statements. You can [read more about ESM](/docs/guide/concepts/esm).

#### #### What's Corepack

[Corepack](https://nodejs.org/api/corepack.html) makes sure you are using the correct version for package manager when you run corresponding commands. Projects might have `packageManager` field in their `package.json`.

Under projects with configuration as shown below, Corepack will install `v7.5.0` of `pnpm` (if you don't have it already) and use it to run your commands.

```
```jsonc [package.json]
{
  "packageManager": "pnpm@7.5.0"
}
```
```

#### #### Use ESLint

We use [ESLint](https://eslint.org) for both linting and formatting with [`@nuxt/eslint-config`](https://github.com/nuxt/eslint-config).

#### ##### IDE Setup

We recommend using [VS Code](https://code.visualstudio.com) along with the [ESLint extension](https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint). If you would like, you can enable auto-fix and formatting when you save the code you are editing:

```
```json [settings.json]
{
  "editor.codeActionsOnSave": {
    "source.fixAll": "never",
    "source.fixAll.eslint": "explicit"
  }
}
```
```

#### #### No Prettier

Since ESLint is already configured to format the code, there is no need to duplicate the functionality with Prettier. To format the code, you can run `yarn lint --fix`, `pnpm lint --fix`, or `bun run lint --fix` or referring the [ESLint section] (#use-eslint) for IDE Setup.

If you have Prettier installed in your editor, we recommend you disable it when working on the project to avoid conflict.

**\*\*Note\*\*:** [we are discussing](https://github.com/nuxt/eslint-config/issues/224) enabling Prettier in future.

#### #### Package Manager

For libraries, we recommend `pnpm`. For modules, we still recommend `yarn` but we may well switch this recommendation to `pnpm` in future once we support plug and play mode with Nuxt itself.

It is important to enable Corepack to ensure you are on the same version of the package manager as the project. Corepack is built-in to new node versions for seamless package manager integration.

To enable it, run

```
```bash [Terminal]
corepack enable
```
```

You only need to do this one time, after Node.js is installed on your computer.

#### ## Documentation Style Guide

Documentation is an essential part of Nuxt. We aim to be an intuitive framework - and a big part of that is making sure that both the developer experience and the docs are perfect across the ecosystem. 🤝

Here are some tips that may help improve your documentation:

- \* Avoid subjective words like `_simply_`, `_just_`, `_obviously..._` when possible.

Keep in mind your readers can have different backgrounds and experiences. Therefore, these words don't convey meaning and can be harmful.

```
::caution{ icon="i-ph-x-circle-duotone"}
```

Simply make sure the function returns a promise.

```
::
```

```
::tip{icon="i-ph-check-circle-duotone"}
```

Make sure the function returns a [promise]([https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)).

```
::
```

\* Prefer [active voice](<https://developers.google.com/tech-writing/one/active-voice>).

```
::caution{icon="i-ph-x-circle-duotone"}
```

An error will be thrown by Nuxt.

```
::
```

```
::tip{icon="i-ph-check-circle-duotone"}
```

Nuxt will throw an error.

```
::
```

```
::read-more{to="/docs/community/framework-contribution#documentation-guide"}
```

Learn how to contribute to the documentation.

```
::
```

```

title: "nuxi info"
description: The info command logs information about the current or specified Nuxt project.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/info.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi info [rootDir]
```
```

The `info` command logs information about the current or specified Nuxt project.

| Option            | Default | Description                              |
|-------------------|---------|------------------------------------------|
| ----- ----- ----- |         |                                          |
| `rootDir`         | `.`     | The directory of the target application. |

```

title: "nuxi generate"
description: Pre-renders every route of the application and stores the result in plain HTML files.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/generate.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi generate [rootDir] [--dotenv]
```
```

The `generate` command pre-renders every route of your application and stores the result in plain HTML files that you can deploy on any static hosting services. The command triggers the `nuxi build` command with the `prerender` argument set to `true`

| Option                | Default          | Description                                                                             |
|-----------------------|------------------|-----------------------------------------------------------------------------------------|
| <code>rootDir</code>  | <code>`.`</code> | The root directory of the application to generate                                       |
| <code>--dotenv</code> | <code>`.`</code> | Point to another <code>.env</code> file to load, <b>relative</b> to the root directory. |

```
::read-more{to="/docs/getting-started/deployment#static-hosting"}
Read more about pre-rendering and static hosting.
::
```

```

title: "nuxi devtools"
description: The devtools command allows you to enable or disable Nuxt DevTools on a per-project basis.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/devtools.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi devtools enable|disable [rootDir]
```
```

Running `nuxi devtools enable` will install the Nuxt DevTools globally, and also enable it within the particular project you are using. It is saved as a preference in your user-level `.nuxtrc`. If you want to remove devtools support for a particular project, you can run `nuxi devtools disable`.

| Option            | Default | Description                                                    |
|-------------------|---------|----------------------------------------------------------------|
| ----- ----- ----- |         |                                                                |
| `rootDir`         | `.`     | The root directory of the app you want to enable devtools for. |

```
::read-more{icon="i-simple-icons-nuxtdotjs" to="https://devtools.nuxt.com" target="_blank"}
Read more about the Nuxt DevTools.
::
```

```

title: "useRoute"
description: The useRoute composable returns the current route.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
 size: xs

```

```
:::note
Within the template of a Vue component, you can access the route using `useRoute`.
:::
```

## ## Example

In the following example, we call an API via [useFetch](/docs/api/composables/use-fetch) using a dynamic page parameter - `slug` - as part of the URL.

```
````html [~/pages/[slug]].vue]
<script setup lang="ts">
const route = useRoute()
const { data: mountain } = await useFetch(`/api/mountains/${route.params.slug}`)
</script>

<template>
  <div>
    <h1>{{ mountain.title }}</h1>
    <p>{{ mountain.description }}</p>
  </div>
</template>
````
```

If you need to access the route query parameters (for example `example` in the path `/test?example=true`), then you can use `useRoute().query` instead of `useRoute().params`.

## ## API

Apart from dynamic parameters and query parameters, `useRoute()` also provides the following computed references related to the current route:

- `fullPath`: encoded URL associated with the current route that contains path, query and hash
- `hash`: decoded hash section of the URL that starts with a #
- `matched`: array of normalized matched routes with current route location
- `meta`: custom data attached to the record
- `name`: unique name for the route record
- `path`: encoded pathname section of the URL
- `redirectedFrom`: route location that was attempted to access before ending up on the current route location

```
:::note
Browsers don't send [URL fragments](https://url.spec.whatwg.org/#concept-url-fragment) (for example `#foo`) when making requests. So using `route.fullPath` in your template can trigger hydration issues because this will include the fragment on client but not the server.
:::
```

```
:read-more{icon="i-simple-icons-vuedotjs" to="https://router.vuejs.org/api/#RouteLocationNormalizedLoaded"}
```

```

title: "nuxt.config.ts"
description: "Nuxt can be easily configured with a single nuxt.config file."
head.title: "nuxt.config.ts"
navigation.icon: i-ph-file-duotone

```

The `nuxt.config` file extension can either be `.js`, `.ts` or `.mjs`.

```
```ts twoslash [nuxt.config.ts]
export default defineNuxtConfig({
  // My Nuxt config
})
```
```

```
::tip
`defineNuxtConfig` helper is globally available without import.
::
```

You can explicitly import `defineNuxtConfig` from `nuxt/config` if you prefer:

```
```ts twoslash [nuxt.config.ts]
import { defineNuxtConfig } from 'nuxt/config'

export default defineNuxtConfig({
  // My Nuxt config
})
```
```

```
::read-more{to="/docs/api/configuration/nuxt-config"}
Discover all the available options in the Nuxt configuration documentation.
::
```

To ensure your configuration is up to date, Nuxt will make a full restart when detecting changes in the main configuration file, the `.env` (</docs/guide/directory-structure/env>), `.nuxtignore` (</docs/guide/directory-structure/nuxtignore>) and `.nuxtrc` dotfiles.

The `.nuxtrc` file can be used to configure Nuxt with a flat syntax. It is based on `unjs/rc9` (<https://github.com/unjs/rc9>).

```
``` [.nuxtrc]
ssr=false
```
```

If present the properties in `.nuxtrc` file will overwrite the properties in the `nuxt.config` file.



```

title: 'refreshNuxtData'
description: refreshNuxtData refetches all data from the server and updates the page.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
 size: xs

::note
`refreshNuxtData` re-fetches all data from the server and updates the page as well as invalidates the cache of
[`useAsyncData`](/docs/api/composables/use-async-data) , `useLazyAsyncData`, [`useFetch`](/docs/api/composables/use-fetch)
and `useLazyFetch`.
::

Type

```ts
refreshNuxtData(keys?: string | string[])
```

Parameters:

* `keys`:

 Type: `String | String[]`

 `refreshNuxtData` accepts a single or an array of strings as `keys` that are used to fetch the data. This parameter is
 optional. All [`useAsyncData`](/docs/api/composables/use-async-data) and [`useFetch`](/docs/api/composables/use-fetch)
 are re-fetched when no `keys` are specified.

Refresh All Data

This example below refreshes all data being fetched using [`useAsyncData`](/docs/api/composables/use-async-data) and
[`useFetch`](/docs/api/composables/use-fetch) on the current page.

```vue [pages/some-page.vue]
<script setup lang="ts">
const refreshing = ref(false)
const refreshAll = async () => {
  refreshing.value = true
  try {
    await refreshNuxtData()
  } finally {
    refreshing.value = false
  }
}
</script>

<template>
  <div>
    <button :disabled="refreshing" @click="refreshAll">
      Refetch All Data
    </button>
  </div>
</template>
```

Refresh Specific Data

This example below refreshes only data where the key matches to `count`.

```vue [pages/some-page.vue]
<script setup lang="ts">
const { status, data: count } = await useLazyAsyncData('count', () => $fetch('/api/count'))
const refresh = () => refreshNuxtData('count')
</script>

<template>
  <div>
    {{ status === 'pending' ? 'Loading' : count }}
  </div>
  <button @click="refresh">Refresh</button>
</template>
```

:read-more{to="/docs/getting-started/data-fetching"}

```

```

title: "useRouter"
description: "The useRouter composable returns the router instance."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
 size: xs

```

```
```vue [pages/index.vue]
<script setup lang="ts">
const router = useRouter()
</script>
```
```

If you only need the router instance within your template, use ``$router``:

```
```vue [pages/index.vue]
<template>
  <button @click="$router.back()">Back</button>
</template>
```
```

If you have a ``pages/`` directory, ``useRouter`` is identical in behavior to the one provided by ``vue-router``.

```
::read-more{icon="i-simple-icons-vuedotjs" to="https://router.vuejs.org/api/interfaces/Router.html#Properties-currentRoute"
target="_blank"}
Read `vue-router` documentation about the `Router` interface.
::
```

## ## Basic Manipulation

- [``addRoute()``](https://router.vuejs.org/api/interfaces/Router.html#addRoute): Add a new route to the router instance. ``parentName`` can be provided to add new route as the child of an existing route.
- [``removeRoute()``](https://router.vuejs.org/api/interfaces/Router.html#removeRoute): Remove an existing route by its name.
- [``getRoutes()``](https://router.vuejs.org/api/interfaces/Router.html#getRoutes): Get a full list of all the route records.
- [``hasRoute()``](https://router.vuejs.org/api/interfaces/Router.html#hasRoute): Checks if a route with a given name exists.
- [``resolve()``](https://router.vuejs.org/api/interfaces/Router.html#resolve): Returns the normalized version of a route location. Also includes an ``href`` property that includes any existing base.

```
```ts [Example]
const router = useRouter()

router.addRoute({ name: 'home', path: '/home', component: Home })
router.removeRoute('home')
router.getRoutes()
router.hasRoute('home')
router.resolve({ name: 'home' })
```
```

```
::note
`router.addRoute()` adds route details into an array of routes and it is useful while building [Nuxt plugins]
(/docs/guide/directory-structure/plugins) while `router.push()` on the other hand, triggers a new navigation immediately and
it is useful in pages, Vue components and composable.
::
```

## ## Based on History API

- [``back()``](https://router.vuejs.org/api/interfaces/Router.html#back): Go back in history if possible, same as ``router.go(-1)``.
- [``forward()``](https://router.vuejs.org/api/interfaces/Router.html#forward): Go forward in history if possible, same as ``router.go(1)``.
- [``go()``](https://router.vuejs.org/api/interfaces/Router.html#go): Move forward or backward through the history without the hierarchical restrictions enforced in ``router.back()`` and ``router.forward()``.
- [``push()``](https://router.vuejs.org/api/interfaces/Router.html#push): Programmatically navigate to a new URL by pushing an entry in the history stack. **It is recommended to use [``navigateTo``](/docs/api/utils/navigate-to) instead.**
- [``replace()``](https://router.vuejs.org/api/interfaces/Router.html#replace): Programmatically navigate to a new URL by replacing the current entry in the routes history stack. **It is recommended to use [``navigateTo``](/docs/api/utils/navigate-to) instead.**

```
```ts [Example]
const router = useRouter()

router.back()
router.forward()
router.go(3)
router.push({ path: "/home" })
router.replace({ hash: "#bio" })
```
```

```
::read-more{icon="i-simple-icons-mdnwebdocs" color="gray" to="https://developer.mozilla.org/en-US/docs/Web/API/History"
target="_blank"}
```

Read more about the browser's History API.

::

## ## Navigation Guards

`useRouter` composable provides `afterEach`, `beforeEach` and `beforeResolve` helper methods that acts as navigation guards.

However, Nuxt has a concept of **route middleware** that simplifies the implementation of navigation guards and provides a better developer experience.

`:read-more{to="/docs/guide/directory-structure/middleware"}`

## ## Promise and Error Handling

- [`isReady()`](https://router.vuejs.org/api/interfaces/Router.html#isReady): Returns a Promise that resolves when the router has completed the initial navigation.
- [`onError`](https://router.vuejs.org/api/interfaces/Router.html#onError): Adds an error handler that is called every time a non caught error happens during navigation.

`:read-more{icon="i-simple-icons-vuedotjs" to="https://router.vuejs.org/api/interfaces/Router.html#Methods" title="Vue Router Docs" target="_blank"}`

## ## Universal Router Instance

If you do not have a `pages/` folder, then [`useRouter`](/docs/api/composables/use-router) will return a universal router instance with similar helper methods, but be aware that not all features may be supported or behave in exactly the same way as with `vue-router`.

```

title: 'reloadNuxtApp'
description: reloadNuxtApp will perform a hard reload of the page.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/chunk.ts
 size: xs

::note
`reloadNuxtApp` will perform a hard reload of your app, re-requesting a page and its dependencies from the server.
::

By default, it will also save the current `state` of your app (that is, any state you could access with `useState`).

::read-more{to="/docs/guide/going-further/experimental-features#restorestate" icon="i-ph-star-duotone"}
You can enable experimental restoration of this state by enabling the `experimental.restoreState` option in your
`nuxt.config` file.
::

Type

```ts
reloadNuxtApp(options?: ReloadNuxtAppOptions)

interface ReloadNuxtAppOptions {
  ttl?: number
  force?: boolean
  path?: string
  persistState?: boolean
}
```

`options` (optional)

Type: `ReloadNuxtAppOptions`

An object accepting the following properties:

- `path` (optional)

 Type: `string`

 Default: `window.location.pathname`

 The path to reload (defaulting to the current path). If this is different from the current window location it will trigger a navigation and add an entry in the browser history.

- `ttl` (optional)

 Type: `number`

 Default: `10000`

 The number of milliseconds in which to ignore future reload requests. If called again within this time period, `reloadNuxtApp` will not reload your app to avoid reload loops.

- `force` (optional)

 Type: `boolean`

 Default: `false`

 This option allows bypassing reload loop protection entirely, forcing a reload even if one has occurred within the previously specified TTL.

- `persistState` (optional)

 Type: `boolean`

 Default: `false`

 Whether to dump the current Nuxt state to sessionStorage (as `nuxt:reload:state`). By default this will have no effect on reload unless `experimental.restoreState` is also set, or unless you handle restoring the state yourself.

```

```

title: 'setLayout'
description: setLayout allows you to dynamically change the layout of a page.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/router.ts
 size: xs

```

```
::important
`setLayout` allows you to dynamically change the layout of a page. It relies on access to the Nuxt context and therefore
can only be called within the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context).
::
```

```
```ts [middleware/custom-layout.ts]
export default defineNuxtRouteMiddleware((to) => {
  // Set the layout on the route you are navigating _to_
  setLayout('other')
})
```
```

```
::note
If you choose to set the layout dynamically on the server side, you must do so before the layout is rendered by Vue (that
is, within a plugin or route middleware) to avoid a hydration mismatch.
::
```

```

title: package.json
head.title: package.json
description: The package.json file contains all the dependencies and scripts for your application.
navigation.icon: i-ph-file-duotone

```

The minimal `package.json` of your Nuxt application should looks like:

```
```json [package.json]
{
  "name": "nuxt-app",
  "private": true,
  "type": "module",
  "scripts": {
    "build": "nuxt build",
    "dev": "nuxt dev",
    "generate": "nuxt generate",
    "preview": "nuxt preview",
    "postinstall": "nuxt prepare"
  },
  "devDependencies": {
    "@nuxt/devtools": "latest",
    "nuxt": "latest",
    "vue": "latest",
    "vue-router": "latest"
  }
}
```
```

```
::read-more{icon="i-simple-icons-npm" color="gray" to="https://docs.npmjs.com/cli/configuring-npm/package-json"
target="_blank"}
Read more about the `package.json` file.
::
```

```

title: 'useRuntimeConfig'
description: 'Access runtime config variables with the useRuntimeConfig composable.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/asyncData.ts
 size: xs

```

## ## Usage

```

```vue [app.vue]
<script setup lang="ts">
const config = useRuntimeConfig()
</script>
```

```

```

```ts [server/api/foo.ts]
export default defineEventHandler((event) => {
  const config = useRuntimeConfig(event)
})
```

```

:read-more{to="/docs/guide/going-further/runtime-config"}

## ## Define Runtime Config

The example below shows how to set a public API base URL and a secret API token that is only accessible on the server.

We should always define `runtimeConfig` variables inside `nuxt.config`.

```

```ts [nuxt.config.ts]
export default defineNuxtConfig({
  runtimeConfig: {
    // Private keys are only available on the server
    apiSecret: '123',

    // Public keys that are exposed to the client
    public: {
      apiBase: process.env.NUXT_PUBLIC_API_BASE || '/api'
    }
  }
})
```

```

::note

Variables that need to be accessible on the server are added directly inside `runtimeConfig`. Variables that need to be accessible on both the client and the server are defined in `runtimeConfig.public`.

::

:read-more{to="/docs/guide/going-further/runtime-config"}

## ## Access Runtime Config

To access runtime config, we can use `useRuntimeConfig()` composable:

```

```ts [server/api/test.ts]
export default defineEventHandler((event) => {
  const config = useRuntimeConfig(event)

  // Access public variables
  const result = await $fetch(`/test`, {
    baseURL: config.public.apiBase,
    headers: {
      // Access a private variable (only available on the server)
      Authorization: `Bearer ${config.apiSecret}`
    }
  })
  return result
})
```

```

In this example, since `apiBase` is defined within the `public` namespace, it is universally accessible on both server and client-side, while `apiSecret` **is only accessible on the server-side**.

## ## Environment Variables

It is possible to update runtime config values using a matching environment variable name prefixed with `NUXT\_`.

:read-more{to="/docs/guide/going-further/runtime-config"}

## ### Using the `.env` File

We can set the environment variables inside the ``.env`` file to make them accessible during **development** and **build/generate**.

```
```.env
Nuxt_PUBLIC_API_BASE = "https://api.localhost:5555"
Nuxt_API_SECRET = "123"
```.
```

**note**

Any environment variables set within ``.env`` file are accessed using `process.env`` in the Nuxt app during **development** and **build/generate**.

**warning**

In **production runtime**, you should use platform environment variables and ``.env`` is not used.

[:read-more{to="/docs/guide/directory-structure/env"}](/docs/guide/directory-structure/env)

### `app`` namespace

Nuxt uses `app`` namespace in runtime-config with keys including `baseUrl`` and `cdnURL``. You can customize their values at runtime by setting environment variables.

**note**

This is a reserved namespace. You should not introduce additional keys inside `app``.

### `app.baseUrl``

By default, the `baseUrl`` is set to `''/``.

However, the `baseUrl`` can be updated at runtime by setting the `Nuxt_APP_BASE_URL`` as an environment variable.

Then, you can access this new base URL using `config.app.baseUrl``:

```
```.ts [plugins/my-plugin.ts]
export default defineNuxtPlugin((NuxtApp) => {
  const config = useRuntimeConfig()

  // Access baseUrl universally
  const baseUrl = config.app.baseUrl
})
```.
```

### `app.cdnURL``

This example shows how to set a custom CDN url and access them using `useRuntimeConfig()``.

You can use a custom CDN for serving static assets inside `.output/public`` using the `Nuxt_APP_CDN_URL`` environment variable.

And then access the new CDN url using `config.app.cdnURL``.

```
```.ts [server/api/foo.ts]
export default defineEventHandler((event) => {
  const config = useRuntimeConfig(event)

  // Access cdnURL universally
  const cdnURL = config.app.cdnURL
})
```.
```

[:read-more{to="/docs/guide/going-further/runtime-config"}](/docs/guide/going-further/runtime-config)



```

title: "tsconfig.json"
description: "Nuxt generates a .nuxt/tsconfig.json file with sensible defaults and your aliases."
head.title: "tsconfig.json"
navigation.icon: i-ph-file-duotone

```

Nuxt [automatically generates](/docs/guide/concepts/typescript) a `.nuxt/tsconfig.json` file with the resolved aliases you are using in your Nuxt project, as well as with other sensible defaults.

You can benefit from this by creating a `tsconfig.json` in the root of your project with the following content:

```
```json [tsconfig.json]
{
  "extends": "./.nuxt/tsconfig.json"
}
```
```

::note

As you need to, you can customize the contents of this file. However, it is recommended that you don't overwrite `target`, `module` and `moduleResolution`.

::

::note

If you need to customize your `paths`, this will override the auto-generated path aliases. Instead, we recommend that you add any path aliases you need to the `[alias](/docs/api/nuxt-config#alias)` property within your `nuxt.config`, where they will get picked up and added to the auto-generated `tsconfig`.

::

```

title: 'Framework'
navigation.icon: i-ph-github-logo-duotone
description: Some specific points about contributions to the framework repository.

```

Once you've read the [general contribution guide](/docs/community/contribution), here are some specific points to make about contributions to the [nuxt/nuxt](https://github.com/nuxt/nuxt) repository.

## ## Monorepo Guide

- ``packages/kit``: Toolkit for authoring Nuxt Modules, published as [nuxt/kit](https://npmjs.com/package/@nuxt/kit).
- ``packages/nuxt``: The core of Nuxt, published as [nuxt](https://npmjs.com/package/nuxt).
- ``packages/schema``: Cross-version Nuxt typedefs and defaults, published as [nuxt/schema](https://npmjs.com/package/@nuxt/schema).
- ``packages/test-utils``: Test utilities for Nuxt, published as [nuxt/test-utils](https://npmjs.com/package/@nuxt/test-utils).
- ``packages/vite``: The [Vite](https://vitejs.dev) bundler for Nuxt, published as [nuxt/vite-builder](https://npmjs.com/package/@nuxt/vite-builder).
- ``packages/webpack``: The [webpack](https://webpack.js.org) bundler for Nuxt 3, published as [nuxt/webpack-builder](https://npmjs.com/package/@nuxt/webpack-builder).

## ## Setup

To contribute to Nuxt, you need to set up a local environment.

1. [Fork](https://help.github.com/articles/fork-a-repo) the [nuxt/nuxt](https://github.com/nuxt/nuxt) repository to your own GitHub account and then [clone](https://help.github.com/articles/cloning-a-repository) it to your local device.
2. Ensure using the latest [Node.js](https://nodejs.org/en) (20.x)
3. Enable [Corepack](https://github.com/nodejs/corepack) to have ``pnpm`` and ``yarn``

```
```bash [Terminal]
corepack enable
```
```
4. Run ``pnpm install`` to Install the dependencies with pnpm:

```
```bash [Terminal]
pnpm install
```

::note
If you are adding a dependency, please use `pnpm add`. :br
The `pnpm-lock.yaml` file is the source of truth for all Nuxt dependencies.
::
```
5. Activate the passive development system

```
```bash [Terminal]
pnpm dev:prepare
```
```
6. Check out a branch where you can work and commit your changes:

```
```bash [Terminal]
git checkout -b my-new-branch
```
```

Then, test your changes against the [playground](#playground) and [test](#testing) your changes before submitting a pull request.

## ### Playground

While working on a pull request, you will likely want to check if your changes are working correctly.

You can modify the example app in ``playground/``, and run:

```
```bash [Terminal]
pnpm dev
```
```

```
::important
Please make sure not to commit it to your branch, but it could be helpful to add some example code to your PR description.
This can help reviewers and other Nuxt users understand the feature you've built in-depth.
::
```

## ### Testing

Every new feature should have a corresponding unit test (if possible). The ``test/`` directory in this repository is currently a work in progress, but do your best to create a new test following the example of what's already there.

Before creating a PR or marking it as ready-to-review, ensure that all tests pass by running:

```
```bash [Terminal]
pnpm test
```
```

## ### Linting

You might have noticed already that we use ESLint to enforce a coding standard.

Before committing your changes, to verify that the code style is correct, run:

```
```bash [Terminal]
pnpm lint
```
```

```
::note
You can use `pnpm lint --fix` to fix most of the style changes. :br
If there are still errors left, you must correct them manually.
::
```

### ### Documentation

If you are adding a new feature or refactoring or changing the behavior of Nuxt in any other manner, you'll likely want to document the changes. Please include any changes to the docs in the same PR. You don't have to write documentation up on the first commit (but please do so as soon as your pull request is mature enough).

```
::important
Make sure to make changes according to the [Documentation Style Guide](/docs/community/contribution#documentation-style-guide).
::
```

### ### Final Checklist

When submitting your PR, there is a simple template that you have to fill out. Please tick all appropriate "answers" in the checklists.

### ## Documentation Guide

If you spot an area where we can improve documentation or error messages, please do open a PR - even if it's just to fix a typo!

```
::important
Make sure to make changes according to the [Documentation Style Guide](/docs/community/contribution#documentation-style-guide).
::
```

### ### Quick Edits

If you spot a typo or want to rephrase a sentence, you can click on the **Edit this page** link located on the right aside in the **Community** section.

Make the change directly in the GitHub interface and open a Pull Request.

### ### Longer Edits

The documentation content is inside the `docs/` directory of the [nuxt/nuxt](https://github.com/nuxt/nuxt) repository and written in markdown.

```
::note
To preview the docs locally, follow the steps on [nuxt/nuxt.com](https://github.com/nuxt/nuxt.com) repository.
::
```

```
::note
We recommend that you install the [MDC extension](https://marketplace.visualstudio.com/items?itemName=Nuxt.mdc) for VS Code.
::
```

### ### Linting Docs

Documentation is linted using [MarkdownLint](https://github.com/DavidAnson/markdownlint) and [case police](https://github.com/antfu/case-police) to keep the documentation cohesive.

```
```bash [Terminal]
pnpm lint:docs
```
```

```
::note
You can also run `pnpm lint:docs:fix` to highlight and resolve any lint issues.
::
```

### ### Open a PR

Please make sure your PR title adheres to the [conventional commits](https://www.conventionalcommits.org) guidelines.

```
```bash [Example of PR title]
docs: update the section about the nuxt.config.ts file
```
```

```

title: 'Roadmap'
description: 'Nuxt is constantly evolving, with new features and modules being added all the time.'
navigation.icon: i-ph-map-trifold-duotone

::read-more{to="/blog"}
See our blog for the latest framework and ecosystem announcements.
::

Status Reports

::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/issues/13653" target="_blank"}
Documentation Progress
::
::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/discussions/16119" target="_blank"}
Rendering Optimizations: Today and Tomorrow
::
::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/image/discussions/563" target="_blank"}
Nuxt Image: Performance and Status
::

Roadmap

In roadmap below are some features we are planning or working on at the moment.

::tip
Check [Discussions](https://github.com/nuxt/nuxt/discussions) and [RFCs]
(https://github.com/nuxt/nuxt/discussions/categories/rfcs) for more upcoming features and ideas.
::

Milestone	Expected date	Notes	Description
-
SEO & PWA | 2024 | [nuxt/nuxt#18395](https://github.com/nuxt/nuxt/discussions/18395) | Migrating from [nuxt-
community/pwa-module](https://github.com/nuxt-community/pwa-module) for built-in SEO utils and service worker support
Assets | 2024 | [nuxt/nuxt#22012](https://github.com/nuxt/nuxt/discussions/22012) | Allow developers and
modules to handle loading third-party assets.
Translations | - | [nuxt/translations#4](https://github.com/nuxt/translations/discussions/4) ([request access]
(https://github.com/nuxt/nuxt/discussions/16054)) | A collaborative project for a stable translation process for Nuxt docs.
Currently pending for ideas and documentation tooling support (content v2 with remote sources).

Core Modules Roadmap

In addition to the Nuxt framework, there are modules that are vital for the ecosystem. Their status will be updated below.

Module	Status	Nuxt Support	Repository	Description
[Scripts](https://scripts.nuxt.com) | Public Preview | 3.x | [nuxt/scripts](https://github.com/nuxt/scripts) | Easy 3rd party script management.
Ally | Planned | 3.x | `nuxt/ally` to be announced | Accessibility
hinting and utilities [nuxt/nuxt#23255](https://github.com/nuxt/nuxt/issues/23255)
Auth | Planned | 3.x | `nuxt/auth` to be announced | Support is planned
after session support.
Hints | Planned | 3.x | `nuxt/hints` to be announced | Guidance and
suggestions for enhancing development practices.

Release Cycle

Since January 2023, we've adopted a consistent release cycle for Nuxt 3, following [semver](https://semver.org). We aim
for major framework releases every year, with an expectation of patch releases every week or so and minor releases every
month or so. They should never contain breaking changes except within options clearly marked as `experimental`.

Ongoing Support for Nuxt

Going forward from v3, we commit to support each major version of Nuxt for a minimum of a year after the last release, and to
providing an upgrade path for current users at that point.

Current Packages

The current active version of [Nuxt](https://nuxt.com) is v3 which is available as `nuxt` on npm with the `latest` tag.

Nuxt 2 is in maintenance mode and is available on npm with the `2x` tag. It will reach End of Life (EOL) on June 30, 2024.

Each active version has its own nightly releases which are generated automatically. For more about enabling the Nuxt nightly
release channel, see [the nightly release channel docs](/docs/guide/going-further/nightly-release-channel).

Release |
| Initial release | End Of Life | Docs
-----|-----|-----|-----
4.x (scheduled) |
| 2024 Q3 |
```

**3.x** (stable) | [</a> | 2022-11-16 | TBA |](https://npmjs.com/package/nuxt)

**2.x** (unsupported) | [</a> | 2018-09-21 | 2024-06-30 |](https://www.npmjs.com/package/nuxt?activeTab=versions)

**1.x** (unsupported) | [</a> | 2018-01-08 | 2019-09-21 | &nbsp;](https://www.npmjs.com/package/nuxt?activeTab=versions)

Support Status

| Status      | Description                                                                   |
|-------------|-------------------------------------------------------------------------------|
| Unsupported | This version is not maintained any more and will not receive security patches |
| Maintenance | This version will only receive security patches                               |
| Stable      | This version is being developed for and will receive security patches         |
| Development | This version could be unstable                                                |
| Scheduled   | This version does not exist yet but is planned                                |

```

title: 'Releases'
description: Discover the latest releases of Nuxt & Nuxt official modules.
navigation.icon: i-ph-notification-duotone

::card-group
 ::card

 icon: i-simple-icons-github
 title: nuxt/nuxt
 to: https://github.com/nuxt/nuxt/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt framework releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/cli
 to: https://github.com/nuxt/cli/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt CLI (`nuxi`) releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/content
 to: https://github.com/nuxt/content/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt Content releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/devtools
 to: https://github.com/nuxt/devtools/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt DevTools releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/fonts
 to: https://github.com/nuxt/fonts/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt Fonts releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/image
 to: https://github.com/nuxt/image/releases
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt Image releases.
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/scripts
 to: https://github.com/nuxt/scripts/tags
 target: _blank
 ui.icon.base: text-black dark:text-white

 Nuxt Scripts releases. (Public Preview)
 ::
 ::card

 icon: i-simple-icons-github
 title: nuxt/ui
 to: https://github.com/nuxt/ui/releases
 target: _blank
```

```
ui.icon.base: text-black dark:text-white

Nuxt UI releases.
::
::

::read-more{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt" target="_blank"}
Discover the `nuxt` organization on GitHub
::
```

```

title: 'nuxi prepare'
description: The prepare command creates a .nuxt directory in your application and generates types.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/prepare.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi prepare [--log-level] [rootDir]
```
```

The `prepare` command creates a `[.nuxt]`(/docs/guide/directory-structure/nuxt) directory in your application and generates types. This can be useful in a CI environment or as a `postinstall` command in your `[package.json]`(/docs/guide/directory-structure/package).

| Option               | Default          | Description                                       |
|----------------------|------------------|---------------------------------------------------|
| ----- ----- -----    |                  |                                                   |
| <code>rootDir</code> | <code>`.`</code> | The root directory of the application to prepare. |



```

title: "nuxi module"
description: "Search and add modules to your Nuxt application with the command line."
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/module/
 size: xs

```

Nuxi provides a few utilities to work with [Nuxt modules](/modules) seamlessly.

## nuxi module add

```
```bash [Terminal]
npx nuxi module add <NAME>
```
```

| Option            | Default | Description                        |
|-------------------|---------|------------------------------------|
| ----- ----- ----- |         |                                    |
| `NAME`            | -       | The name of the module to install. |

The command lets you install [Nuxt modules](/modules) in your application with no manual work.

- When running the command, it will:
- install the module as a dependency using your package manager
  - add it to your [package.json](/docs/guide/directory-structure/package) file
  - update your [nuxt.config](/docs/guide/directory-structure/nuxt-config) file

**Example:**

```
Installing the [Pinia](/modules/pinia) module
```bash [Terminal]
npx nuxi module add pinia
```
```

## nuxi module search

```
```bash [Terminal]
npx nuxi module search <QUERY>
```
```

| Option            | Default | Description                           |
|-------------------|---------|---------------------------------------|
| ----- ----- ----- |         |                                       |
| `QUERY`           | -       | The name of the module to search for. |

The command searches for Nuxt modules matching your query that are compatible with your Nuxt version.

**Example:**

```
```bash [Terminal]
npx nuxi module search pinia
```
```

```

title: "nuxi init"
description: The init command initializes a fresh Nuxt project.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/cli/blob/main/src/commands/init.ts
 size: xs

```

```
```bash [Terminal]
npx nuxi init [--verbose|-v] [--template,-t] [dir]
```
```

The `init` command initializes a fresh Nuxt project using `[unjs/giget]`(<https://github.com/unjs/giget>).

## Options

| Option                         | Default            | Description                                                                                                                       |
|--------------------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| ----- ----- -----              |                    |                                                                                                                                   |
| <code>--cwd</code>             |                    | Current working directory                                                                                                         |
| <code>--log-level</code>       |                    | Log level                                                                                                                         |
| <code>--template, -t</code>    | <code>v3</code>    | Specify template name or git repository to use as a template. Format is <code>gh:org/name</code> to use a custom github template. |
| <code>--force, -f</code>       | <code>false</code> | Force clone to any existing directory.                                                                                            |
| <code>--offline</code>         | <code>false</code> | Force offline mode (do not attempt to download template from GitHub and only use local cache).                                    |
| <code>--prefer-offline</code>  | <code>false</code> | Prefer offline mode (try local cache first to download templates).                                                                |
| <code>--no-install</code>      | <code>false</code> | Skip installing dependencies.                                                                                                     |
| <code>--git-init</code>        | <code>false</code> | Initialize git repository.                                                                                                        |
| <code>--shell</code>           | <code>false</code> | Start shell after installation in project directory (experimental).                                                               |
| <code>--package-manager</code> | <code>npm</code>   | Package manager choice (npm, pnpm, yarn, bun).                                                                                    |
| <code>--dir</code>             |                    | Project directory.                                                                                                                |

## Environment variables

- `NUXI_INIT_REGISTRY`: Set to a custom template registry. ([learn more](<https://github.com/unjs/giget#custom-registry>)).
- Default registry is loaded from `[nuxt/starter/templates]`(<https://github.com/nuxt/starter/tree/templates/templates>)

```

title: 'useSeoMeta'
description: The useSeoMeta composable lets you define your site's SEO meta tags as a flat object with full TypeScript support.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/unjs/unhead/blob/main/packages/unhead/src/composables/useSeoMeta.ts
 size: xs

```

This helps you avoid common mistakes, such as using `name` instead of `property`, as well as typos - with over 100+ meta tags fully typed.

```

::important
This is the recommended way to add meta tags to your site as it is XSS safe and has full TypeScript support.
::

```

```

:read-more{to="/docs/getting-started/seo-meta"}

```

## ## Usage

```

```vue [app.vue]
<script setup lang="ts">
useSeoMeta({
  title: 'My Amazing Site',
  ogTitle: 'My Amazing Site',
  description: 'This is my amazing site, let me tell you all about it.',
  ogDescription: 'This is my amazing site, let me tell you all about it.',
  ogImage: 'https://example.com/image.png',
  twitterCard: 'summary_large_image',
})
</script>
```

```

When inserting tags that are reactive, you should use the computed getter syntax (`() => value`):

```

```vue [app.vue]
<script setup lang="ts">
const title = ref('My title')

useSeoMeta({
  title,
  description: () => `description: ${title.value}`
})
</script>
```

```

## ## Parameters

There are over 100 parameters. See the [full list of parameters in the source code](https://github.com/harlan-zw/zhead/blob/main/packages/zhead/src/metaFlat.ts#L1035).

```

:read-more{to="/docs/getting-started/seo-meta"}

```

```

title: 'updateAppConfig'
description: 'Update the App Config at runtime.'
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/config.ts
 size: xs

```

```
:::note
Updates the [app.config](/docs/guide/directory-structure/app-config) using deep assignment. Existing (nested) properties
will be preserved.
:::
```

## ## Usage

```
```js
const appConfig = useAppConfig() // { foo: 'bar' }

const newAppConfig = { foo: 'baz' }

updateAppConfig(newAppConfig)

console.log(appConfig) // { foo: 'baz' }
```

:read-more{to="/docs/guide/directory-structure/app-config"}
```

```

title: 'setResponseStatus'
description: setResponseStatus sets the statusCode (and optionally the statusMessage) of the response.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/ssr.ts
 size: xs

```

Nuxt provides composables and utilities for first-class server-side-rendering support.

`setResponseStatus` sets the statusCode (and optionally the statusMessage) of the response.

```
::important
`setResponseStatus` can only be called in the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context).
::
```

```
```js
const event = useRequestEvent()

// event will be undefined in the browser
if (event) {
  // Set the status code to 404 for a custom 404 page
  setResponseStatus(event, 404)

  // Set the status message as well
  setResponseStatus(event, 404, 'Page Not Found')
}
```
```

```
::note
In the browser, setResponseStatus will have no effect.
::
```

```
:read-more{to="/docs/getting-started/error-handling"}
```

```

title: 'useServerSeoMeta'
description: The useServerSeoMeta composable lets you define your site's SEO meta tags as a flat object with full TypeScript support.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/unjs/unhead/blob/main/packages/unhead/src/composables/useServerSeoMeta.ts
 size: xs

```

Just like [`useSeoMeta`](/docs/api/composables/use-seo-meta), `useServerSeoMeta` composable lets you define your site's SEO meta tags as a flat object with full TypeScript support.

`:read-more{to="/docs/api/composables/use-seo-meta"}`

In most instances, the meta doesn't need to be reactive as robots will only scan the initial load. So we recommend using [`useServerSeoMeta`](/docs/api/composables/use-server-seo-meta) as a performance-focused utility that will not do anything (or return a `head` object) on the client.

```
```vue [app.vue]
<script setup lang="ts">
useServerSeoMeta({
  robots: 'index, follow'
})
</script>
```
```

Parameters are exactly the same as with [`useSeoMeta`](/docs/api/composables/use-seo-meta)

`:read-more{to="/docs/getting-started/seo-meta"}`

```

title: 'showError'
description: Nuxt provides a quick and simple way to show a full screen error page if needed.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/error.ts
 size: xs

```

Within the [Nuxt context](/docs/guide/going-further/nuxt-app#the-nuxt-context) you can use `showError` to show an error.

**Parameters:**

```
- `error`: `string | Error | Partial<{ cause, data, message, name, stack, statusCode, statusMessage }>`

```ts
showError("ðŸ˜± Oh no, an error has been thrown.")
showError({
  statusCode: 404,
  statusMessage: "Page Not Found"
})
```
```

The error is set in the state using [`useError()`](/docs/api/composables/use-error) to create a reactive and SSR-friendly shared error state across components.

```
:::tip
`showError` calls the `app:error` hook.
:::
```

```
:read-more{to="/docs/getting-started/error-handling"}
```

```

title: "useState"
description: The useState composable creates a reactive and SSR-friendly shared state.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/composables/state.ts
 size: xs

```

## ## Usage

```
```:ts
// Create a reactive state and set default value
const count = useState('counter', () => Math.round(Math.random() * 100))
```

:read-more{to="/docs/getting-started/state-management"}

::important
Because the data inside `useState` will be serialized to JSON, it is important that it does not contain anything that cannot be serialized, such as classes, functions or symbols.
::

::warning
`useState` is a reserved function name transformed by the compiler, so you should not name your own function `useState`.
::

::tip{icon="i-ph-video-duotone" to="https://www.youtube.com/watch?v=mv0WcBABcIk" target="_blank"}
Watch a video from Alexander Lichter about why and when to use `useState`.
::
```

## ## Using `shallowRef`

If you don't need your state to be deeply reactive, you can combine `useState` with [`shallowRef`] (<https://vuejs.org/api/reactivity-advanced.html#shallowref>). This can improve performance when your state contains large objects and arrays.

```
```:ts
const state = useState('my-shallow-state', () => shallowRef({ deep: 'not reactive' }))
// isShallow(state) === true
```
```

## ## Type

```
```:ts
useState<T>(init?: () => T | Ref<T>): Ref<T>
useState<T>(key: string, init?: () => T | Ref<T>): Ref<T>
```
```

- `key`: A unique key ensuring that data fetching is properly de-duplicated across requests. If you do not provide a key, then a key that is unique to the file and line number of the instance of [`useState`] (</docs/api/composables/use-state>) will be generated for you.
- `init`: A function that provides initial value for the state when not initiated. This function can also return a `Ref`.
- `T`: (typescript only) Specify the type of state



```

title: Configuration
description: 'Learn how to configure Nuxt Bridge to your own needs.'

```

## ## Feature Flags

You can optionally disable some features from bridge or opt-in to less stable ones. In normal circumstances, it is always best to stick with defaults!

You can check [bridge/src/module.ts](https://github.com/nuxt/bridge/blob/main/packages/bridge/src/module.ts) for latest defaults.

```
``ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'
export default defineNuxtConfig({
 bridge: {

 // -- Opt-in features --

 // Use Vite as the bundler instead of webpack 4
 // vite: true,

 // Enable Nuxt 3 compatible useHead
 // meta: true,

 // Enable definePageMeta macro
 // macros: {
 // pageMeta: true
 // },

 // Enable transpiling TypeScript with esbuild
 // typescript: {
 // esbuild: true
 // },

 // -- Default features --

 // Use legacy server instead of Nitro
 // nitro: false,

 // Disable Nuxt 3 compatible `nuxtApp` interface
 // app: false,

 // Disable Composition API support
 // capi: false,

 // ... or just disable legacy Composition API support
 // capi: {
 // legacy: false
 // },

 // Do not transpile modules
 // transpile: false,

 // Disable <script setup> support
 // scriptSetup: false,

 // Disable composables auto importing
 // imports: false,

 // Do not warn about module incompatibilities
 // constraints: false
 },

 vite: {
 // Config for Vite
 }
})
``
```

## ## Migration of each option

### ### router.base

```
``diff
export default defineNuxtConfig({
- router: {
- base: '/my-app/'
- }
+ app: {
+ baseURL: '/my-app/'
+ }
```

```
})
```
```

```
### build.publicPath
```

```
```diff  
export default defineNuxtConfig({
- build: {
- publicPath: 'https://my-cdn.net'
- }
+ app: {
+ cdnURL: 'https://my-cdn.net'
+ }
})
```
```

```
---
title: TypeScript
description: 'Learn how to use TypeScript with Nuxt Bridge.'
---
```

Remove Modules

- Remove `@nuxt/typescript-build`: Bridge enables same functionality
- Remove `@nuxt/typescript-runtime` and `nuxt-ts`: Nuxt 2 has built-in runtime support

Set `bridge.typescript`

```
``ts
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
  bridge: {
    typescript: true,
    nitro: false // If migration to Nitro is complete, set to true
  }
})
``
```

Update `tsconfig.json`

If you are using TypeScript, you can edit your `tsconfig.json` to benefit from auto-generated Nuxt types:

```
``diff [tsconfig.json]
{
+ "extends": "../.nuxt/tsconfig.json",
  "compilerOptions": {
    ...
  }
}
``
```

::note

As `./.nuxt/tsconfig.json` is generated and not checked into version control, you'll need to generate that file before running your tests. Add `nuxi prepare` as a step before your tests, otherwise you'll see `TS5083: Cannot read file '~/nuxt/tsconfig.json'`

::

::note

Keep in mind that all options extended from `./.nuxt/tsconfig.json` will be overwritten by the options defined in your `tsconfig.json`.

Overwriting options such as `"compilerOptions.paths"` with your own configuration will lead TypeScript to not factor in the module resolutions from `./.nuxt/tsconfig.json`. This can lead to module resolutions such as `#imports` not being recognized.

In case you need to extend options provided by `./.nuxt/tsconfig.json` further, you can use the `alias` property within your `nuxt.config`. `nuxi` will pick them up and extend `./.nuxt/tsconfig.json` accordingly.

::

```
---
title: Overview
description: Reduce the differences with Nuxt 3 and reduce the burden of migration to Nuxt 3.
---

::note
If you're starting a fresh Nuxt 3 project, please skip this section and go to [Nuxt 3 Installation](/docs/getting-started/introduction).
::

::warning
Nuxt Bridge provides identical features to Nuxt 3 ([docs](/docs/guide/concepts/auto-imports)) but there are some limitations, notably that [useAsyncData](/docs/api/composables/use-async-data) and [useFetch](/docs/api/composables/use-fetch) composables are not available. Please read the rest of this page for details.
::

Bridge is a forward-compatibility layer that allows you to experience many of the new Nuxt 3 features by simply installing and enabling a Nuxt module.

Using Nuxt Bridge, you can make sure your project is (almost) ready for Nuxt 3 and you can gradually proceed with the transition to Nuxt 3.

## First Step

### Upgrade Nuxt 2

Make sure your dev server (nuxt dev) isn't running, remove any package lock files (package-lock.json and yarn.lock), and install the latest Nuxt 2 version:



```
diff [package.json]
- "nuxt": "^2.16.3"
+ "nuxt": "^2.17.3"
```



Then, reinstall your dependencies:

::code-group



```
bash [yarn]
yarn install

bash [npm]
npm install
```



::

::note
Once the installation is complete, make sure both development and production builds are working as expected before proceeding.
::

### Install Nuxt Bridge

Install @nuxt/bridge and nuxi as development dependencies:

::code-group



```
bash [Yarn]
yarn add --dev @nuxt/bridge nuxi

bash [npm]
npm install -D @nuxt/bridge nuxi
```



::

### Update nuxt.config

Please make sure to avoid any CommonJS syntax such as module.exports, require or require.resolve in your config file. It will soon be deprecated and unsupported.

You can use static import, dynamic import() and export default instead. Using TypeScript by renaming to [nuxt.config.ts](/docs/guide/directory-structure/nuxt-config) is also possible and recommended.



```
ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
 bridge: false
})
```


```

...

Update Commands

The ``nuxt`` command should now be changed to the ``nuxt2`` command.

```
```diff
{
 "scripts": {
 - "dev": "nuxt",
 + "dev": "nuxt2",
 - "build": "nuxt build",
 + "build": "nuxt2 build",
 - "start": "nuxt start",
 + "start": "nuxt2 start"
 }
}
```
```

Try running ``nuxt2`` once here. You will see that the application works as before.

(If `'bridge'` is set to `false`, your application will operate without any changes as before.)

Upgrade Steps

With Nuxt Bridge, the migration to Nuxt 3 can proceed in steps. The below ``Upgrade Steps`` does not need to be done all at once.

- [TypeScript](/docs/bridge/typescript)
- [Migrate Legacy Composition API](/docs/bridge/bridge-composition-api)
- [Plugins and Middleware](/docs/bridge/plugins-and-middleware)
- [Migrate New Composition API](/docs/bridge/nuxt3-compatible-api)
- [Meta Tags](/docs/bridge/meta)
- [Runtime Config](/docs/bridge/runtime-config)
- [Nitro](/docs/bridge/nitro)
- [Vite](/docs/bridge/vite)

Migrate from CommonJS to ESM

Nuxt 3 natively supports TypeScript and ECMAScript Modules. Please check [Native ES Modules](/docs/guide/concepts/esm) for more info and upgrading.

title: Overview

description: Nuxt 3 is a complete rewrite of Nuxt 2, and also based on a new set of underlying technologies.

There are significant changes when migrating a Nuxt 2 app to Nuxt 3, although you can expect migration to become more straightforward as we move toward a stable release.

::note

This migration guide is under progress to align with the development of Nuxt 3.

::

Some of these significant changes include:

1. Moving from Vue 2 to Vue 3, including defaulting to the Composition API and script setup.
1. Moving from webpack 4 and Babel to Vite or webpack 5 and esbuild.
1. Moving from a runtime Nuxt dependency to a minimal, standalone server compiled with nitropack.

::tip

If you need to remain on Nuxt 2, but want to benefit from Nuxt 3 features in Nuxt 2, you can alternatively check out [how to get started with Bridge](/docs/bridge/overview).

::

Next Steps

- Learn about differences in [configuration](/docs/migration/configuration)

```
---
title: Server
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 server.'
---
```

In a built Nuxt 3 application, there is no runtime Nuxt dependency. That means your site will be highly performant, and ultra-slim. But it also means you can no longer hook into runtime Nuxt server hooks.

```
:read-more{to="/docs/guide/concepts/server-engine"}
```

Steps

1. Remove the `render` key in your `nuxt.config`.
2. Any files in `~/server/api` and `~/server/middleware` will be automatically registered; you can remove them from your `serverMiddleware` array.
3. Update any other items in your `serverMiddleware` array to point to files or npm packages directly, rather than using inline functions.

```
:read-more{to="/docs/guide/directory-structure/server"}
:read-more{to="/docs/guide/going-further/hooks#server-hooks-runtime"}
```

```
---
title: Build Tooling
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 build tooling.'
---
```

We use the following build tools by default:

- [Vite](https://vitejs.dev) or [webpack](https://webpack.js.org)
- [Rollup](https://rollupjs.org)
- [PostCSS](https://postcss.org)
- [esbuild](https://esbuild.github.io)

For this reason, most of your previous `build` configuration in `nuxt.config` will now be ignored, including any custom babel configuration.

If you need to configure any of Nuxt's build tools, you can do so in your `nuxt.config`, using the new top-level `vite`, `webpack` and `postcss` keys.

In addition, Nuxt ships with TypeScript support.

`:read-more{to="/docs/guide/concepts/typescript"}`

Steps

1. Remove `@nuxt/typescript-build` and `@nuxt/typescript-runtime` from your dependencies and modules.
2. Remove any unused babel dependencies from your project.
3. Remove any explicit core-js dependencies.
4. Migrate `require` to `import`.

`<!-- TODO: Enabling webpack builder -->`


```
---
title: "nuxi upgrade"
description: The upgrade command upgrades Nuxt to the latest version.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/upgrade.ts
    size: xs
---
```

```
```bash [Terminal]
npx nuxi upgrade [--force|-f]
```
```

The `upgrade` command upgrades Nuxt to the latest version.

| Option | Default | Description |
|-------------------|---------|---|
| ----- ----- ----- | | |
| --force, -f | false | Removes `node_modules` and lock files before upgrade. |

```
---
title: "nuxi typecheck"
description: The typecheck command runs vue-tsc to check types throughout your app.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/typecheck.ts
    size: xs
---
```

```
```bash [Terminal]
npx nuxi typecheck [--log-level] [rootDir]
```
```

The `typecheck` command runs `[`vue-tsc`](https://github.com/vuejs/language-tools/tree/master/packages/tsc)` to check types throughout your app.

| Option | Default | Description |
|----------------------|-------------------|--|
| ----- ----- ----- | | |
| <code>rootDir</code> | <code> `.`</code> | The directory of the target application. |

```
::note
This command sets process.env.NODE_ENV to production. To override, define NODE_ENV in a [`.env`](/docs/guide/directory-structure/env) file or as a command-line argument.
::

::read-more{to="/docs/guide/concepts/typescript#type-checking"}
Read more on how to enable type-checking at build or development time.
::
```

```
---
title: "nuxi preview"
description: The preview command starts a server to preview your application after the build command.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/cli/blob/main/src/commands/preview.ts
    size: xs
---

```bash [Terminal]
npx nuxi preview|start [rootDir] [--dotenv]
```
```

The ``preview`` command starts a server to preview your Nuxt application after running the ``build`` command. The ``start`` command is an alias for ``preview``. When running your application in production refer to the [Deployment section](/docs/getting-started/deployment).

| Option | Default | Description |
|-------------------------|-------------------|---|
| ----- ----- ----- | | |
| <code>`rootDir`</code> | <code> `.`</code> | The root directory of the application to preview. |
| <code>`--dotenv`</code> | <code> `.`</code> | Point to another <code>`.env`</code> file to load, **relative** to the root directory. |

This command sets ``process.env.NODE_ENV`` to ``production``. To override, define ``NODE_ENV`` in a ``.env`` file or as command-line argument.

`::note`
For convenience, in preview mode, your `[`.env`]`(/docs/guide/directory-structure/env) file will be loaded into ``process.env``. (However, in production you will need to ensure your environment variables are set yourself.)
`::`

```
---
title: Runtime Config
description: Nuxt Kit provides a set of utilities to help you access and modify Nuxt runtime configuration.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/runtime-config.ts
    size: xs
---
```

`useRuntimeConfig`

At build-time, it is possible to access the resolved Nuxt [runtime config](/docs/guide/going-further/runtime-config).

Type

```
``ts
function useRuntimeConfig (): Record<string, unknown>
``
```

`updateRuntimeConfig`

It is also possible to update runtime configuration. This will be merged with the existing runtime configuration, and if Nitro has already been initialized it will trigger an HMR event to reload the Nitro runtime config.

```
``ts
function updateRuntimeConfig (config: Record<string, unknown>): void | Promise<void>
``
```

```

---
title: "Modules"
description: Nuxt Kit provides a set of utilities to help you create and use modules. You can use these utilities to create your own modules or to reuse existing modules.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/module
    size: xs
---

```

Modules are the building blocks of Nuxt. Kit provides a set of utilities to help you create and use modules. You can use these utilities to create your own modules or to reuse existing modules. For example, you can use the `defineNuxtModule` function to define a module and the `installModule` function to install a module programmatically.

`defineNuxtModule`

Define a Nuxt module, automatically merging defaults with user provided options, installing any hooks that are provided, and calling an optional setup function for full control.

Type

```

````ts
function defineNuxtModule<OptionsT extends ModuleOptions> (definition: ModuleDefinition<OptionsT> | NuxtModule<OptionsT>):
NuxtModule<OptionsT>

type ModuleOptions = Record<string, any>

interface ModuleDefinition<T extends ModuleOptions = ModuleOptions> {
 meta?: ModuleMeta
 defaults?: T | ((nuxt: Nuxt) => T)
 schema?: T
 hooks?: Partial<NuxtHooks>
 setup?: (this: void, resolvedOptions: T, nuxt: Nuxt) => Awaitable<void | false | ModuleSetupReturn>
}

interface NuxtModule<T extends ModuleOptions = ModuleOptions> {
 (this: void, inlineOptions: T, nuxt: Nuxt): Awaitable<void | false | ModuleSetupReturn>
 getOptions?: (inlineOptions?: T, nuxt?: Nuxt) => Promise<T>
 getMeta?: () => Promise<ModuleMeta>
}

interface ModuleSetupReturn {
 timings?: {
 setup?: number
 [key: string]: number | undefined
 }
}

interface ModuleMeta {
 name?: string
 version?: string
 configKey?: string
 compatibility?: NuxtCompatibility
 [key: string]: unknown
}
````

```

Parameters

`definition`

****Type**:** `ModuleDefinition<T> | NuxtModule<T>`

****Required**:** `true`

A module definition object or a module function.

- `meta` (optional)

****Type**:** `ModuleMeta`

Metadata of the module. It defines the module name, version, config key and compatibility.

- `defaults` (optional)

****Type**:** `T | ((nuxt: Nuxt) => T)`

Default options for the module. If a function is provided, it will be called with the Nuxt instance as the first argument.

- `schema` (optional)

****Type**:** `T`

Schema for the module options. If provided, options will be applied to the schema.

- `hooks` (optional)

****Type**:** `Partial<NuxtHooks>`

Hooks to be installed for the module. If provided, the module will install the hooks.

- `setup` (optional)

****Type**:** `(this: void, resolvedOptions: T, nuxt: Nuxt) => Awaitable<void | false | ModuleSetupReturn>`

Setup function for the module. If provided, the module will call the setup function.

Examples

```
```:ts
// https://github.com/nuxt/starter/tree/module
import { defineNuxtModule } from '@nuxt/kit'

export default defineNuxtModule({
 meta: {
 name: 'my-module',
 configKey: 'myModule'
 },
 defaults: {
 test: 123
 },
 setup (options, nuxt) {
 nuxt.hook('modules:done', () => {
 console.log('My module is ready with current test option: ', options.test)
 })
 }
})
```:
```

`installModule`

Install specified Nuxt module programmatically. This is helpful when your module depends on other modules. You can pass the module options as an object to `inlineOptions` and they will be passed to the module's `setup` function.

Type

```
```:ts
async function installModule (moduleToInstall: string | NuxtModule, inlineOptions?: any, nuxt?: Nuxt)
```:
```

Parameters

`moduleToInstall`

****Type**:** `string` | `NuxtModule`

****Required**:** `true`

The module to install. Can be either a string with the module name or a module object itself.

`inlineOptions`

****Type**:** `any`

****Default**:** `{}`

An object with the module options to be passed to the module's `setup` function.

`nuxt`

****Type**:** `Nuxt`

****Default**:** `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

Examples

```
```:ts
import { defineNuxtModule, installModule } from '@nuxt/kit'

export default defineNuxtModule({
 async setup (options, nuxt) {
 // will install @nuxtjs/fontaine with Roboto font and Impact fallback
 await installModule('@nuxtjs/fontaine', {
```

```
// module configuration
fonts: [
 {
 family: 'Roboto',
 fallbacks: ['Impact'],
 fallbackName: 'fallback-a',
 }
]
})
}
})
,,
```

```

title: "Templates"
description: Nuxt Kit provides a set of utilities to help you work with templates. These functions allow you to generate extra files during development and build time.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/template.ts
 size: xs

```

Templates allows to generate extra files during development and build time. These files will be available in virtual filesystem and can be used in plugins, layouts, components, etc. ``addTemplate`` and ``addTypeTemplate`` allow you to add templates to the Nuxt application. ``updateTemplates`` allows you to regenerate templates that match the filter.

## ``addTemplate``

Renders given template during build into the project buildDir.

### Type

```

```ts
function addTemplate (template: NuxtTemplate | string): ResolvedNuxtTemplate

interface NuxtTemplate {
  src?: string
  filename?: string
  dst?: string
  options?: Record<string, any>
  getContents?: (data: Record<string, any>) => string | Promise<string>
  write?: boolean
}

interface ResolvedNuxtTemplate {
  src: string
  filename: string
  dst: string
  options: Record<string, any>
  getContents: (data: Record<string, any>) => string | Promise<string>
  write: boolean
  filename: string
  dst: string
}
```

```

### Parameters

#### ``template``

**\*\*Type\*\*:** ``NuxtTemplate | string``

**\*\*Required\*\*:** ``true``

A template object or a string with the path to the template. If a string is provided, it will be converted to a template object with ``src`` set to the string value. If a template object is provided, it must have the following properties:

- ``src`` (optional)

**\*\*Type\*\*:** ``string``

Path to the template. If ``src`` is not provided, ``getContents`` must be provided instead.

- ``filename`` (optional)

**\*\*Type\*\*:** ``string``

Filename of the template. If ``filename`` is not provided, it will be generated from the ``src`` path. In this case, the ``src`` option is required.

- ``dst`` (optional)

**\*\*Type\*\*:** ``string``

Path to the destination file. If ``dst`` is not provided, it will be generated from the ``filename`` path and nuxt ``buildDir`` option.

- ``options`` (optional)

**\*\*Type\*\*:** ``Options``

Options to pass to the template.

- ``getContents`` (optional)



**\*\*Type\*\*:** `(data: Options) => string | Promise<string>`

A function that will be called with the `options` object. It should return a string or a promise that resolves to a string. If `src` is provided, this function will be ignored.

- `write` (optional)

**\*\*Type\*\*:** `boolean`

If set to `true`, the template will be written to the destination file. Otherwise, the template will be used only in virtual filesystem.

### Examples

::code-group

```
``ts [module.ts]
// https://github.com/nuxt/bridge
import { addTemplate, defineNuxtModule } from '@nuxt/kit'
import { defu } from 'defu'

export default defineNuxtModule({
 setup(options, nuxt) {
 const globalMeta = defu(nuxt.options.app.head, {
 charset: options.charset,
 viewport: options.viewport
 })

 addTemplate({
 filename: 'meta.config.mjs',
 getContents: () => `export default ` + JSON.stringify({ globalMeta, mixinKey: 'setup' })
 })
 }
})
``
```

```
``ts [plugin.ts]
import { createHead as createClientHead, createServerHead } from '@unhead/vue'
import { defineNuxtPlugin } from '#imports'
// @ts-ignore
import metaConfig from '#build/meta.config.mjs'

export default defineNuxtPlugin((nuxtApp) => {
 const createHead = import.meta.server ? createServerHead : createClientHead
 const head = createHead()
 head.push(metaConfig.globalMeta)

 nuxtApp.vueApp.use(head)
})
``
```

::

## `addTypeTemplate`

Renders given template during build into the project buildDir, then registers it as types.

### Type

```
``ts
function addTypeTemplate (template: NuxtTypeTemplate | string): ResolvedNuxtTemplate

interface NuxtTemplate {
 src?: string
 filename?: string
 dst?: string
 options?: Record<string, any>
 getContents?: (data: Record<string, any>) => string | Promise<string>
}

interface ResolvedNuxtTemplate {
 src: string
 filename: string
 dst: string
 options: Record<string, any>
 getContents: (data: Record<string, any>) => string | Promise<string>
 write: boolean
 filename: string
 dst: string
}
``
```

### ### Parameters

#### #### `template`

**\*\*Type\*\*:** `NuxtTypeTemplate | string`

**\*\*Required\*\*:** `true`

A template object or a string with the path to the template. If a string is provided, it will be converted to a template object with `src` set to the string value. If a template object is provided, it must have the following properties:

- `src` (optional)

**\*\*Type\*\*:** `string`

Path to the template. If `src` is not provided, `getContents` must be provided instead.

- `filename` (optional)

**\*\*Type\*\*:** `string`

Filename of the template. If `filename` is not provided, it will be generated from the `src` path. In this case, the `src` option is required.

- `dst` (optional)

**\*\*Type\*\*:** `string`

Path to the destination file. If `dst` is not provided, it will be generated from the `filename` path and nuxt `buildDir` option.

- `options` (optional)

**\*\*Type\*\*:** `Options`

Options to pass to the template.

- `getContents` (optional)

**\*\*Type\*\*:** `(data: Options) => string | Promise<string>`

A function that will be called with the `options` object. It should return a string or a promise that resolves to a string. If `src` is provided, this function will be ignored.

### ### Examples

```
``ts
// https://github.com/Hebilicious/nuxtpress
import { addTypeTemplate, defineNuxtModule } from "@nuxt/kit"
```

```
export default defineNuxtModule({
 setup() {
 addTypeTemplate({
 filename: "types/markdown.d.ts",
 getContents: () => /* ts */`
 declare module '*.md' {
 import type { ComponentOptions } from 'vue'
 const Component: ComponentOptions
 export default Component
 }`
 })
 }
})
}
}
...`
```

### ## `updateTemplates`

Regenerate templates that match the filter. If no filter is provided, all templates will be regenerated.

#### ### Type

```
``ts
async function updateTemplates (options: UpdateTemplatesOptions): void
```

```
interface UpdateTemplatesOptions {
 filter?: (template: ResolvedNuxtTemplate) => boolean
}
```

```
interface ResolvedNuxtTemplate {
 src: string
 filename: string
 dst: string
 options: Record<string, any>
```

```
 getContents: (data: Record<string, any>) => string | Promise<string>
 write: boolean
 filename: string
 dst: string
}
``,`
```

### ### Parameters

#### #### `options`

**\*\*Type\*\*:** `UpdateTemplatesOptions`

**\*\*Default\*\*:** `{}`

Options to pass to the template. This object can have the following property:

- `filter` (optional)

**\*\*Type\*\*:** `(template: ResolvedNuxtTemplate) => boolean`

A function that will be called with the `template` object. It should return a boolean indicating whether the template should be regenerated. If `filter` is not provided, all templates will be regenerated.

### ### Example

```
````ts
// https://github.com/nuxt/nuxt
import { defineNuxtModule, updateTemplates } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    // watch and rebuild routes template list when one of the pages changes
    nuxt.hook('builder:watch', async (event, relativePath) => {
      if (event === 'change') { return }

      const path = resolve(nuxt.options.srcDir, relativePath)
      if (updateTemplatePaths.some(dir => path.startsWith(dir))) {
        await updateTemplates({
          filter: template => template.filename === 'routes.mjs'
        })
      }
    })
  })
})
``,`
```

```
---
title: 'Import meta'
description: Understand where your code is running using `import.meta`.
---
```

The `import.meta` object

With ES modules you can obtain some metadata from the code that imports or compiles your ES-module. This is done through `import.meta`, which is an object that provides your code with this information. Throughout the Nuxt documentation you may see snippets that use this already to figure out whether the code is currently running on the client or server side.

```
::read-more{to="https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import.meta"}
Read more about `import.meta`.
::
```

Runtime (App) Properties

These values are statically injected and can be used for tree-shaking your runtime code.

| Property | Type | Description |
|-------------------------|---------|--|
| --- | --- | --- |
| `import.meta.client` | boolean | True when evaluated on the client side. |
| `import.meta.browser` | boolean | True when evaluated on the client side. |
| `import.meta.server` | boolean | True when evaluated on the server side. |
| `import.meta.nitro` | boolean | True when evaluated on the server side. |
| `import.meta.dev` | boolean | True when running the Nuxt dev server. |
| `import.meta.test` | boolean | True when running in a test context. |
| `import.meta.prerender` | boolean | True when rendering HTML on the server in the prerender stage of your build. |

Builder Properties

These values are available both in modules and in your `nuxt.config`.

| Property | Type | Description |
|-------------------|--------|---------------------------------------|
| --- | --- | --- |
| `import.meta.env` | object | Equals `process.env` |
| `import.meta.url` | string | Resolvable path for the current file. |

Examples

Using `import.meta.url` to resolve files within modules

```
``ts [modules/my-module/index.ts]
import { createResolver } from 'nuxt/kit'

// Resolve relative from the current file
const resolver = createResolver(import.meta.url)

export default defineNuxtModule({
  meta: { name: 'myModule' },
  setup() {
    addComponent({
      name: 'MyModuleComponent',
      // Resolves to '/modules/my-module/components/MyModuleComponent.vue'
      filePath: resolver.resolve('./components/MyModuleComponent.vue')
    })
  }
})
``
```

title: 'Lifecycle Hooks'
description: Nuxt provides a powerful hooking system to expand almost every aspect using hooks.

:read-more{to="/docs/guide/going-further/hooks"}

App Hooks (runtime)

Check the [app source code](https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts#L37) for all available hooks.

| Hook | Arguments | Environment | Description |
|------------------------------|---------------------|-----------------|---|
| ----- | ----- | ----- | ----- |
| `app:created` | `vueApp` | Server & Client | Called when initial `vueApp` instance is created. |
| `app:error` | `err` | Server & Client | Called when a fatal error occurs. |
| `app:error:cleared` | `{ redirect? }` | Server & Client | Called when a fatal error occurs. |
| `app:data:refresh` | `keys?` | Server & Client | (internal) |
| `vue:setup` | - | Server & Client | (internal) |
| `vue:error` | `err, target, info` | Server & Client | Called when a vue error propagates to the root component.
[Learn More](https://vuejs.org/api/composition-api-lifecycle.html#onerrorcaptured). |
| `app:rendered` | `renderContext` | Server | Called when SSR rendering is done. |
| `app:redirection` | - | Server | Called before SSR redirection. |
| `app:beforeMount` | `vueApp` | Client | Called before mounting the app, called only on client side. |
| `app:mounted` | `vueApp` | Client | Called when Vue app is initialized and mounted in browser. |
| `app:suspense:resolve` | `appComponent` | Client | On [Suspense](https://vuejs.org/guide/built-ins/suspense.html#suspense) resolved event. |
| `app:manifest:update` | `{ id, timestamp }` | Client | Called when there is a newer version of your app detected. |
| `link:prefetch` | `to` | Client | Called when a ` <nuxtlink>` is observed to be prefetched.</nuxtlink> |
| `page:start` | `pageComponent?` | Client | Called on [Suspense](https://vuejs.org/guide/built-ins/suspense.html#suspense) pending event. |
| `page:finish` | `pageComponent?` | Client | Called on [Suspense](https://vuejs.org/guide/built-ins/suspense.html#suspense) resolved event. |
| `page:loading:start` | - | Client | Called when the `setup()` of the new page is running. |
| `page:loading:end` | - | Client | Called after `page:finish` |
| `page:transition:finish` | `pageComponent?` | Client | After page transition [onAfterLeave](https://vuejs.org/guide/built-ins/transition.html#javascript-hooks) event. |
| `dev:ssr-logs` | `logs` | Client | Called with an array of server-side logs that have been passed to the client (if `features.devLogs` is enabled). |
| `page:view-transition:start` | `transition` | Client | Called after `document.startViewTransition` is called when [experimental viewTransition support is enabled](/docs/getting-started/transitions#view-transitions-api-experimental). |

Nuxt Hooks (build time)

Check the [schema source code](https://github.com/nuxt/nuxt/blob/main/packages/schema/src/types/hooks.ts#L53) for all available hooks.

| Hook | Arguments | Description |
|--------------------------|-------------------------|---|
| ----- | ----- | ----- |
| `kit:compatibility` | `compatibility, issues` | Allows extending compatibility checks. |
| `ready` | `nuxt` | Called after Nuxt initialization, when the Nuxt instance is ready to work. |
| `close` | `nuxt` | Called when Nuxt instance is gracefully closing. |
| `restart` | `{ hard?: boolean }` | To be called to restart the current Nuxt instance. |
| `modules:before` | - | Called during Nuxt initialization, before installing user modules. |
| `modules:done` | - | Called during Nuxt initialization, after installing user modules. |
| `app:resolve` | `app` | Called after resolving the `app` instance. |
| `app:templates` | `app` | Called during `NuxtApp` generation, to allow customizing, modifying or adding new files to the build directory (either virtually or to written to `.nuxt`). |
| `app:templatesGenerated` | `app` | Called after templates are compiled into the [virtual file system](/docs/guide/directory-structure/nuxt#virtual-file-system) (vfs). |
| `build:before` | - | Called before Nuxt bundle builder. |
| `build:done` | - | Called after Nuxt bundle builder is complete. |
| `build:manifest` | `manifest` | Called during the manifest build by Vite and webpack. This allows customizing the manifest that Nitro will use to render ` <script>` and `<link>` tags in the final HTML.</td></tr><tr><td>`builder:generateApp`</td><td>`options`</td><td>Called before generating the app.</td></tr><tr><td>`builder:watch`</td><td>`event, path`</td><td>Called at build time in development when the watcher spots a change to a file or directory in the project.</td></tr><tr><td>`pages:extend`</td><td>`pages`</td><td>Called after pages routes are resolved.</td></tr><tr><td>`pages:routerOptions`</td><td>`{ files: Array<{ path: string, optional?: boolean }> }`</td><td>Called when resolving `router.options` files. Later items in the array override earlier ones.</td></tr><tr><td>`server:devHandler`</td><td>`handler`</td><td>Called when the dev middleware is being registered on the Nitro dev server.</td></tr><tr><td>`imports:sources`</td><td>`presets`</td><td>Called at setup allowing modules to extend sources.</td></tr><tr><td>`imports:extend`</td><td>`imports`</td><td>Called at setup allowing modules to extend imports.</td></tr><tr><td>`imports:context`</td><td>`context`</td><td>Called when the [unimport](https://github.com/unjs/unimport) context is created.</td></tr><tr><td>`imports:dirs`</td><td>`dirs`</td><td>Allows extending import directories.</td></tr><tr><td>`components:dirs`</td><td>`dirs`</td><td>Called within `app:resolve` allowing to extend the directories that are scanned for auto-importable components.</td></tr><tr><td>`components:extend`</td><td>`components`</td><td>Allows extending new components.</td></tr><tr><td>`nitro:config`</td><td>`nitroConfig`</td><td>Called before initializing Nitro, allowing customization of Nitro's configuration.</td></tr></table></div></script> |

| | | | |
|--|---|----------------|---|
| <code>`nitro:init`</code> | <code> `nitro`</code> | <code> </code> | Called after Nitro is initialized, which allows registering Nitro hooks and interacting directly with Nitro. |
| <code>`nitro:build:before`</code> | <code> `nitro`</code> | <code> </code> | Called before building the Nitro instance. |
| <code>`nitro:build:public-assets`</code> | <code> `nitro`</code> | <code> </code> | Called after copying public assets. Allows modifying public assets before Nitro server is built. |
| <code>`prerender:routes`</code> | <code> `ctx`</code> | <code> </code> | Allows extending the routes to be pre-rendered. |
| <code>`build:error`</code> | <code> `error`</code> | <code> </code> | Called when an error occurs at build time. |
| <code>`prepare:types`</code> | <code> `options`</code> | <code> </code> | Called before Nuxt writes <code>`.nuxt/tsconfig.json`</code> and <code>`.nuxt/nuxt.d.ts`</code> , allowing addition of custom references and declarations in <code>`.nuxt.d.ts`</code> , or directly modifying the options in <code>`.tsconfig.json`</code> |
| <code>`listen`</code> | <code> `listenerServer, listener`</code> | <code> </code> | Called when the dev server is loading. |
| <code>`schema:extend`</code> | <code> `schemas`</code> | <code> </code> | Allows extending default schemas. |
| <code>`schema:resolved`</code> | <code> `schema`</code> | <code> </code> | Allows extending resolved schema. |
| <code>`schema:beforeWrite`</code> | <code> `schema`</code> | <code> </code> | Called before writing the given schema. |
| <code>`schema:written`</code> | <code> -</code> | <code> </code> | Called after the schema is written. |
| <code>`vite:extend`</code> | <code> `viteBuildContext`</code> | <code> </code> | Allows to extend Vite default context. |
| <code>`vite:extendConfig`</code> | <code> `viteInlineConfig, env`</code> | <code> </code> | Allows to extend Vite default config. |
| <code>`vite:configResolved`</code> | <code> `viteInlineConfig, env`</code> | <code> </code> | Allows to read the resolved Vite config. |
| <code>`vite:serverCreated`</code> | <code> `viteServer, env`</code> | <code> </code> | Called when the Vite server is created. |
| <code>`vite:compiled`</code> | <code> -</code> | <code> </code> | Called after Vite server is compiled. |
| <code>`webpack:config`</code> | <code> `webpackConfigs`</code> | <code> </code> | Called before configuring the webpack compiler. |
| <code>`webpack:configResolved`</code> | <code> `webpackConfigs`</code> | <code> </code> | Allows to read the resolved webpack config. |
| <code>`webpack:compile`</code> | <code> `options`</code> | <code> </code> | Called right before compilation. |
| <code>`webpack:compiled`</code> | <code> `options`</code> | <code> </code> | Called after resources are loaded. |
| <code>`webpack:change`</code> | <code> `shortPath`</code> | <code> </code> | Called on <code>`.change`</code> on WebpackBar. |
| <code>`webpack:error`</code> | <code> -</code> | <code> </code> | Called on <code>`.done`</code> if has errors on WebpackBar. |
| <code>`webpack:done`</code> | <code> -</code> | <code> </code> | Called on <code>`.allDone`</code> on WebpackBar. |
| <code>`webpack:progress`</code> | <code> `statesArray`</code> | <code> </code> | Called on <code>`.progress`</code> on WebpackBar. |

Nitro App Hooks (runtime, server-side)

See [Nitro](https://nitro.unjs.io/guide/plugins#available-hooks) for all available hooks.

| Hook | Arguments | Description | Types |
|---|---|--|--|
| <code>`dev:ssr-logs`</code> | <code> `{ path, logs }`</code> | <code> </code> Server | <code> </code> Called at the end of a request cycle with an array of server-side logs. |
| <code>`render:response`</code> | <code> `response, { event }`</code> | <code> </code> Called before sending the response. | <code> </code> [response] |
| (https://github.com/nuxt/nuxt/blob/71ef8bd3ff207fd51c2ca18d5a8c7140476780c7/packages/nuxt/src/core/runtime/nitro/renderer.ts#L24), [event](https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38) | | | |
| <code>`render:html`</code> | <code> `html, { event }`</code> | <code> </code> Called before constructing the HTML. | <code> </code> [html] |
| (https://github.com/nuxt/nuxt/blob/71ef8bd3ff207fd51c2ca18d5a8c7140476780c7/packages/nuxt/src/core/runtime/nitro/renderer.ts#L15), [event](https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38) | | | |
| <code>`render:island`</code> | <code> `islandResponse, { event, islandContext }`</code> | <code> </code> Called before constructing the island HTML. | <code> </code> [islandResponse] |
| (https://github.com/nuxt/nuxt/blob/e50cabfed1984c341af0d0c056a325a8aec26980/packages/nuxt/src/core/runtime/nitro/renderer.ts#L28), [event](https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38), [islandContext](https://github.com/nuxt/nuxt/blob/e50cabfed1984c341af0d0c056a325a8aec26980/packages/nuxt/src/core/runtime/nitro/renderer.ts#L38) | | | |
| <code>`close`</code> | <code> -</code> | <code> </code> Called when Nitro is closed. | <code> </code> - |
| <code>`error`</code> | <code> `error, { event? }`</code> | <code> </code> Called when an error occurs. | <code> </code> [error] |
| (https://github.com/unjs/nitro/blob/d20ffcdbd16fc4003b774445e1a01e698c2bb078a/src/types/runtime/nitro.ts#L48), [event](https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38) | | | |
| <code>`request`</code> | <code> `event`</code> | <code> </code> Called when a request is received. | <code> </code> [event] |
| (https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38) | | | |
| <code>`beforeResponse`</code> | <code> `event, { body }`</code> | <code> </code> Called before sending the response. | <code> </code> [event] |
| (https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38), unknown | | | |
| <code>`afterResponse`</code> | <code> `event, { body }`</code> | <code> </code> Called after sending the response. | <code> </code> [event] |
| (https://github.com/unjs/h3/blob/f6ceb5581043dc4d8b6eab91e9be4531e0c30f8e/src/types.ts#L38), unknown | | | |

```
---
title: Plugins and Middleware
description: 'Learn how to migrate from Nuxt 2 to Nuxt Bridge new plugins and middleware.'
---
```

New Plugins Format

You can now migrate to the Nuxt 3 plugins API, which is slightly different in format from Nuxt 2.

Plugins now take only one argument (`nuxtApp`). You can find out more in [the docs](/docs/guide/directory-structure/plugins).

```
```js [plugins/hello.ts]
export default defineNuxtPlugin(nuxtApp => {
 nuxtApp.provide('injected', () => 'my injected function')
 // now available on `nuxtApp.$injected`
})
```
```

::note

If you want to use the new Nuxt composables (such as [`useNuxtApp`](/docs/api/composables/use-nuxt-app) or `useRuntimeConfig`) within your plugins, you will need to use the `defineNuxtPlugin` helper for those plugins.

Although a compatibility interface is provided via `nuxtApp.vueApp` you should avoid registering plugins, directives, mixins or components this way without adding your own logic to ensure they are not installed more than once, or this may cause a memory leak.

New Middleware Format

You can now migrate to the Nuxt 3 middleware API, which is slightly different in format from Nuxt 2.

Middleware now take only two argument (`to`, `from`). You can find out more in [the docs](/docs/guide/directory-structure/middleware).

```
```ts twoslash
export default defineNuxtRouteMiddleware((to) => {
 if (to.path !== '/') {
 return navigateTo('/')
 }
})
```
```

::important

Use of `defineNuxtRouteMiddleware` is not supported outside of the middleware directory.

definePageMeta

You can also use [`definePageMeta`](/docs/api/utils/define-page-meta) in Nuxt Bridge.

You can be enabled with the `macros.pageMeta` option in your configuration file

```
```ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
 bridge: {
 macros: {
 pageMeta: true
 }
 }
})
```
```

::note

But only for `middleware` and `layout`.

```
---
title: Legacy Composition API
description: 'Learn how to migrate to Composition API with Nuxt Bridge.'
---
```

Nuxt Bridge provides access to Composition API syntax. It is specifically designed to be aligned with Nuxt 3. Because of this, there are a few extra steps to take when enabling Nuxt Bridge, if you have been using the Composition API previously.

Remove Modules

- Remove `@vue/composition-api` from your dependencies.
- Remove `@nuxtjs/composition-api` from your dependencies (and from your modules in `nuxt.config`).

Using `@vue/composition-api`

If you have been using just `@vue/composition-api` and not `@nuxtjs/composition-api`, then things are very straightforward.

1. First, remove the plugin where you are manually registering the Composition API. Nuxt Bridge will handle this for you.

```
```diff
- import Vue from 'vue'
- import VueCompositionApi from '@vue/composition-api'
-
- Vue.use(VueCompositionApi)
```
```

2. Otherwise, there is nothing you need to do. However, if you want, you can remove your explicit imports from `@vue/composition-api` and rely on Nuxt Bridge auto-importing them for you.

Migrating from `@nuxtjs/composition-api`

Nuxt Bridge implements the Composition API slightly differently from `@nuxtjs/composition-api` and provides different composables (designed to be aligned with the composables that Nuxt 3 provides).

Because some composables have been removed and don't yet have a replacement, this will be a slightly more complicated process.

Remove `@nuxtjs/composition-api/module` from your buildModules

You don't have to immediately update your imports yet - Nuxt Bridge will automatically provide a 'shim' for most imports you currently have, to give you time to migrate to the new, Nuxt 3-compatible composables, with the following exceptions:

- `withContext` has been removed. See [below](/docs/bridge/nuxt3-compatible-api#usecontext-and-withcontext).
- `useStatic` has been removed. There is no current replacement. Feel free to raise a discussion if you have a use case for this.
- `reqRef` and `reqSsrRef`, which were deprecated, have now been removed entirely. Follow the instructions below regarding [ssrRef](/docs/bridge/nuxt3-compatible-api#ssrref-and-shallowssrref) to replace this.

Set `bridge.capi`

```
```ts
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
 bridge: {
 capi: true,
 nitro: false // If migration to Nitro is complete, set to true
 }
})
```
```

For each other composable you are using from `@nuxtjs/composition-api`, follow the steps below.

useFetch

`\$fetchState` and `\$fetch` have been removed.

```
```diff
const {
- $fetch,
- $fetchState,
+ fetch,
+ fetchState,
} = useFetch(() => { posts.value = await $fetch('/api/posts') })
```
```

`defineNuxtMiddleware`

This was a type-helper stub function that is now removed.

Remove the `defineNuxtMiddleware` wrapper:

```
```diff
```



```
- import { defineNuxtMiddleware } from '@nuxtjs/composition-api'
- export default defineNuxtMiddleware((ctx) => {})
+ export default (ctx) => {}
```

```

For typescript support, you can use `@nuxt/types`:

```
```ts
import type { Middleware } from '@nuxt/types'

export default <Middleware> function (ctx) { }
```
```

`defineNuxtPlugin`

This was a type-helper stub function that is now removed.

You may also keep using Nuxt 2-style plugins, by removing the function (as with `[defineNuxtMiddleware]` (`#definenuxtmiddleware`)).

Remove the `defineNuxtPlugin` wrapper:

```
```diff
- import { defineNuxtPlugin } from '@nuxtjs/composition-api'
- export default defineNuxtPlugin((ctx, inject) => {})
+ export default (ctx, inject) => {}
```
```

For typescript support, you can use `@nuxt/types`:

```
```ts
import type { Plugin } from '@nuxt/types'

export default <Plugin> function (ctx, inject) {}
```
```

:::alert{type="warning"}

While this example is valid, Nuxt 3 introduces a new `defineNuxtPlugin` function that has a slightly different signature.
::

:ReadMore{link="/docs/guide/directory-structure/plugins#creating-plugins"}

`useRouter` and `useRoute`

Nuxt Bridge provides direct replacements for these composables via `[`useRouter`](/docs/api/composables/use-router)` and ``useRoute``.

The only key difference is that `[`useRoute`](/docs/api/composables/use-route)` no longer returns a computed property.

```
```diff
- import { useRouter, useRoute } from '@nuxtjs/composition-api'

 const router = useRouter()
 const route = useRoute()

- console.log(route.value.path)
+ console.log(route.path)
```
```

```
---
title: New Composition API
description: Nuxt Bridge implements composables compatible with Nuxt 3.
---
```

By migrating from `@nuxtjs/composition-api` to the Nuxt 3 compatible API, there will be less rewriting when migrating to Nuxt 3.

`ssrRef` and `shallowSsrRef`

These two functions have been replaced with a new composable that works very similarly under the hood: `useState`.

The key differences are that you must provide a `_key_` for this state (which Nuxt generated automatically for `ssrRef` and `shallowSsrRef`), and that it can only be called within a Nuxt 3 plugin (which is defined by `defineNuxtPlugin`) or a component instance. (In other words, you cannot use `useState` (</docs/api/composables/use-state>) with a global/ambient context, because of the danger of shared state across requests.)

```
```diff
- import { ssrRef } from '@nuxtjs/composition-api'

- const ref1 = ssrRef('initialData')
- const ref2 = ssrRef(() => 'factory function')
+ const ref1 = useState('ref1-key', () => 'initialData')
+ const ref2 = useState('ref2-key', () => 'factory function')
 // accessing the state
 console.log(ref1.value)
```
```

Because the state is keyed, you can access the same state from multiple locations, as long as you are using the same key.

You can read more about how to use this composable in [\[the Nuxt 3 docs\]](/docs/api/composables/use-state).

`ssrPromise`

This function has been removed, and you will need to find an alternative implementation if you were using it. If you have a use case for `ssrPromise`, please let us know via a discussion.

`onGlobalSetup`

This function has been removed, but its use cases can be met by using `useNuxtApp` (</docs/api/composables/use-nuxt-app>) or `useState` (</docs/api/composables/use-state>) within `defineNuxtPlugin`. You can also run any custom code within the `setup()` function of a layout.

```
```diff
- import { onGlobalSetup } from '@nuxtjs/composition-api'

- export default () => {
- onGlobalSetup(() => {
+ export default defineNuxtPlugin((nuxtApp) => {
+ nuxtApp.hook('vue:setup', () => {
 // ...
 })
- }
+ })
```
```

`useStore`

In order to access Vuex store instance, you can use `useNuxtApp().$store`.

```
```diff
- import { useStore } from '@nuxtjs/composition-api`
+ const { $store } = useNuxtApp()
```
```

`useContext` and `withContext`

You can access injected helpers using `useNuxtApp`.

```
```diff
- import { useContext } from '@nuxtjs/composition-api`
+ const { $axios } = useNuxtApp()
```
```

::note
`useNuxtApp()` also provides a key called `nuxt2Context` which contains all the same properties you would normally access from Nuxt 2 context, but it's advised not to use this directly, as it won't exist in Nuxt 3. Instead, see if there is another way to access what you need. (If not, please raise a feature request or discussion.)
::

`wrapProperty`

This helper function is not provided any more but you can replace it with the following code:

```
```js
const wrapProperty = (property, makeComputed = true) => () => {
 const vm = getCurrentInstance().proxy
 return makeComputed ? computed(() => vm[property]) : vm[property]
}
```
```

`useAsync` and `useFetch`

These two composables can be replaced with `useLazyAsyncData` and `useLazyFetch`, which are documented [in the Nuxt 3 docs] (/docs/getting-started/data-fetching). Just like the previous `@nuxtjs/composition-api` composables, these composables do not block route navigation on the client-side (hence the 'lazy' part of the name).

::important

Note that the API is entirely different, despite similar sounding names. Importantly, you should not attempt to change the value of other variables outside the composable (as you may have been doing with the previous `useFetch`).

::

::warning

The `useLazyFetch` must have been configured for [Nitro] (/docs/bridge/nitro).

::

Migrating to the new composables from `useAsync`:

```
```diff
<script setup>
- import { useAsync } from '@nuxtjs/composition-api'
- const posts = useAsync(() => $fetch('/api/posts'))
+ const { data: posts } = useLazyAsyncData('posts', () => $fetch('/api/posts'))
+ // or, more simply!
+ const { data: posts } = useLazyFetch('/api/posts')
</script>
```
```

Migrating to the new composables from `useFetch`:

```
```diff
<script setup>
- import { useFetch } from '@nuxtjs/composition-api'
- const posts = ref([])
- const { fetch } = useFetch(() => { posts.value = await $fetch('/api/posts') })
+ const { data: posts, refresh } = useLazyAsyncData('posts', () => $fetch('/api/posts'))
+ // or, more simply!
+ const { data: posts, refresh } = useLazyFetch('/api/posts')
+ function updatePosts() {
- return fetch()
+ return refresh()
 }
</script>
```
```

`useMeta`

In order to interact with `vue-meta`, you may use `useNuxt2Meta`, which will work in Nuxt Bridge (but not Nuxt 3) and will allow you to manipulate your meta tags in a `vue-meta`-compatible way.

```
```diff
<script setup>
- import { useMeta } from '@nuxtjs/composition-api'
+ useNuxt2Meta({
+ title: 'My Nuxt App',
+ })
</script>
```
```

You can also pass in computed values or refs, and the meta values will be updated reactively:

```
```ts
<script setup>
const title = ref('my title')
useNuxt2Meta({
 title,
})
title.value = 'new title'
</script>
```
```

::note

Be careful not to use both `useNuxt2Meta()` and the Options API `head()` within the same component, as behavior may be unpredictable.

::

Nuxt Bridge also provides a Nuxt 3-compatible meta implementation that can be accessed with the [`useHead`] (/docs/api/composables/use-head) composable.

```
``diff
<script setup>
- import { useMeta } from '@nuxtjs/composition-api'
  useHead({
    title: 'My Nuxt App',
  })
</script>
```
```

You will also need to enable it explicitly in your `nuxt.config`:

```
```js
import { defineNuxtConfig } from '@nuxt/bridge'
export default defineNuxtConfig({
  bridge: {
    meta: true
  }
})
```
```

This [`useHead`] (/docs/api/composables/use-head) composable uses `@unhead/vue` under the hood (rather than `vue-meta`) to manipulate your `<head>`. Accordingly, it is recommended not to use both the native Nuxt 2 `head()` properties as well as [`useHead`] (/docs/api/composables/use-head), as they may conflict.

For more information on how to use this composable, see [the Nuxt 3 docs] (/docs/getting-started/seo-meta).

### ### Explicit Imports

Nuxt exposes every auto-import with the `#imports` alias that can be used to make the import explicit if needed:

```
```vue
<script setup lang="ts">
import { ref, computed } from '#imports'

const count = ref(1)
const double = computed(() => count.value * 2)
</script>
```
```

### ### Disabling Auto-imports

If you want to disable auto-importing composables and utilities, you can set `imports.autoImport` to `false` in the `nuxt.config` file.

```
```ts [nuxt.config.ts]
export default defineNuxtConfig({
  imports: {
    autoImport: false
  }
})
```
```

This will disable auto-imports completely but it's still possible to use [explicit imports] (#explicit-imports) from `#imports`.

```

title: Auto Imports
description: Nuxt 3 adopts a minimal friction approach, meaning wherever possible components and composables are auto-imported.

::note
In the rest of the migration documentation, you will notice that key Nuxt and Vue utilities do not have explicit imports. This is not a typo; Nuxt will automatically import them for you, and you should get full type hinting if you have followed [the instructions](/docs/migration/configuration#typescript) to use Nuxt's TypeScript support.
::

[Read more about auto imports](/docs/guide/concepts/auto-imports)

Migration

1. If you have been using `@nuxt/components` in Nuxt 2, you can remove `components: true` in your `nuxt.config`. If you had a more complex setup, then note that the component options have changed somewhat. See the [components documentation](/docs/guide/directory-structure/components) for more information.

::tip
You can look at `.nuxt/types/components.d.ts` and `.nuxt/types/imports.d.ts` to see how Nuxt has resolved your components and composable auto-imports.
::
```

```

title: Modules
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 modules.'

```

## ## Module Compatibility

Nuxt 3 has a basic backward compatibility layer for Nuxt 2 modules using `@nuxt/kit` auto wrappers. But there are usually steps to follow to make modules compatible with Nuxt 3 and sometimes, using Nuxt Bridge is required for cross-version compatibility.

We have prepared a [Dedicated Guide](/docs/guide/going-further/modules) for authoring Nuxt 3 ready modules using `@nuxt/kit`. Currently best migration path is to follow it and rewrite your modules. Rest of this guide includes preparation steps if you prefer to avoid a full rewrite yet making modules compatible with Nuxt 3.

```
::tip{icon="i-ph-puzzle-piece-duotone" to="/modules"}
Explore Nuxt 3 compatible modules.
::
```

## ### Plugin Compatibility

Nuxt 3 plugins are **not** fully backward compatible with Nuxt 2.

```
:read-more{to="/docs/guide/directory-structure/plugins"}
```

## ### Vue Compatibility

Plugins or components using the Composition API need exclusive Vue 2 or Vue 3 support.

By using [vue-demi](https://github.com/vueuse/vue-demi) they should be compatible with both Nuxt 2 and 3.

## ## Module Migration

When Nuxt 3 users add your module, you will not have access to the module container (`this.*`) so you will need to use utilities from `@nuxt/kit` to access the container functionality.

## ### Test with `@nuxt/bridge`

Migrating to `@nuxt/bridge` is the first and most important step for supporting Nuxt 3.

If you have a fixture or example in your module, add `@nuxt/bridge` package to its config (see [example](/docs/bridge/overview#update-nuxtconfig))

## ### Migrate from CommonJS to ESM

Nuxt 3 natively supports TypeScript and ECMAScript Modules. Please check [Native ES Modules](/docs/guide/concepts/esm) for more info and upgrading.

## ### Ensure Plugins Default Export

If you inject a Nuxt plugin that does not have `export default` (such as global Vue plugins), ensure you add `export default () => { }` to the end of it.

```
::code-group
```

```
```js [Before]
// ~/plugins/vuelidate.js
import Vue from 'vue'
import Vuelidate from 'vuelidate'
```

```
Vue.use(Vuelidate)
```
```

```
```js [After]
// ~/plugins/vuelidate.js
import Vue from 'vue'
import Vuelidate from 'vuelidate'
```

```
Vue.use(Vuelidate)
```

```
export default () => { }
```
```

```
::
```

## ### Avoid Runtime Modules

With Nuxt 3, Nuxt is now a build-time-only dependency, which means that modules shouldn't attempt to hook into the Nuxt runtime.

Your module should work even if it's only added to [`buildModules`](/docs/api/nuxt-config#runtimeconfig) (instead of `modules`). For example:

- Avoid updating `process.env` within a Nuxt module and reading by a Nuxt plugin; use `[`runtimeConfig`](/docs/api/nuxt-config#runtimeconfig)` instead.
- (\*) Avoid depending on runtime hooks like `vue-renderer:*` for production
- (\*) Avoid adding `serverMiddleware` by importing them inside the module. Instead, add them by referencing a file path so that they are independent of the module's context

(\*) Unless it is for `nuxt dev` purpose only and guarded with `if (nuxt.options.dev) { }`.

:::tip

Continue reading about Nuxt 3 modules in the [\[Modules Author Guide\]\(/docs/guide/going-further/modules\)](/docs/guide/going-further/modules).

:::

### ### Use TypeScript (Optional)

While it is not essential, most of the Nuxt ecosystem is shifting to use TypeScript, so it is highly recommended to consider migration.

:::tip

You can start migration by renaming `.js` files, to `.ts`. TypeScript is designed to be progressive!

:::

:::tip

You can use TypeScript syntax for Nuxt 2 and 3 modules and plugins without any extra dependencies.

:::

```

title: Configuration
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 new configuration.'

```

```
`nuxt.config`
```

The starting point for your Nuxt app remains your `nuxt.config` file.

```
::note
Nuxt configuration will be loaded using [`unjs/jiti`](https://github.com/unjs/jiti) and [`unjs/c12`]
(https://github.com/unjs/c12).
::
```

```
Migration
```

1. You should migrate to the new `defineNuxtConfig` function that provides a typed configuration schema.

```
::code-group
```

```
```ts [Nuxt 2]
export default {
  // ...
}
```

```ts [Nuxt 3]
export default defineNuxtConfig({
  // ...
})
```
```

```
::
```

1. If you were using `router.extendRoutes` you can migrate to the new `pages:extend` hook:

```
::code-group
```

```
```ts [Nuxt 2]
export default {
  router: {
    extendRoutes (routes) {
      //
    }
  }
}
```

```ts [Nuxt 3]
export default defineNuxtConfig({
  hooks: {
    'pages:extend' (routes) {
      //
    }
  }
})
```
```

```
::
```

1. If you were using `router.routeNameSplitter` you can achieve same result by updating route name generation logic in the new `pages:extend` hook:

```
::code-group
```

```
```ts [Nuxt 2]
export default {
  router: {
    routeNameSplitter: '/'
  }
}
```

```ts [Nuxt 3]
import { createResolver } from '@nuxt/kit'

export default defineNuxtConfig({
  hooks: {
    'pages:extend' (routes) {
      const routeNameSplitter = '/'
      const root = createResolver(import.meta.url).resolve('./pages')

      function updateName(routes) {
```



```

    if (!routes) return

    for (const route of routes) {
      const relativePath = route.file.substring(root.length + 1)
      route.name = relativePath.slice(0, -4).replace(/\/index$/, '').replace(/\/g, routeNameSplitter)

      updateName(route.children)
    }
  }
  updateName(routes)
},
},
})
``

::

```

ESM Syntax

Nuxt 3 is an [ESM native framework](/docs/guide/concepts/esm). Although [`unjs/jiti`](https://github.com/unjs/jiti) provides semi compatibility when loading `nuxt.config` file, avoid any usage of `require` and `module.exports` in this file.

1. Change `module.exports` to `export default`
1. Change `const lib = require('lib')` to `import lib from 'lib'`

Async Configuration

In order to make Nuxt loading behavior more predictable, async config syntax is deprecated. Consider using Nuxt hooks for async operations.

Dotenv

Nuxt has built-in support for loading `.env` files. Avoid directly importing it from `nuxt.config`.

Modules

Nuxt and Nuxt Modules are now build-time-only.

Migration

1. Move all your `buildModules` into `modules`.
2. Check for Nuxt 3 compatibility of modules.
3. If you have any local modules pointing to a directory you should update this to point to the entry file:

```

``diff
  export default defineNuxtConfig({
    modules: [
-     '~/modules/my-module'
+     '~/modules/my-module/index'
    ]
  })
``

```

:::tip

If you are a module author, you can check out [more information about module compatibility](/docs/migration/module-authors) and [our module author guide](/docs/guide/going-further/modules).

::

Directory Changes

The `static/` (for storing static assets) has been renamed to `public/`. You can either rename your `static` directory to `public`, or keep the name by setting `dir.public` in your `nuxt.config`.

```
:read-more{to="/docs/guide/directory-structure/public"}
```

TypeScript

It will be much easier to migrate your application if you use Nuxt's TypeScript integration. This does not mean you need to write your application in TypeScript, just that Nuxt will provide automatic type hints for your editor.

You can read more about Nuxt's TypeScript support [in the docs](/docs/guide/concepts/typescript).

:::note

Nuxt can type-check your app using [`vue-tsc`](https://github.com/vuejs/language-tools/tree/master/packages/tsc) with `nuxt typecheck` command.

::

Migration

1. Create a `tsconfig.json` with the following content:

```

``json
{

```

```
"extends": "../.nuxt/tsconfig.json"
},
},
```

1. Run ``npx nuxi prepare`` to generate ``.nuxt/tsconfig.json``.

1. Install Volar following the instructions in the [docs](/docs/getting-started/introduction#prerequisites).

Vue Changes

There are a number of changes to what is recommended Vue best practice, as well as a number of breaking changes between Vue 2 and 3.

It is recommended to read the [Vue 3 migration guide](https://v3-migration.vuejs.org) and in particular the [breaking changes list](https://v3-migration.vuejs.org/breaking-changes).

It is not currently possible to use the [Vue 3 migration build](https://v3-migration.vuejs.org/migration-build.html) with Nuxt 3.

Vuex

Nuxt no longer provides a Vuex integration. Instead, the official Vue recommendation is to use ``pinia``, which has built-in Nuxt support via a [Nuxt module](https://pinia.vuejs.org/ssr/nuxt.html). [Find out more about pinia here](https://pinia.vuejs.org).

A simple way to provide global state management with pinia would be:

Install the [`@pinia/nuxt``](/modules/pinia) module:

```
```bash [Terminal]
yarn add pinia @pinia/nuxt
```
```

Enable the module in your nuxt configuration:

```
```ts [nuxt.config.ts]
import { defineNuxtConfig } from 'nuxt/config';

export default defineNuxtConfig({
 modules: ['@pinia/nuxt']
})
```
```

Create a ``store`` folder at the root of your application:

```
```ts [store/index.ts]
import { defineStore } from 'pinia'

export const useMainStore = defineStore('main', {
 state: () => ({
 counter: 0,
 }),
 actions: {
 increment() {
 // `this` is the store instance
 this.counter++
 },
 },
})
```
```

Create a [plugin](/docs/guide/directory-structure/plugins) file to globalize your store:

```
```ts [plugins/pinia.ts]
import { useMainStore } from '~/store'

export default defineNuxtPlugin(({ $pinia }) => {
 return {
 provide: {
 store: useMainStore($pinia)
 }
 }
})
```
```

If you want to keep using Vuex, you can manually migrate to Vuex 4 following [these steps](https://vuex.vuejs.org/guide/migrating-to-4-0-from-3-x.html).

Once it's done you will need to add the following plugin to your Nuxt app:

```
```ts [plugins/vuex.ts]
import store from '~/store'

export default defineNuxtPlugin(nuxtApp => {
```

```
 nuxtApp.vueApp.use(store);
 })
 ...
```

For larger apps, this migration can entail a lot of work. If updating Vuex still creates roadblocks, you may want to use the community module: [nuxt3-vuex-module](<https://github.com/vedmant/nuxt3-vuex#nuxt3-vuex-module>), which should work out of the box.

```

title: Nuxt Configuration
titleTemplate: '%s'
description: Discover all the options you can use in your nuxt.config.ts file.
navigation.icon: i-ph-gear-duotone

::note{icon="i-simple-icons-github" color="gray" to="https://github.com/nuxt/nuxt/tree/main/packages/schema/src/config"
target="_blank"}
This file is auto-generated from Nuxt source code.
::

<!-- GENERATED_CONFIG_DOCS -->
```

```

title: "Logging"
description: Nuxt Kit provides a set of utilities to help you work with logging. These functions allow you to log messages with extra features.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/logger.ts
 size: xs

```

Nuxt provides a logger instance that you can use to log messages with extra features. `useLogger` allows you to get a logger instance.

## `useLogger`

Returns a logger instance. It uses `[consola](https://github.com/unjs/consola)` under the hood.

### Type

```

```ts
function useLogger (tag?: string, options?: Partial<ConsolaOptions>): ConsolaInstance
```

```

### Parameters

#### `tag`

**Type:** `string`

**Optional:** `true`

A tag to prefix all log messages with.

#### `options`

**Type:** `Partial<ConsolaOptions>`

**Optional:** `true`

Consola configuration options

### Examples

```

```ts
import { defineNuxtModule, useLogger } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const logger = useLogger('my-module')

    logger.info('Hello from my module!')
  }
})
```

```

```

```ts
import { defineNuxtModule, useLogger } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const logger = useLogger('my-module', { level: options.quiet ? 0 : 3 })

    logger.info('Hello from my module!')
  }
})
```

```

```

title: Resolving
description: Nuxt Kit provides a set of utilities to help you resolve paths. These functions allow you to resolve paths relative to the current module, with unknown name or extension.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/resolve.ts
 size: xs

```

Sometimes you need to resolve a paths: relative to the current module, with unknown name or extension. For example, you may want to add a plugin that is located in the same directory as the module. To handle this cases, nuxt provides a set of utilities to resolve paths. `resolvePath` and `resolveAlias` are used to resolve paths relative to the current module. `findPath` is used to find first existing file in given paths. `createResolver` is used to create resolver relative to base path.

## `resolvePath`

Resolves full path to a file or directory respecting Nuxt alias and extensions options. If path could not be resolved, normalized input path will be returned.

### Type

```

````ts
async function resolvePath (path: string, options?: ResolvePathOptions): Promise<string>
````

```

### Parameters

#### `path`

**Type:** `string`  
**Required:** `true`

Path to resolve.

#### `options`

**Type:** `ResolvePathOptions`  
**Default:** `{}`

Options to pass to the resolver. This object can have the following properties:

- `cwd` (optional)

**Type:** `string`  
**Default:** `process.cwd()`  
 Current working directory.

- `alias` (optional)

**Type:** `Record<string, string>`  
**Default:** `{}`  
 Alias map.

- `extensions` (optional)

**Type:** `string[]`  
**Default:** `['.js', '.mjs', '.ts', '.jsx', '.tsx', '.json']`  
 Extensions to try.

### Examples

```

````ts
// https://github.com/P4scal/nuxt-headlessui
import { defineNuxtModule, resolvePath } from '@nuxt/kit'
import { join } from 'pathe'

const headlessComponents: ComponentGroup[] = [
  {
    relativePath: 'combobox/combobox.js',
    chunkName: 'headlessui/combobox',
    exports: [
      'Combobox',

```

```

        'ComboboxLabel',
        'ComboboxButton',
        'ComboboxInput',
        'ComboboxOptions',
        'ComboboxOption'
    ]
  },
]

export default defineNuxtModule({
  meta: {
    name: 'nuxt-headlessui',
    configKey: 'headlessui',
  },
  defaults: {
    prefix: 'Headless'
  },
  async setup (options) {
    const entrypoint = await resolvePath('@headlessui/vue')
    const root = join(entrypoint, '../components')

    for (const group of headlessComponents) {
      for (const e of group.exports) {
        addComponent(
          {
            name: e,
            export: e,
            filePath: join(root, group.relativePath),
            chunkName: group.chunkName,
            mode: 'all'
          }
        )
      }
    }
  }
})

```

`resolveAlias`

Resolves path aliases respecting Nuxt alias options.

Type

```

```ts
function resolveAlias (path: string, alias?: Record<string, string>): string
```

```

Parameters

`path`

Type: `string`

Required: `true`

Path to resolve.

`alias`

Type: `Record<string, string>`

Default: `{}`

Alias map. If not provided, it will be read from `nuxt.options.alias`.

`findPath`

Try to resolve first existing file in given paths.

Type

```

```ts
async function findPath (paths: string | string[], options?: ResolvePathOptions, pathType: 'file' | 'dir'): Promise<string | null>

interface ResolvePathOptions {
 cwd?: string
 alias?: Record<string, string>
 extensions?: string[]
}
```

```

Parameters

`paths`

****Type**:** `string | string[]`

****Required**:** `true`

A path or an array of paths to resolve.

`options`

****Type**:** `ResolvePathOptions`

****Default**:** `{}`

Options to pass to the resolver. This object can have the following properties:

- `cwd` (optional)

****Type**:** `string`

****Default**:** `process.cwd()`

Current working directory.

- `alias` (optional)

****Type**:** `Record<string, string>`

****Default**:** `{}`

Alias map.

- `extensions` (optional)

****Type**:** `string[]`

****Default**:** `['.js', '.mjs', '.ts', '.jsx', '.tsx', '.json']`

Extensions to try.

`pathType`

****Type**:** `'file' | 'dir'`

****Default**:** `'file'`

Type of path to resolve. If set to `'file'`, the function will try to resolve a file. If set to `'dir'`, the function will try to resolve a directory.

`createResolver`

Creates resolver relative to base path.

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/resolving-paths-and-injecting-assets-to-the-app?friend=nuxt" target="_blank"}

Watch Vue School video about createResolver.

::

Type

```ts

function createResolver (basePath: string | URL): Resolver

```
interface Resolver {
 resolve (...path: string[]): string
 resolvePath (path: string, options?: ResolvePathOptions): Promise<string>
}
```

```
interface ResolvePathOptions {
 cwd?: string
 alias?: Record<string, string>
 extensions?: string[]
}
```

``

### ### Parameters

#### #### `basePath`

**\*\*Type\*\*:** `string`



**\*\*Required\*\*:** `true`

Base path to resolve from.

### ### Examples

```
```:ts
// https://github.com/vuejs/pinia/blob/v2/packages/nuxt
import {
  defineNuxtModule,
  isNuxt2,
  createResolver,
} from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)

    nuxt.hook('modules:done', () => {
      if (isNuxt2()) {
        addPlugin(resolver.resolve('./runtime/plugin.vue2'))
      } else {
        addPlugin(resolver.resolve('./runtime/plugin.vue3'))
      }
    })
  }
})
```:
```

```

title: "Nitro"
description: Nuxt Kit provides a set of utilities to help you work with Nitro. These functions allow you to add server handlers, plugins, and prerender routes.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/nitro.ts
 size: xs

```

Nitro is an open source TypeScript framework to build ultra-fast web servers. Nuxt uses Nitro as its server engine. You can use `useNitro` to access the Nitro instance, `addServerHandler` to add a server handler, `addDevServerHandler` to add a server handler to be used only in development mode, `addServerPlugin` to add a plugin to extend Nitro's runtime behavior, and `addPrerenderRoutes` to add routes to be prerendered by Nitro.

## `addServerHandler`

Adds a nitro server handler. Use it if you want to create server middleware or custom route.

### Type

```

```ts
function addServerHandler (handler: NitroEventHandler): void

export interface NitroEventHandler {
  handler: string;
  route?: string;
  middleware?: boolean;
  lazy?: boolean;
  method?: string;
}
```

```

### Parameters

#### `handler`

**Type:** `NitroEventHandler`

**Required:** `true`

A handler object with the following properties:

- `handler` (required)

**Type:** `string`

Path to event handler.

- `route` (optional)

**Type:** `string`

Path prefix or route. If an empty string used, will be used as a middleware.

- `middleware` (optional)

**Type:** `boolean`

Specifies this is a middleware handler. Middleware are called on every route and should normally return nothing to pass to the next handlers.

- `lazy` (optional)

**Type:** `boolean`

Use lazy loading to import handler.

- `method` (optional)

**Type:** `string`

Router method matcher. If handler name contains method name, it will be used as a default value.

### Examples

::code-group

```

```ts [module.ts]
// https://github.com/nuxt-modules/robots
import { createResolver, defineNuxtModule, addServerHandler } from '@nuxt/kit'

```

```
export default defineNuxtModule({
  setup(options) {
    const resolver = createResolver(import.meta.url)

    addServerHandler({
      route: '/robots.txt'
      handler: resolver.resolve('./runtime/robots.get.ts')
    })
  }
})
```

```

```
```ts [runtime/robots.get.ts]
export default defineEventHandler(() => {
  return {
    body: `User-agent: *\nDisallow: /`
  }
})
```
```

::

## `addDevServerHandler`

Adds a nitro server handler to be used only in development mode. This handler will be excluded from production build.

### Type

```
```ts
function addDevServerHandler (handler: NitroDevEventHandler): void

export interface NitroDevEventHandler {
  handler: EventHandler;
  route?: string;
}
```
```

### Parameters

#### `handler`

**\*\*Type\*\*:** `NitroEventHandler`

**\*\*Required\*\*:** `true`

A handler object with the following properties:

- `handler` (required)

**\*\*Type\*\*:** `string`

The event handler.

- `route` (optional)

**\*\*Type\*\*:** `string`

Path prefix or route. If an empty string used, will be used as a middleware.

### Examples

::code-group

```
```ts [module.ts]
import { createResolver, defineNuxtModule, addDevServerHandler } from '@nuxt/kit'

export default defineNuxtModule({
  setup() {
    const resolver = createResolver(import.meta.url)

    addDevServerHandler({
      handler: () => {
        return {
          body: `Response generated at ${new Date().toISOString()}`
        }
      },
      route: '/_handler'
    })
  }
})
```
```

::

```

````ts
// https://github.com/nuxt-modules/tailwindcss
import { joinURL } from 'ufo'
import { defineNuxtModule, addDevServerHandler } from '@nuxt/kit'

export default defineNuxtModule({
  async setup(options) {
    const route = joinURL(nuxt.options.app?.baseUrl, '/_tailwind')

    // @ts-ignore
    const createServer = await import('tailwind-config-viewer/server/index.js').then(r => r.default || r) as any
    const viewerDevMiddleware = createServer({ tailwindConfigProvider: () => options, routerPrefix: route }).asMiddleware()

    addDevServerHandler({ route, handler: viewerDevMiddleware })
  }
})
````

```

## ## `useNitro`

Returns the Nitro instance.

```

::warning
You can call `useNitro()` only after `ready` hook.
::

::note
Changes to the Nitro instance configuration are not applied.
::

```

## ### Type

```

````ts
function useNitro (): Nitro

export interface Nitro {
  options: NitroOptions;
  scannedHandlers: NitroEventHandler[];
  vfs: Record<string, string>;
  hooks: Hookable<NitroHooks>;
  unimport?: Unimport;
  logger: ConsolaInstance;
  storage: Storage;
  close: () => Promise<void>;
  updateConfig: (config: NitroDynamicConfig) => void | Promise<void>;
}
````

```

## ### Examples

```

````ts
// https://github.com/nuxt/nuxt/blob/4e05650cde31ca73be4d14b1f0d23c7854008749/packages/nuxt/src/core/nuxt.ts#L404
import { defineNuxtModule, useNitro, addPlugin, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)

    nuxt.hook('ready', () => {
      const nitro = useNitro()
      if (nitro.options.static && nuxt.options.experimental.payloadExtraction === undefined) {
        console.warn('Using experimental payload extraction for full-static output. You can opt-out by setting `experimental.payloadExtraction` to `false`.')
        nuxt.options.experimental.payloadExtraction = true
      }
      nitro.options.replace['process.env.NUXT_PAYLOAD_EXTRACTION'] = String(!nuxt.options.experimental.payloadExtraction)
      nitro.options._config.replace!['process.env.NUXT_PAYLOAD_EXTRACTION'] = String(!nuxt.options.experimental.payloadExtraction)

      if (!nuxt.options.dev && nuxt.options.experimental.payloadExtraction) {
        addPlugin(resolver.resolve(nuxt.options.appDir, 'plugins/payload.client'))
      }
    })
  }
})
````

```

## ## `addServerPlugin`

Add plugin to extend Nitro's runtime behavior.

```

::tip

```

You can read more about Nitro plugins in the [Nitro documentation](https://nitro.unjs.io/guide/plugins).

::

### ### Type

```
````ts
function addServerPlugin (plugin: string): void
````
```

### ### Parameters

#### `plugin`

**Type:** `string`

**Required:** `true`

Path to the plugin. The plugin must export a function that accepts Nitro instance as an argument.

### ### Examples

::code-group

```
````ts [module.ts]
import { createResolver, defineNuxtModule, addServerPlugin } from '@nuxt/kit'

export default defineNuxtModule({
  setup() {
    const resolver = createResolver(import.meta.url)
    addServerPlugin(resolver.resolve('./runtime/plugin.ts'))
  }
})
````
```

```
````ts [runtime/plugin.ts]
export default defineNitroPlugin((nitroApp) => {
  nitroApp.hooks.hook("request", (event) => {
    console.log("on request", event.path);
  });

  nitroApp.hooks.hook("beforeResponse", (event, { body }) => {
    console.log("on response", event.path, { body });
  });

  nitroApp.hooks.hook("afterResponse", (event, { body }) => {
    console.log("on after response", event.path, { body });
  });
});
````
```

::

### ## `addPrerenderRoutes`

Add routes to be prerendered to Nitro.

### ### Type

```
````ts
function addPrerenderRoutes (routes: string | string[]): void
````
```

### ### Parameters

#### `routes`

**Type:** `string | string[]`

**Required:** `true`

A route or an array of routes to prerender.

### ### Examples

```
````ts
import { defineNuxtModule, addPrerenderRoutes } from '@nuxt/kit'

export default defineNuxtModule({
  meta: {
    name: 'nuxt-sitemap',
    configKey: 'sitemap',
  },
  defaults: {
```

```

    sitemapUrl: '/sitemap.xml',
    prerender: true,
  },
  setup(options) {
    if (options.prerender) {
      addPrerenderRoutes(options.sitemapUrl)
    }
  }
})
`

```

`addServerImportsDir`

Add a directory to be scanned for auto-imports by Nitro.

Type

```

`
`ts
function addServerImportsDir (dirs: string | string[], opts: { prepend?: boolean }): void
`
`

```

Parameters

`dirs`

Type: `string | string[]`

Required: `true`

A directory or an array of directories to register to be scanned by Nitro

Examples

```

`
`ts
import { defineNuxtModule, createResolver, addServerImportsDir } from '@nuxt/kit'

export default defineNuxtModule({
  meta: {
    name: 'my-module',
    configKey: 'myModule',
  },
  setup(options) {
    const resolver = createResolver(import.meta.url)
    addServerImportsDir(resolver.resolve('./runtime/server/utils'))
  }
})
`
`

```

`addServerScanDir`

Add directories to be scanned by Nitro. It will check for subdirectories, which will be registered just like the `~/server` folder is.

Type

```

`
`ts
function addServerScanDir (dirs: string | string[], opts: { prepend?: boolean }): void
`
`

```

Parameters

`dirs`

Type: `string | string[]`

Required: `true`

A directory or an array of directories to register to be scanned for by Nitro as server dirs.

Examples

```

`
`ts
import { defineNuxtModule, createResolver, addServerScanDir } from '@nuxt/kit'
export default defineNuxtModule({
  meta: {
    name: 'my-module',
    configKey: 'myModule',
  },
  setup(options) {
    const resolver = createResolver(import.meta.url)
    addServerScanDir(resolver.resolve('./runtime/server'))
  }
})
`
`

```



```
---
title: 'Nuxt API Reference'
titleTemplate: '%s'
description: 'Explore all Nuxt Internals: Components, Composables, Utils, Commands and more.'
navigation: false
surround: false
---
```

```
::card-group
::card{icon="i-ph-cube-duotone" title="Components" to="/docs/api/components/client-only"}
Explore Nuxt built-in components for pages, layouts, head, and more.
::
::card{icon="i-ph-arrows-left-right-duotone" title="Composables" to="/docs/api/composables/use-app-config"}
Discover Nuxt composable functions for data-fetching, head management and more.
::
::card{icon="i-ph-function-duotone" title="Utils" to="/docs/api/utils/dollarfetch"}
Learn about Nuxt utility functions for navigation, error handling and more.
::
::card{icon="i-ph-terminal-window-duotone" title="Commands" to="/docs/api/commands/add"}
List of Nuxt CLI commands to init, analyze, build, and preview your application.
::
::card{icon="i-ph-toolbox-duotone" title="Nuxt Kit" to="/docs/api/kit/modules"}
Understand Nuxt Kit utilities to create modules and control Nuxt.
::
::card{icon="i-ph-brain-duotone" title="Advanced" to="/docs/api/advanced/hooks"}
Go deep in Nuxt internals with Nuxt lifecycle hooks.
::
::card{icon="i-ph-gear-duotone" title="Nuxt Configuration" to="/docs/api/nuxt-config"}
Explore all Nuxt configuration options to customize your application.
::
::
```



```
---
title: Meta Tags
description: 'Learn how to migrate from Nuxt 2 to Nuxt Bridge new meta tags.'
---
```

If you need to access the component state with ``head``, you should migrate to using `[`useHead`](/docs/api/composables/use-head)` .

If you need to use the Options API, there is a ``head()`` method you can use when you use ``defineNuxtComponent`` .

Migration

Set ``bridge.meta``

```
```js
import { defineNuxtConfig } from '@nuxt/bridge'
export default defineNuxtConfig({
 bridge: {
 meta: true,
 nitro: false // If migration to Nitro is complete, set to true
 }
})
```
```

Update head properties

In your ``nuxt.config``, rename ``head`` to ``meta``. (Note that objects no longer have a ``hid`` key for deduplication.)

::code-group

```
```ts [Nuxt 2]
export default {
 head: {
 titleTemplate: '%s - Nuxt',
 meta: [
 { charset: 'utf-8' },
 { name: 'viewport', content: 'width=device-width, initial-scale=1' },
 { hid: 'description', name: 'description', content: 'Meta description' }
]
 }
}
```
```

```
```ts [Nuxt 3]
export default defineNuxtConfig({
 app: {
 head: {
 titleTemplate: '%s - Nuxt',
 meta: [
 { charset: 'utf-8' },
 { name: 'viewport', content: 'width=device-width, initial-scale=1' },
 { name: 'description', content: 'Meta description' }
]
 }
 }
})
```
```

::

``useHead`` Composables

Nuxt Bridge provides a new Nuxt 3 meta API that can be accessed with a new `[`useHead`](/docs/api/composables/use-head)` composable.

```
```vue
<script setup lang="ts">
useHead({
 title: 'My Nuxt App',
})
</script>
```
```

::tip

This `[`useHead`](/docs/api/composables/use-head)` composable uses ``@unhead/vue`` under the hood (rather than ``vue-meta``) to manipulate your ``<head>``.

::

::warning

We recommend not using the native Nuxt 2 ``head()`` properties in addition to `[`useHead`](/docs/api/composables/use-head)` , as they may conflict.

::

For more information on how to use this composable, see [the docs](/docs/getting-started/seo-meta).

Options API

```
```\nvue\n<script>\n// if using options API `head` method you must use `defineNuxtComponent`\nexport default defineNuxtComponent({\n  head (nuxtApp) {\n    // `head` receives the nuxt app but cannot access the component instance\n    return {\n      meta: [{\n        name: 'description',\n        content: 'This is my page description.'\n      }]\n    }\n  }\n})\n</script>\n```\n
```

::warning  
Possible breaking change: `head` receives the nuxt app but cannot access the component instance. If the code in your `head` tries to access the data object through `this` or `this.\$data`, you will need to migrate to the `useHead` composable.  
::

## ## Title Template

If you want to use a function (for full control), then this cannot be set in your nuxt.config, and it is recommended instead to set it within your `/layouts` directory.

```
```\nvue [layouts/default.vue]\n<script setup lang="ts">\nuseHead({\n  titleTemplate: (titleChunk) => {\n    return titleChunk ? `${titleChunk} - Site Title` : 'Site Title';\n  }\n})\n</script>\n```\n
```

```
---
title: Runtime Config
description: 'Nuxt provides a runtime config API to expose configuration and secrets within your application.'
---
```

```
:::warning
When using `runtimeConfig` option, [nitro](/docs/bridge/nitro) must have been configured.
:::
```

Update Runtime Config

Nuxt 3 approaches runtime config differently than Nuxt 2, using a new combined `runtimeConfig` option.

First, you'll need to combine your `publicRuntimeConfig` and `privateRuntimeConfig` properties into a new one called `runtimeConfig`, with the public config within a key called `public`.

```
```diff
// nuxt.config.js
- privateRuntimeConfig: {
- apiKey: process.env.NUXT_API_KEY || 'super-secret-key'
- },
- publicRuntimeConfig: {
- websiteURL: 'https://public-data.com'
- }
+ runtimeConfig: {
+ apiKey: process.env.NUXT_API_KEY || 'super-secret-key',
+ public: {
+ websiteURL: 'https://public-data.com'
+ }
+ }
```
```

This also means that when you need to access public runtime config, it's behind a property called `public`. If you use public runtime config, you'll need to update your code.

```
```diff
// MyWidget.vue
- <div>Website: {{ $config.websiteURL }}</div>
+ <div>Website: {{ $config.public.websiteURL }}</div>
```
```

```
---
title: Nitro
description: 'Activate Nitro to your Nuxt 2 application with Nuxt Bridge.'
---
```

Remove Modules

- Remove `@nuxt/nitro`: Bridge injects same functionality

Update Config

```
``ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
  bridge: {
    nitro: true
  }
})
``
```

Update Your Scripts

You will also need to update your scripts within your `package.json` to reflect the fact that Nuxt will now produce a Nitro server as build output.

Install Nuxi

Install `nuxi` as a development dependency:

::code-group

```
``bash [yarn]
yarn add --dev nuxi
``
```

```
``bash [npm]
npm install -D nuxi
``
```

::

Nuxi

Nuxt 3 introduced the new Nuxt CLI command [`nuxi`](/docs/api/commands/add). Update your scripts as follows to leverage the better support from Nuxt Bridge:

```
``diff
{
  "scripts": {
-   "dev": "nuxt",
+   "dev": "nuxi dev",
-   "build": "nuxt build",
+   "build": "nuxi build",
-   "start": "nuxt start",
+   "start": "nuxi preview"
  }
}
``
```

::tip
If `nitro: false`, use the `nuxt2` command.
::

Static Target

If you have set `target: 'static'` in your `nuxt.config` then you need to ensure that you update your build script to be `nuxi generate`.

```
``json [package.json]
{
  "scripts": {
    "build": "nuxi generate"
  }
}
``
```

Server Target

For all other situations, you can use the `nuxi build` command.

```
``json [package.json]
{
```

```
"scripts": {  
  "build": "nuxi build",  
  "start": "nuxi preview"  
}  
},  
```
```

```
Exclude Built Nitro Folder From Git
```

Add the folder ``.output`` to the ``.gitignore`` file.

```
Ensure Everything Goes Well
```

â€”i, Try with ``nuxi dev`` and ``nuxi build`` (or ``nuxi generate``) to see if everything goes well.

```

title: Meta Tags
description: Manage your meta tags, from Nuxt 2 to Nuxt 3.

```

Nuxt 3 provides several different ways to manage your meta tags:

1. Through your ``nuxt.config``.
2. Through the `[`useHead`](/docs/api/composables/use-head)` `[composable](/docs/getting-started/seo-meta)`
3. Through `[global meta components](/docs/getting-started/seo-meta)`

You can customize ``title``, ``titleTemplate``, ``base``, ``script``, ``noscript``, ``style``, ``meta``, ``link``, ``htmlAttrs`` and ``bodyAttrs``.

::tip

Nuxt currently uses `[`vueuse/head`](https://github.com/vueuse/head)` to manage your meta tags, but implementation details may change.

::

:read-more{to="/docs/getting-started/seo-meta"}

## ## Migration

1. In your ``nuxt.config``, rename ``head`` to ``meta``. Consider moving this shared meta configuration into your ``app.vue`` instead. (Note that objects no longer have a ``hid`` key for deduplication.)
2. If you need to access the component state with ``head``, you should migrate to using `[`useHead`](/docs/api/composables/use-head)`. You might also consider using the built-in meta-components.
3. If you need to use the Options API, there is a ``head()`` method you can use when you use ``defineNuxtComponent``.

### ### useHead

::code-group

```
```vue [Nuxt 2]
<script>
export default {
  data: () => ({
    title: 'My App',
    description: 'My App Description'
  })
  head () {
    return {
      title: this.title,
      meta: [{
        hid: 'description',
        name: 'description',
        content: this.description
      }]
    }
  }
}
</script>
```
```

```
```vue [Nuxt 3]
<script setup lang="ts">
const title = ref('My App')
const description = ref('My App Description')

// This will be reactive when you change title/description above
useHead({
  title,
  meta: [{
    name: 'description',
    content: description
  }]
})
</script>
```
```

::

### ### Meta-components

Nuxt 3 also provides meta components that you can use to accomplish the same task. While these components look similar to HTML tags, they are provided by Nuxt and have similar functionality.

::code-group

```
```vue [Nuxt 2]
<script>
export default {
  head () {
    return {
```

```

    title: 'My App',
    meta: [{
      hid: 'description',
      name: 'description',
      content: 'My App Description'
    }]
  }
}
</script>
```

```

```

```vue [Nuxt 3]
<template>
  <div>
    <Head>
      <Title>My App</Title>
      <Meta name="description" content="My app description"/>
    </Head>
    <!-- -->
  </div>
</template>
```

```

::

::important

1. Make sure you use capital letters for these component names to distinguish them from native HTML elements (`<Title>` rather than `<title>`).
2. You can place these components anywhere in your template for your page.

::

### ### Options API

```

```vue [Nuxt 3 (Options API)]
<script>
// if using options API `head` method you must use `defineNuxtComponent`
export default defineNuxtComponent({
  head (nuxtApp) {
    // `head` receives the nuxt app but cannot access the component instance
    return {
      meta: [{
        name: 'description',
        content: 'This is my page description.'
      }]
    }
  }
})
</script>
```

```

```

title: Plugins and Middleware
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 plugins and middleware.'

```

## ## Plugins

Plugins now have a different format, and take only one argument (`nuxtApp`).

::code-group

```
```js [Nuxt 2]
export default (ctx, inject) => {
  inject('injected', () => 'my injected function')
}
```

```ts [Nuxt 3]
export default defineNuxtPlugin(nuxtApp => {
  // now available on `nuxtApp.$injected`
  nuxtApp.provide('injected', () => 'my injected function')

  // You can alternatively use this format, which comes with automatic type support
  return {
    provide: {
      injected: () => 'my injected function'
    }
  }
})
```
```

```
::

:read-more{to="/docs/guide/directory-structure/plugins"}

::read-more{to="/docs/api/composables/use-nuxt-app"}
Read more about the format of `nuxtApp`.
::
```

## ### Migration

1. Migrate your plugins to use the `defineNuxtPlugin` helper function.
2. Remove any entries in your `nuxt.config` plugins array that are located in your `plugins/` folder. All files in this directory at the top level (and any index files in any subdirectories) will be automatically registered. Instead of setting `mode` to `client` or `server`, you can indicate this in the file name. For example, `~/plugins/my-plugin.client.ts` will only be loaded on client-side.

## ## Route Middleware

Route middleware has a different format.

::code-group

```
```js [Nuxt 2]
export default function ({ store, redirect }) {
  // If the user is not authenticated
  if (!store.state.authenticated) {
    return redirect('/login')
  }
}
```

```ts [Nuxt 3]
export default defineNuxtRouteMiddleware((to, from) => {
  const auth = useState('auth')
  if (!auth.value.authenticated) {
    return navigateTo('/login')
  }
})
```

::
```

Much like Nuxt 2, route middleware placed in your `~/middleware` folder is automatically registered. You can then specify it by name in a component. However, this is done with `definePageMeta` rather than as a component option.

`navigateTo` is one of a number of route helper functions.

:read-more{to="/docs/guide/directory-structure/middleware"}

## ### Migration

1. Migrate your route middleware to use the `defineNuxtRouteMiddleware` helper function.



1. Any global middleware (such as in your `nuxt.config`) can be placed in your `~/middleware` folder with a `.global` extension, for example `~/middleware/auth.global.ts`.

```

title: Pages and Layouts
description: Learn how to migrate from Nuxt 2 to Nuxt 3 pages and layouts.

```

## ## `app.vue`

Nuxt 3 provides a central entry point to your app via `~/app.vue`.

```
::note
If you don't have an `app.vue` file in your source directory, Nuxt will use its own default version.
::
```

This file is a great place to put any custom code that needs to be run once when your app starts up, as well as any components that are present on every page of your app. For example, if you only have one layout, you can move this to `app.vue` instead.

```
:read-more{to="/docs/guide/directory-structure/app"}
```

```
:link-example{to="/docs/examples/hello-world"}
```

## ### Migration

Consider creating an `app.vue` file and including any logic that needs to run once at the top-level of your app. You can check out [\[an example here\]](/docs/guide/directory-structure/app).

## ## Layouts

If you are using layouts in your app for multiple pages, there is only a slight change required.

In Nuxt 2, the `` component is used within a layout to render the current page. In Nuxt 3, layouts use slots instead, so you will have to replace that component with a `[Read more about layouts].

You will also need to change how you define the layout used by a page using the `definePageMeta` compiler macro. Layouts will be kebab-cased. So `layouts/customLayout.vue` becomes `custom-layout` when referenced in your page.

## ### Migration

1. Replace `

```
```diff [layouts/custom.vue]
<template>
  <div id="app-layout">
    <main>
-     <Nuxt />
+     <slot />
    </main>
  </div>
</template>
```
```

2. Use `[`definePageMeta`]` [\(/docs/api/utils/define-page-meta\)](/docs/api/utils/define-page-meta) to select the layout used by your page.

```
```diff [pages/index.vue]
<script>
+ definePageMeta({
+   layout: 'custom'
+ })
- export default {
-   layout: 'custom'
- }
</script>
```
```

3. Move `~/layouts/\_error.vue` to `~/error.vue`. See [\[the error handling docs\]](/docs/getting-started/error-handling). If you want to ensure that this page uses a layout, you can use `<NuxtLayout>` [\(/docs/guide/directory-structure/layouts\)](/docs/guide/directory-structure/layouts) directly within `error.vue`:

```
```vue [error.vue]
<template>
  <div>
    <NuxtLayout name="default">
      <!-- -->
    </NuxtLayout>
  </div>
</template>
```
```

## ## Pages

Nuxt 3 ships with an optional `vue-router` integration triggered by the existence of a `[`pages/`]` [\(/docs/guide/directory-structure/pages\)](/docs/guide/directory-structure/pages) directory in your source directory. If you only have a single page, you may consider instead moving it to `app.vue` for a lighter build.

## ### Dynamic Routes

The format for defining dynamic routes in Nuxt 3 is slightly different from Nuxt 2, so you may need to rename some of the

files within `pages/`.

1. Where you previously used `\_id` to define a dynamic route parameter you now use `[id]`.
2. Where you previously used `\_.vue` to define a catch-all route, you now use `[...slug].vue`.

### ### Nested Routes

In Nuxt 2, you will have defined any nested routes (with parent and child components) using `` and ``. In Nuxt 3, these have been replaced with a single `` component.

### ### Page Keys and Keep-alive Props

If you were passing a custom page key or keep-alive props to `

```
:read-more{to="/docs/guide/directory-structure/pages#special-metadata"}
```

### ### Page and Layout Transitions

If you have been defining transitions for your page or layout directly in your component options, you will now need to use `definePageMeta` to set the transition. Since Vue 3, [-enter and -leave CSS classes have been renamed](https://v3-migration.vuejs.org/breaking-changes/transition.html). The `style` prop from `

```
:read-more{to="/docs/getting-started/transitions"}
```

### ### Migration

1. Rename any pages with dynamic parameters to match the new format.
2. Update `- 3. If you're using the Composition API, you can also migrate `this.\$route` and `this.\$router` to use [`useRoute`] (/docs/api/composables/use-route) and [`useRouter`] (/docs/api/composables/use-router) composables.

#### #### Example: Dynamic Routes

```
::code-group
```

```
``` [Nuxt 2]
- URL: /users
- Page: /pages/users/index.vue

- URL: /users/some-user-name
- Page: /pages/users/_user.vue
- Usage: params.user

- URL: /users/some-user-name/edit
- Page: /pages/users/_user/edit.vue
- Usage: params.user

- URL: /users/anything-else
- Page: /pages/users/_ .vue
- Usage: params.pathMatch
```
```

```
``` [Nuxt 3]
- URL: /users
- Page: /pages/users/index.vue

- URL: /users/some-user-name
- Page: /pages/users/[user].vue
- Usage: params.user

- URL: /users/some-user-name/edit
- Page: /pages/users/[user]/edit.vue
- Usage: params.user

- URL: /users/anything-else
- Page: /pages/users/[...slug].vue
- Usage: params.slug
```
```

```
::
```

#### #### Example: Nested Routes and `definePageMeta`

```
::code-group
```

```
```vue [Nuxt 2]
<template>
  <div>
    <NuxtChild keep-alive :keep-alive-props="{ exclude: ['modal'] }" :nuxt-child-key="$route.slug" />
  </div>
</template>
```

```

<script>
export default {
  transition: 'page' // or { name: 'page' }
}
</script>
```

```vue [Nuxt 3]
<template>
  <div>
    <NuxtPage />
  </div>
</template>

<script setup lang="ts">
// This compiler macro works in both <script> and <script setup>
definePageMeta({
  // you can also pass a string or a computed property
  key: route => route.slug,
  transition: {
    name: 'page',
  },
  keepalive: {
    exclude: ['modal']
  },
})
</script>
```

```

::

### ## `<NuxtLink>` Component

Most of the syntax and functionality are the same for the global [NuxtLink](/docs/api/components/nuxt-link) component. If you have been using the shortcut `` format, you should update this to use ``.

`<NuxtLink>` is now a drop-in replacement for `_all_` links, even external ones. You can read more about it, and how to extend it to provide your own link component.

```
:read-more{to="/docs/api/components/nuxt-link"}
```

### ## Programmatic Navigation

When migrating from Nuxt 2 to Nuxt 3, you will have to update how you programmatically navigate your users. In Nuxt 2, you had access to the underlying Vue Router with `this.$router`. In Nuxt 3, you can use the `navigateTo()` utility method which allows you to pass a route and parameters to Vue Router.

```

::warning
Ensure to always `await` on [navigateTo](/docs/api/utils/navigate-to) or chain its result by returning from functions.
::

```

::code-group

```

```vue [Nuxt 2]
<script>
export default {
  methods: {
    navigate(){
      this.$router.push({
        path: '/search',
        query: {
          name: 'first name',
          type: '1'
        }
      })
    }
  }
}
</script>
```

```

```

```vue [Nuxt 3]
<script setup lang="ts">
function navigate(){
  return navigateTo({
    path: '/search',
    query: {
      name: 'first name',
      type: '1'
    }
  })
}

```

</script>
```  
  
::

```

title: "Examples"
description: Examples of Nuxt Kit utilities in use.

```

## ## Accessing Nuxt Vite Config

If you are building an integration that needs access to the runtime Vite or webpack config that Nuxt uses, it is possible to extract this using Kit utilities.

Some examples of projects doing this already:

- [histoire](https://github.com/histoire-dev/histoire/blob/main/packages/histoire-plugin-nuxt/src/index.ts)
- [nuxt-vitest](https://github.com/danielroe/nuxt-vitest/blob/main/packages/nuxt-vitest/src/config.ts)
- [@storybook-vue/nuxt](https://github.com/storybook-vue/storybook-nuxt/blob/main/packages/storybook-nuxt/src/preset.ts)

Here is a brief example of how you might access the Vite config from a project; you could implement a similar approach to get the webpack configuration.

```
```js
import { loadNuxt, buildNuxt } from '@nuxt/kit'

// https://github.com/nuxt/nuxt/issues/14534
async function getViteConfig() {
  const nuxt = await loadNuxt({ cwd: process.cwd(), dev: false, overrides: { ssr: false } })
  return new Promise((resolve, reject) => {
    nuxt.hook('vite:extendConfig', (config, { isClient }) => {
      if (isClient) {
        resolve(config)
        throw new Error('_stop_')
      }
    })
    buildNuxt(nuxt).catch((err) => {
      if (!err.toString().includes('_stop_')) {
        reject(err)
      }
    })
  }).finally(() => nuxt.close())
}

const viteConfig = await getViteConfig()
console.log(viteConfig)
```
```

```

title: "Programmatic Usage"
description: Nuxt Kit provides a set of utilities to help you work with Nuxt programmatically. These functions allow you to load Nuxt, build Nuxt, and load Nuxt configuration.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/loader
 size: xs

```

Programmatic usage can be helpful when you want to use Nuxt programmatically, for example, when building a [CLI tool] (<https://github.com/nuxt/cli>) or [test utils](<https://github.com/nuxt/nuxt/tree/main/packages/test-utils>).

## ## `loadNuxt`

Load Nuxt programmatically. It will load the Nuxt configuration, instantiate and return the promise with Nuxt instance.

### ### Type

```

```ts
async function loadNuxt (loadOptions?: LoadNuxtOptions): Promise<Nuxt>

```

```

interface LoadNuxtOptions extends LoadNuxtConfigOptions {
  dev?: boolean
  ready?: boolean
}
```

```

### ### Parameters

#### #### `loadOptions`

**\*\*Type\*\*:** `LoadNuxtOptions`

**\*\*Default\*\*:** `{}`

Loading conditions for Nuxt. `loadNuxt` uses [c12](<https://github.com/unjs/c12>) under the hood, so it accepts the same options as `c12.loadConfig` with some additional options:

- `dev` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `false`

If set to `true`, Nuxt will be loaded in development mode.

- `ready` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `true`

If set to `true`, Nuxt will be ready to use after the `loadNuxt` call. If set to `false`, you will need to call `nuxt.ready()` to make sure Nuxt is ready to use.

## ## `buildNuxt`

Build Nuxt programmatically. It will invoke the builder (currently [vite-builder](<https://github.com/nuxt/nuxt/tree/main/packages/vite>) or [webpack-builder](<https://github.com/nuxt/nuxt/tree/main/packages/webpack>)) to bundle the application.

### ### Type

```

```ts
async function buildNuxt (nuxt: Nuxt): Promise<any>
```

```

### ### Parameters

#### #### `nuxt`

**\*\*Type\*\*:** `Nuxt`

**\*\*Required\*\*:** `true`

Nuxt instance to build. It can be retrieved from the context via `useNuxt()` call.

## ## `loadNuxtConfig`

Load Nuxt configuration. It will return the promise with the configuration object.

```
Type

```ts
async function loadNuxtConfig (options: LoadNuxtConfigOptions): Promise<NuxtOptions>
```

Parameters

`options`

Type: `LoadNuxtConfigOptions`

Required: `true`

Options to pass in [c12](https://github.com/unjs/c12#options) loadConfig call.

`writeTypes`

Generates tsconfig.json and writes it to the project buildDir.

Type

```ts
function writeTypes (nuxt?: Nuxt): void

interface Nuxt {
  options: NuxtOptions
  hooks: Hookable<NuxtHooks>
  hook: Nuxt['hooks']['hook']
  callHook: Nuxt['hooks']['callHook']
  addHooks: Nuxt['hooks']['addHooks']
  ready: () => Promise<void>
  close: () => Promise<void>
  server?: any
  vfs: Record<string, string>
  apps: Record<string, NuxtApp>
}
```

Parameters

`nuxt`

Type: `Nuxt`

Required: `true`

Nuxt instance to build. It can be retrieved from the context via useNuxt() call.
```



```

title: Builder
description: Nuxt Kit provides a set of utilities to help you work with the builder. These functions allow you to extend the
webpack and vite configurations.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/build.ts
 size: xs

```

Nuxt have builders based on [webpack](https://github.com/nuxt/nuxt/tree/main/packages/webpack) and [vite](https://github.com/nuxt/nuxt/tree/main/packages/vite). You can extend the config passed to each one using `extendWebpackConfig` and `extendViteConfig` functions. You can also add additional plugins via `addVitePlugin`, `addWebpackPlugin` and `addBuildPlugin`.

## ## `extendWebpackConfig`

Extends the webpack configuration. Callback function can be called multiple times, when applying to both client and server builds.

### ### Type

```
``ts
function extendWebpackConfig (callback: ((config: WebpackConfig) => void), options?: ExtendWebpackConfigOptions): void

export interface ExtendWebpackConfigOptions {
 dev?: boolean
 build?: boolean
 server?: boolean
 client?: boolean
 prepend?: boolean
}
```

[::read-more{to="https://webpack.js.org/configuration" target="\\_blank" color="gray" icon="i-simple-icons-webpack"}](https://webpack.js.org/configuration)  
Checkout webpack website for more information about its configuration.  
::

### ### Parameters

#### #### `callback`

**Type**: `(config: WebpackConfig) => void`

**Required**: `true`

A callback function that will be called with the webpack configuration object.

#### #### `options`

**Type**: `ExtendWebpackConfigOptions`

**Default**: `{}`

Options to pass to the callback function. This object can have the following properties:

- `dev` (optional)

**Type**: `boolean`

**Default**: `true`

If set to `true`, the callback function will be called when building in development mode.

- `build` (optional)

**Type**: `boolean`

**Default**: `true`

If set to `true`, the callback function will be called when building in production mode.

- `server` (optional)

**Type**: `boolean`

**Default**: `true`

If set to `true`, the callback function will be called when building the server bundle.

- `client` (optional)

**\*\*Type\*\*:** ``boolean``

**\*\*Default\*\*:** ``true``

If set to ``true``, the callback function will be called when building the client bundle.

- ``prepend`` (optional)

**\*\*Type\*\*:** ``boolean``

If set to ``true``, the callback function will be prepended to the array with ``unshift()`` instead of ``push()``.

### ### Examples

```
````ts
import { defineNuxtModule, extendWebpackConfig } from '@nuxt/kit'

export default defineNuxtModule({
  setup() {
    extendWebpackConfig((config) => {
      config.module?.rules.push({
        test: /\.txt$/,
        use: 'raw-loader'
      })
    })
  }
})
````
```

### ## ``extendViteConfig``

Extends the Vite configuration. Callback function can be called multiple times, when applying to both client and server builds.

#### ### Type

```
````ts
function extendViteConfig (callback: ((config: ViteConfig) => void), options?: ExtendViteConfigOptions): void

export interface ExtendViteConfigOptions {
  dev?: boolean
  build?: boolean
  server?: boolean
  client?: boolean
  prepend?: boolean
}
```

[::read-more{to="https://vitejs.dev/config" target="_blank" color="gray" icon="i-simple-icons-vite"}](https://vitejs.dev/config)
Checkout Vite website for more information about its configuration.
::

Parameters

``callback``

****Type**:** ``(config: ViteConfig) => void``

****Required**:** ``true``

A callback function that will be called with the Vite configuration object.

``options``

****Type**:** ``ExtendViteConfigOptions``

****Default**:** ``{}``

Options to pass to the callback function. This object can have the following properties:

- ``dev`` (optional)

****Type**:** ``boolean``

****Default**:** ``true``

If set to ``true``, the callback function will be called when building in development mode.

- ``build`` (optional)

****Type**:** ``boolean``

****Default**:** ``true``

If set to `true`, the callback function will be called when building in production mode.

- `server` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building the server bundle.

- `client` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building the client bundle.

- `prepend` (optional)

****Type**:** `boolean`

If set to `true`, the callback function will be prepended to the array with `unshift()` instead of `push()`.

Examples

```
```\nts
// https://github.com/Hrdtr/nuxt-appwrite
import { defineNuxtModule, extendViteConfig } from '@nuxt/kit'

export default defineNuxtModule({
 setup() {
 extendViteConfig((config) => {
 config.optimizeDeps = config.optimizeDeps || {}
 config.optimizeDeps.include = config.optimizeDeps.include || []
 config.optimizeDeps.include.push('cross-fetch')
 })
 }
})
```\n
```

`addWebpackPlugin`

Append webpack plugin to the config.

Type

```
```\nts
function addWebpackPlugin (pluginOrGetter: PluginOrGetter, options?: ExtendWebpackConfigOptions): void

type PluginOrGetter = WebpackPluginInstance | WebpackPluginInstance[] | (() => WebpackPluginInstance | WebpackPluginInstance[])

interface ExtendWebpackConfigOptions {
 dev?: boolean
 build?: boolean
 server?: boolean
 client?: boolean
 prepend?: boolean
}
```\n
```

::tip
See [webpack website](https://webpack.js.org/concepts/plugins) for more information about webpack plugins. You can also use [this collection](https://webpack.js.org/awesome-webpack/#webpack-plugins) to find a plugin that suits your needs.
::

Parameters

`pluginOrGetter`

****Type**:** `PluginOrGetter`

****Required**:** `true`

A webpack plugin instance or an array of webpack plugin instances. If a function is provided, it must return a webpack plugin instance or an array of webpack plugin instances.

`options`

****Type**:** `ExtendWebpackConfigOptions`

****Default**:** `{}`

Options to pass to the callback function. This object can have the following properties:

- `dev` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building in development mode.

- `build` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building in production mode.

- `server` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building the server bundle.

- `client` (optional)

****Type**:** `boolean`

****Default**:** `true`

If set to `true`, the callback function will be called when building the client bundle.

- `prepend` (optional)

****Type**:** `boolean`

If set to `true`, the callback function will be prepended to the array with `unshift()` instead of `push()`.

Examples

```
````ts
// https://github.com/nuxt-modules/eslint
import EslintWebpackPlugin from 'eslint-webpack-plugin'
import { defineNuxtModule, addWebpackPlugin } from '@nuxt/kit'

export default defineNuxtModule({
 meta: {
 name: 'nuxt-eslint',
 configKey: 'eslint',
 },
 defaults: nuxt => ({
 include: [`${nuxt.options.srcDir}/**/*.{js,jsx,ts,tsx,vue}`],
 lintOnStart: true,
 }),
 setup(options, nuxt) {
 const webpackOptions = {
 ...options,
 context: nuxt.options.srcDir,
 files: options.include,
 lintDirtyModulesOnly: !options.lintOnStart
 }
 addWebpackPlugin(new EslintWebpackPlugin(webpackOptions), { server: false })
 }
})
````
```

`addVitePlugin`

Append Vite plugin to the config.

Type

```
````ts
function addVitePlugin (pluginOrGetter: PluginOrGetter, options?: ExtendViteConfigOptions): void

type PluginOrGetter = VitePlugin | VitePlugin[] | (() => VitePlugin | VitePlugin[])

interface ExtendViteConfigOptions {
 dev?: boolean
}
```

```
 build?: boolean
 server?: boolean
 client?: boolean
 prepend?: boolean
}
```

```
:::tip
See [Vite website](https://vitejs.dev/guide/api-plugin.html) for more information about Vite plugins. You can also use [this repository](https://github.com/vitejs/awesome-vite#plugins) to find a plugin that suits your needs.
::
```

### Parameters

```
`pluginOrGetter`
```

```
Type: `PluginOrGetter`
```

```
Required: `true`
```

A Vite plugin instance or an array of Vite plugin instances. If a function is provided, it must return a Vite plugin instance or an array of Vite plugin instances.

```
`options`
```

```
Type: `ExtendViteConfigOptions`
```

```
Default: `{}`
```

Options to pass to the callback function. This object can have the following properties:

- `dev` (optional)

```
Type: `boolean`
```

```
Default: `true`
```

If set to `true`, the callback function will be called when building in development mode.

- `build` (optional)

```
Type: `boolean`
```

```
Default: `true`
```

If set to `true`, the callback function will be called when building in production mode.

- `server` (optional)

```
Type: `boolean`
```

```
Default: `true`
```

If set to `true`, the callback function will be called when building the server bundle.

- `client` (optional)

```
Type: `boolean`
```

```
Default: `true`
```

If set to `true`, the callback function will be called when building the client bundle.

- `prepend` (optional)

```
Type: `boolean`
```

If set to `true`, the callback function will be prepended to the array with `unshift()` instead of `push()`.

### Examples

```
``ts
// https://github.com/yisibell/nuxt-svg-icons
import { defineNuxtModule, addVitePlugin } from '@nuxt/kit'
import { svg4VuePlugin } from 'vite-plugin-svg4vue'

export default defineNuxtModule({
 meta: {
 name: 'nuxt-svg-icons',
 configKey: 'nuxtSvgIcons',
 },
 defaults: {
 svg4vue: {
```

```

 assetsDirName: 'assets/icons',
 },
 },
 setup(options) {
 addVitePlugin(svg4VuePlugin(options.svg4vue))
 },
 })
 ...

```

## `addBuildPlugin`

Builder-agnostic version of `addWebpackPlugin` and `addVitePlugin`. It will add the plugin to both webpack and vite configurations if they are present.

### Type

```

````ts
function addBuildPlugin (pluginFactory: AddBuildPluginFactory, options?: ExtendConfigOptions): void

```

```

interface AddBuildPluginFactory {
  vite?: () => VitePlugin | VitePlugin[]
  webpack?: () => WebpackPluginInstance | WebpackPluginInstance[]
}

```

```

interface ExtendConfigOptions {
  dev?: boolean
  build?: boolean
  server?: boolean
  client?: boolean
  prepend?: boolean
}
````

```

### Parameters

#### `pluginFactory`

**\*\*Type\*\*:** `AddBuildPluginFactory`

**\*\*Required\*\*:** `true`

A factory function that returns an object with `vite` and/or `webpack` properties. These properties must be functions that return a Vite plugin instance or an array of Vite plugin instances and/or a webpack plugin instance or an array of webpack plugin instances.

#### `options`

**\*\*Type\*\*:** `ExtendConfigOptions`

**\*\*Default\*\*:** `{}`

Options to pass to the callback function. This object can have the following properties:

- `dev` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `true`

If set to `true`, the callback function will be called when building in development mode.

- `build` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `true`

If set to `true`, the callback function will be called when building in production mode.

- `server` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `true`

If set to `true`, the callback function will be called when building the server bundle.

- `client` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `true`

If set to `true`, the callback function will be called when building the client bundle.

- `prepend` (optional)

**\*\*Type\*\*:** `boolean`

If set to `true`, the callback function will be prepended to the array with `unshift()` instead of `push()`.

```

title: Vite
description: 'Activate Vite to your Nuxt 2 application with Nuxt Bridge.'

```

```
:::warning
When using `vite`, [nitro](/docs/bridge/nitro) must have been configured.
:::
```

## ## Remove Modules

- Remove `nuxt-vite`: Bridge enables same functionality

## ## Update Config

```
```:ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
  bridge: {
    vite: true,
    nitro: true
  }
})
```:
```

## ## Configuration

```
```:ts [nuxt.config.ts]
import { defineNuxtConfig } from '@nuxt/bridge'

export default defineNuxtConfig({
  vite: {
    // Config for Vite
  }
})
```:
```



```

title: Runtime Config
description: 'Learn how to migrate from Nuxt 2 to Nuxt 3 runtime config.'

```

If you wish to reference environment variables within your Nuxt 3 app, you will need to use runtime config.

When referencing these variables within your components, you will have to use the [`useRuntimeConfig`](/docs/api/composables/use-runtime-config) composable in your setup method (or Nuxt plugin).

In the `server/` portion of your app, you can use [`useRuntimeConfig`](/docs/api/composables/use-runtime-config) without any import.

```
:read-more{to="/docs/guide/going-further/runtime-config"}
```

## ## Migration

1. Add any environment variables that you use in your app to the `runtimeConfig` property of the `nuxt.config` file.
2. Migrate `process.env` to [`useRuntimeConfig`](/docs/api/composables/use-runtime-config) throughout the Vue part of your app.

```
::code-group
```

```
```ts [nuxt.config.ts]
export default defineNuxtConfig({
  runtimeConfig: {
    // Private config that is only available on the server
    apiSecret: '123',
    // Config within public will be also exposed to the client
    public: {
      apiBase: '/api'
    }
  },
})
```
```

```
```vue [pages/index.vue]
<script setup lang="ts">
const config = useRuntimeConfig()

// instead of process.env you will now access config.public.apiBase
console.log(config.public.apiBase)
</script>
```
```

```
```ts [server/api/hello.ts]
export default defineEventHandler((event) => {
  const config = useRuntimeConfig(event)
  // In server, you can now access config.apiSecret, in addition to config.public
  console.log(config.apiSecret)
  console.log(config.public.apiBase)
})
```
```

```
```ini [.env]
# Runtime config values are automatically replaced by matching environment variables at runtime
Nuxt_API_SECRET=api_secret_token
Nuxt_PUBLIC_API_BASE=https://nuxtjs.org
```
```

```
:::
```

```

title: Component Options
description: 'Learn how to migrate from Nuxt 2 components options to Nuxt 3 composables.'

```

## `asyncData` and `fetch`

Nuxt 3 provides new options for [fetching data from an API](/docs/getting-started/data-fetching).

```
<!-- TODO: Intro to <script setup> -->
<!-- TODO: Mention about options compatibility with asyncData -->
```

### Isomorphic Fetch

In Nuxt 2 you might use `@nuxtjs/axios` or `@nuxt/http` to fetch your data - or just the polyfilled global `fetch`.

In Nuxt 3 you can use a globally available `fetch` method that has the same API as [the Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API/Using\_Fetch) or [`\$fetch`](/docs/api/utils/dollarfetch) method which is using [unjs/ofetch](https://github.com/unjs/ofetch). It has a number of benefits, including:

1. It will handle 'smartly' making [direct API calls](/docs/guide/concepts/server-engine#direct-api-calls) if it's running on the server, or making a client-side call to your API if it's running on the client. (It can also handle calling third-party APIs.)

2. Plus, it comes with convenience features including automatically parsing responses and stringifying data.

You can read more [about direct API calls](/docs/guide/concepts/server-engine#direct-api-calls) or [fetching data](/docs/getting-started/data-fetching).

### Composables

Nuxt 3 provides new composables for fetching data: [`useAsyncData`](/docs/api/composables/use-async-data) and [`useFetch`]. They each have 'lazy' variants (`useLazyAsyncData` and `useLazyFetch`), which do not block client-side navigation.

In Nuxt 2, you'd fetch your data in your component using a syntax similar to:

```
``ts
export default {
 async asyncData({ params, $http }) {
 const post = await $http.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
 return { post }
 },
 // or alternatively
 fetch () {
 this.post = await $http.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
 }
}
``
```

Within your methods and templates, you could use the `post` variable similar how you'd use any other piece of data provided by your component.

With Nuxt 3, you can perform this data fetching using composables in your `setup()` method or `

## `key`

You can now define a key within the [`definePageMeta`](/docs/api/utils/define-page-meta) compiler macro.

```
```diff [pages/index.vue]
- <script>
- export default {
-   key: 'index'
-   // or a method
-   // key: route => route.fullPath
- }
+ <script setup>
+ definePageMeta({
+   key: 'index'
+   // or a method
+   // key: route => route.fullPath
+ })
</script>
```
```

## `layout`

`:read-more{to="/docs/migration/pages-and-layouts"}`

## `loading`

This feature is not yet supported in Nuxt 3.

## `middleware`

`:read-more{to="/docs/migration/plugins-and-middleware"}`

## `scrollToTop`

This feature is not yet supported in Nuxt 3. If you want to overwrite the default scroll behavior of `vue-router`, you can do so in `~/app/router.options.ts` (see [docs](/docs/guide/recipes/custom-routing#router-options)) for more info. Similar to `key`, specify it within the [`definePageMeta`](/docs/api/utils/define-page-meta) compiler macro.

```
```diff [pages/index.vue]
- <script>
- export default {
-   scrollToTop: false
- }
+ <script setup>
+ definePageMeta({
+   scrollToTop: false
+ })
</script>
```
```

## `transition`

`:read-more{to="/docs/getting-started/transitions"}`

## `validate`

The validate hook in Nuxt 3 only accepts a single argument, the `route`. Just as in Nuxt 2, you can return a boolean value. If you return false and another match can't be found, this will mean a 404. You can also directly return an object with `statusCode`/`statusMessage` to respond immediately with an error (other matches will not be checked).

```
```diff [pages/users/[id\].vue]
- <script>
- export default {
-   async validate({ params }) {
-     return /\d+/.test(params.id)
-   }
- }
+ <script setup>
+ definePageMeta({
+   validate: async (route) => {
+     const nuxtApp = useNuxtApp()
+     return /\d+/.test(route.params.id)
+   }
+ })
</script>
```
```

## `watchQuery`

This is not supported in Nuxt 3. Instead, you can directly use a watcher to trigger refetching data.

```
```vue [pages/users/[id\].vue]
<script setup lang="ts">
```

```
const route = useRoute()
const { data, refresh } = await useFetch('/api/user')
watch(() => route.query, () => refresh())
</script>
``
```

```

---
title: "Compatibility"
description: Nuxt Kit provides a set of utilities to help you check the compatibility of your modules with different Nuxt versions.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/compatibility.ts
    size: xs
---

```

Nuxt Kit utilities can be used in Nuxt 3, Nuxt 2 with Bridge and even Nuxt 2 without Bridge. To make sure your module is compatible with all versions, you can use the ``checkNuxtCompatibility``, ``assertNuxtCompatibility`` and ``hasNuxtCompatibility`` functions. They will check if the current Nuxt version meets the constraints you provide. Also you can use ``isNuxt2``, ``isNuxt3`` and ``getNuxtVersion`` functions for more granular checks.

``checkNuxtCompatibility``

Checks if constraints are met for the current Nuxt version. If not, returns an array of messages. Nuxt 2 version also checks for ``bridge`` support.

Type

```

````ts
async function checkNuxtCompatibility(
 constraints: NuxtCompatibility,
 nuxt?: Nuxt
): Promise<NuxtCompatibilityIssues>;

interface NuxtCompatibility {
 nuxt?: string;
 bridge?: boolean;
 builder?: {
 // Set `false` if your module is not compatible with a builder
 // or a semver-compatible string version constraint
 vite?: false | string;
 webpack?: false | string;
 };
}

interface NuxtCompatibilityIssue {
 name: string;
 message: string;
}

interface NuxtCompatibilityIssues extends Array<NuxtCompatibilityIssue> {
 toString(): string;
}
```

```

Parameters

``constraints``

****Type**:** ``NuxtCompatibility``

****Default**:** ``{}``

Constraints to check for. It accepts the following properties:

- ``nuxt`` (optional)

****Type**:** ``string``

Nuxt version in semver format. Versions may be defined in Node.js way, for example: ``>=2.15.0 <3.0.0``.

- ``bridge`` (optional)

****Type**:** ``boolean``

If set to ``true``, it will check if the current Nuxt version supports ``bridge``.

``nuxt``

****Type**:** ``Nuxt``

****Default**:** ``useNuxt()``

Nuxt instance. If not provided, it will be retrieved from the context via ``useNuxt()`` call.

``assertNuxtCompatibility``

Asserts that constraints are met for the current Nuxt version. If not, throws an error with the list of issues as string.

Type

```
````ts
async function assertNuxtCompatibility(
 constraints: NuxtCompatibility,
 nuxt?: Nuxt
): Promise<true>;

interface NuxtCompatibility {
 nuxt?: string;
 bridge?: boolean;
}
````
```

Parameters

`constraints`

Type: `NuxtCompatibility`

Default: `{}`

Constraints to check for. It accepts the following properties:

- `nuxt` (optional)

Type: `string`

Nuxt version in semver format. Versions may be defined in Node.js way, for example: `>=2.15.0 <3.0.0`.

- `bridge` (optional)

Type: `boolean`

If set to `true`, it will check if the current Nuxt version supports `bridge`.

`nuxt`

Type: `Nuxt`

Default: `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

`hasNuxtCompatibility`

Checks if constraints are met for the current Nuxt version. Return `true` if all constraints are met, otherwise returns `false`. Nuxt 2 version also checks for `bridge` support.

Type

```
````ts
async function hasNuxtCompatibility(
 constraints: NuxtCompatibility,
 nuxt?: Nuxt
): Promise<boolean>;

interface NuxtCompatibility {
 nuxt?: string;
 bridge?: boolean;
}
````
```

Parameters

`constraints`

Type: `NuxtCompatibility`

Default: `{}`

Constraints to check for. It accepts the following properties:

- `nuxt` (optional)

Type: `string`

Nuxt version in semver format. Versions may be defined in Node.js way, for example: `>=2.15.0 <3.0.0`.

- `bridge` (optional)

Type: `boolean`

If set to `true`, it will check if the current Nuxt version supports `bridge`.

`nuxt`

****Type**:** `Nuxt`

****Default**:** `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

`isNuxt2`

Checks if the current Nuxt version is 2.x.

Type

```
``ts
function isNuxt2(nuxt?: Nuxt): boolean;
``
```

Parameters

`nuxt`

****Type**:** `Nuxt`

****Default**:** `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

`isNuxt3`

Checks if the current Nuxt version is 3.x.

Type

```
``ts
function isNuxt3(nuxt?: Nuxt): boolean;
``
```

Parameters

`nuxt`

****Type**:** `Nuxt`

****Default**:** `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

`getNuxtVersion`

Returns the current Nuxt version.

Type

```
``ts
function getNuxtVersion(nuxt?: Nuxt): string;
``
```

Parameters

`nuxt`

****Type**:** `Nuxt`

****Default**:** `useNuxt()`

Nuxt instance. If not provided, it will be retrieved from the context via `useNuxt()` call.

```

---
title: "Auto-imports"
description: Nuxt Kit provides a set of utilities to help you work with auto-imports. These functions allow you to register
your own utils, composables and Vue APIs.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/imports.ts
    size: xs
---

# Auto-imports

Nuxt auto-imports helper functions, composables and Vue APIs to use across your application without explicitly importing
them. Based on the directory structure, every Nuxt application can also use auto-imports for its own composables and plugins.
With Nuxt Kit you can also add your own auto-imports. `addImports` and `addImportsDir` allow you to add imports to the Nuxt
application. `addImportsSources` allows you to add listed imports from 3rd party packages to the Nuxt application.

::note
These functions are designed for registering your own utils, composables and Vue APIs. For pages, components and plugins,
please refer to the specific sections: [Pages](/docs/api/kit/pages), [Components](/docs/api/kit/components), [Plugins]
(/docs/api/kit/plugins).
::

Nuxt auto-imports helper functions, composables and Vue APIs to use across your application without explicitly importing
them. Based on the directory structure, every Nuxt application can also use auto-imports for its own composables and plugins.
Composables or plugins can use these functions.

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/expanding-nuxt-s-auto-imports?friend=nuxt" target="_blank"}
Watch Vue School video about Auto-imports Nuxt Kit utilities.
::

## `addImports`

Add imports to the Nuxt application. It makes your imports available in the Nuxt application without the need to import them
manually.

### Type

```ts
function addImports (imports: Import | Import[]): void

interface Import {
 from: string
 priority?: number
 disabled?: boolean
 meta?: {
 description?: string
 docsUrl?: string
 [key: string]: any
 }
 type?: boolean
 typeFrom?: string
 name: string
 as?: string
}
```

### Parameters

#### `imports`

**Type**: `Import | Import[]`

**Required**: `true`

An object or an array of objects with the following properties:

- `from` (required)

  **Type**: `string`

  Module specifier to import from.

- `priority` (optional)

  **Type**: `number`

  **Default**: `1`

  Priority of the import, if multiple imports have the same name, the one with the highest priority will be used.

- `disabled` (optional)

```


****Type**:** `boolean`

If this import is disabled.

- `meta` (optional)

****Type**:** `object`

Metadata of the import.

- `meta.description` (optional)

****Type**:** `string`

Short description of the import.

- `meta.docsUrl` (optional)

****Type**:** `string`

URL to the documentation.

- `meta[key]` (optional)

****Type**:** `any`

Additional metadata.

- `type` (optional)

****Type**:** `boolean`

If this import is a pure type import.

- `typeFrom` (optional)

****Type**:** `string`

Using this as the from when generating type declarations.

- `name` (required)

****Type**:** `string`

Import name to be detected.

- `as` (optional)

****Type**:** `string`

Import as this name.

Examples

```
````ts
// https://github.com/pi0/storyblok-nuxt
import { defineNuxtModule, addImports, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
 setup(options, nuxt) {
 const names = [
 "useStoryblok",
 "useStoryblokApi",
 "useStoryblokBridge",
 "renderRichText",
 "RichTextSchema"
];

 names.forEach((name) =>
 addImports({ name, as: name, from: "@storyblok/vue" })
);
 }
})
````
```

`addImportsDir`

Add imports from a directory to the Nuxt application. It will automatically import all files from the directory and make them available in the Nuxt application without the need to import them manually.

Type

```
``ts
function addImportsDir (dirs: string | string[], options?: { prepend?: boolean }): void
``
```

Parameters

`dirs`

Type: `string | string[]`

Required: `true`

A string or an array of strings with the path to the directory to import from.

`options`

Type: `{ prepend?: boolean }`

Default: `{}`

Options to pass to the import. If `prepend` is set to `true`, the imports will be prepended to the list of imports.

Examples

```
``ts
// https://github.com/vueuse/motion/tree/main/src/nuxt
import { defineNuxtModule, addImportsDir, createResolver } from '@nuxt/kit'

export default defineNuxtModule({
  meta: {
    name: '@vueuse/motion',
    configKey: 'motion',
  },
  setup(options, nuxt) {
    const resolver = createResolver(import.meta.url)
    addImportsDir(resolver.resolve('./runtime/composables'))
  },
})
``
```

`addImportsSources`

Add listed imports to the Nuxt application.

Type

```
``ts
function addImportsSources (importSources: ImportSource | ImportSource[]): void

interface Import {
  from: string
  priority?: number
  disabled?: boolean
  meta?: {
    description?: string
    docsUrl?: string
    [key: string]: any
  }
  type?: boolean
  typeFrom?: string
  name: string
  as?: string
}

interface ImportSource extends Import {
  imports: (PresetImport | ImportSource)[]
}

type PresetImport = Omit<Import, 'from'> | string | [name: string, as?: string, from?: string]
``
```

Parameters

`importSources`

Type: `ImportSource | ImportSource[]`

Required: `true`

An object or an array of objects with the following properties:

- `imports` (required)

****Type**:** `PresetImport | ImportSource[]`

****Required**:** `true`

An object or an array of objects, which can be import names, import objects or import sources.

- `from` (required)

****Type**:** `string`

Module specifier to import from.

- `priority` (optional)

****Type**:** `number`

****Default**:** `1`

Priority of the import, if multiple imports have the same name, the one with the highest priority will be used.

- `disabled` (optional)

****Type**:** `boolean`

If this import is disabled.

- `meta` (optional)

****Type**:** `object`

Metadata of the import.

- `meta.description` (optional)

****Type**:** `string`

Short description of the import.

- `meta.docsUrl` (optional)

****Type**:** `string`

URL to the documentation.

- `meta[key]` (optional)

****Type**:** `any`

Additional metadata.

- `type` (optional)

****Type**:** `boolean`

If this import is a pure type import.

- `typeFrom` (optional)

****Type**:** `string`

Using this as the from when generating type declarations.

- `name` (required)

****Type**:** `string`

Import name to be detected.

- `as` (optional)

****Type**:** `string`

Import as this name.

Examples

```
``ts
// https://github.com/elk-zone/elk
import { defineNuxtModule, addImportsSources } from '@nuxt/kit'

export default defineNuxtModule({
  setup() {
    // add imports from h3 to make them autoimported
```

```
    addImportsSources({
      from: 'h3',
      imports: ['defineEventHandler', 'getQuery', 'getRouterParams', 'readBody', 'sendRedirect'] as Array<keyof typeof
import('h3')>,
    })
  }
})
...
```

```

---
title: "Components"
description: Nuxt Kit provides a set of utilities to help you work with components. You can register components globally or locally, and also add directories to be scanned for components.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/components.ts
    size: xs
---

```

Components are the building blocks of your Nuxt application. They are reusable Vue instances that can be used to create a user interface. In Nuxt, components from the components directory are automatically imported by default. However, if you need to import components from an alternative directory or wish to selectively import them as needed, `@nuxt/kit` provides the `addComponentsDir` and `addComponent` methods. These utils allow you to customize the component configuration to better suit your needs.

```

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/injecting-components-and-component-directories?friend=nuxt"
target="_blank"}
Watch Vue School video about injecting components.
::

```

`addComponentsDir`

Register a directory to be scanned for components and imported only when used. Keep in mind, that this does not register components globally, until you specify `global: true` option.

Type

```

````ts
async function addComponentsDir (dir: ComponentsDir, opts: { prepend?: boolean } = {}): void

interface ComponentsDir {
 path: string
 pattern?: string | string[]
 ignore?: string[]
 prefix?: string
 pathPrefix?: boolean
 enabled?: boolean
 prefetch?: boolean
 preload?: boolean
 isAsync?: boolean
 extendComponent?: (component: Component) => Promise<Component | void> | (Component | void)
 global?: boolean
 island?: boolean
 watch?: boolean
 extensions?: string[]
 transpile?: 'auto' | boolean
}

// You can augment this interface (exported from `@nuxt/schema`) if needed
interface ComponentMeta {
 [key: string]: unknown
}

interface Component {
 pascalName: string
 kebabName: string
 export: string
 filePath: string
 shortPath: string
 chunkName: string
 prefetch: boolean
 preload: boolean
 global?: boolean
 island?: boolean
 mode?: 'client' | 'server' | 'all'
 priority?: number
 meta?: ComponentMeta
}
...

```

#### ### Parameters

##### #### `dir`

**\*\*Type\*\*:** `ComponentsDir`

**\*\*Required\*\*:** `true`

An object with the following properties:

- `path` (required)

**\*\*Type\*\*:** `string`

Path (absolute or relative) to the directory containing your components.

You can use Nuxt aliases (~ or @) to refer to directories inside project or directly use an npm package path similar to require.

- `pattern` (optional)

**\*\*Type\*\*:** `string | string[]`

Accept Pattern that will be run against specified path.

- `ignore` (optional)

**\*\*Type\*\*:** `string[]`

Ignore patterns that will be run against specified path.

- `prefix` (optional)

**\*\*Type\*\*:** `string`

Prefix all matched components with this string.

- `pathPrefix` (optional)

**\*\*Type\*\*:** `boolean`

Prefix component name by its path.

- `enabled` (optional)

**\*\*Type\*\*:** `boolean`

Ignore scanning this directory if set to `true`.

- `prefetch` (optional)

**\*\*Type\*\*:** `boolean`

These properties (prefetch/preload) are used in production to configure how components with Lazy prefix are handled by webpack via its magic comments.

Learn more on [webpack documentation](https://webpack.js.org/api/module-methods/#magic-comments)

- `preload` (optional)

**\*\*Type\*\*:** `boolean`

These properties (prefetch/preload) are used in production to configure how components with Lazy prefix are handled by webpack via its magic comments.

Learn more on [webpack documentation](https://webpack.js.org/api/module-methods/#magic-comments)

- `isAsync` (optional)

**\*\*Type\*\*:** `boolean`

This flag indicates, component should be loaded async (with a separate chunk) regardless of using Lazy prefix or not.

- `extendComponent` (optional)

**\*\*Type\*\*:** `(component: Component) => Promise<Component | void> | (Component | void)`

A function that will be called for each component found in the directory. It accepts a component object and should return a component object or a promise that resolves to a component object.

- `global` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `false`

If enabled, registers components to be globally available.

- `island` (optional)

**\*\*Type\*\*:** `boolean`

If enabled, registers components as islands.

- `watch` (optional)

**\*\*Type\*\*:** `boolean`

Watch specified path for changes, including file additions and file deletions.

- `extensions` (optional)

**\*\*Type\*\*:** `string[]`

Extensions supported by Nuxt builder.

- `transpile` (optional)

**\*\*Type\*\*:** `'auto' | boolean`

Transpile specified path using `build.transpile`. If set to `'auto'`, it will set `transpile: true` if `node_modules/` is in path.

#### `opts`

**\*\*Required\*\*:** `false`

- `prepend` (optional)

**\*\*Type\*\*:** `boolean`

If set to `true`, the directory will be prepended to the array with `unshift()` instead of `push()`.

## `addComponent`

Register a component to be automatically imported.

### Type

```
````ts
async function addComponent (options: AddComponentOptions): void
```

```
interface AddComponentOptions {
  name: string,
  filePath: string,
  pascalName?: string,
  kebabName?: string,
  export?: string,
  shortPath?: string,
  chunkName?: string,
  prefetch?: boolean,
  preload?: boolean,
  global?: boolean,
  island?: boolean,
  mode?: 'client' | 'server' | 'all',
  priority?: number,
}
```
```

### Parameters

#### `options`

**\*\*Type\*\*:** `AddComponentOptions`

**\*\*Required\*\*:** `true`

An object with the following properties:

- `name` (required)

**\*\*Type\*\*:** `string`

Component name.

- `filePath` (required)

**\*\*Type\*\*:** `string`

Path to the component.

- `pascalName` (optional)

**\*\*Type\*\*:** `pascalCase(options.name)`

Pascal case component name. If not provided, it will be generated from the component name.

- `kebabName` (optional)

**\*\*Type\*\*:** `kebabCase(options.name)`

Kebab case component name. If not provided, it will be generated from the component name.

- `export` (optional)

**\*\*Type\*\*:** `string`

**\*\*Default\*\*:** `'default'`

Specify named or default export. If not provided, it will be set to `'default'`.

- `shortPath` (optional)

**\*\*Type\*\*:** `string`

Short path to the component. If not provided, it will be generated from the component path.

- `chunkName` (optional)

**\*\*Type\*\*:** `string`

**\*\*Default\*\*:** `'components/' + kebabCase(options.name)`

Chunk name for the component. If not provided, it will be generated from the component name.

- `prefetch` (optional)

**\*\*Type\*\*:** `boolean`

These properties (prefetch/preload) are used in production to configure how components with Lazy prefix are handled by webpack via its magic comments.

Learn more on [webpack documentation](https://webpack.js.org/api/module-methods/#magic-comments)

- `preload` (optional)

**\*\*Type\*\*:** `boolean`

These properties (prefetch/preload) are used in production to configure how components with Lazy prefix are handled by webpack via its magic comments.

Learn more on [webpack documentation](https://webpack.js.org/api/module-methods/#magic-comments)

- `global` (optional)

**\*\*Type\*\*:** `boolean`

**\*\*Default\*\*:** `false`

If enabled, registers component to be globally available.

- `island` (optional)

**\*\*Type\*\*:** `boolean`

If enabled, registers component as island. You can read more about islands in [`<NuxtIsland>`](/docs/api/components/nuxt-island#nuxtisland) component description.

- `mode` (optional)

**\*\*Type\*\*:** `'client' | 'server' | 'all'`

**\*\*Default\*\*:** `'all'`

This options indicates if component should render on client, server or both. By default, it will render on both client and server.

- `priority` (optional)

**\*\*Type\*\*:** `number`

**\*\*Default\*\*:** `1`

Priority of the component, if multiple components have the same name, the one with the highest priority will be used.



```

navigation: false

```

## # Nuxt Docs

This repository contains the documentation of Nuxt hosted on [<https://nuxt.com/docs>](https://nuxt.com/docs)

## ## Contributing

Have a look at [<https://github.com/nuxt/nuxt.com>](https://github.com/nuxt/nuxt.com) to run the website locally.

```

title: "Context"
description: Nuxt Kit provides a set of utilities to help you work with context.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/context.ts
 size: xs

```

Nuxt modules allow you to enhance Nuxt's capabilities. They offer a structured way to keep your code organized and modular. If you're looking to break down your module into smaller components, Nuxt offers the `useNuxt` and `tryUseNuxt` functions. These functions enable you to conveniently access the Nuxt instance from the context without having to pass it as argument.

```

::note
When you're working with the setup function in Nuxt modules, Nuxt is already provided as the second argument. This means you can directly utilize it without needing to call useNuxt(). You can look at [Nuxt Site Config] (https://github.com/harlan-zw/nuxt-site-config) as an example of usage.
::

```

## ## `useNuxt`

Get the Nuxt instance from the context. It will throw an error if Nuxt is not available.

### ### Type

```

```ts
function useNuxt(): Nuxt

interface Nuxt {
  options: NuxtOptions
  hooks: Hookable<NuxtHooks>
  hook: Nuxt['hooks']['hook']
  callHook: Nuxt['hooks']['callHook']
  addHooks: Nuxt['hooks']['addHooks']
  ready: () => Promise<void>
  close: () => Promise<void>
  server?: any
  vfs: Record<string, string>
  apps: Record<string, NuxtApp>
}
```

```

### ### Examples

#### ::code-group

```

```ts [setupTranspilation.ts]
// https://github.com/Lexpeartha/nuxt-xstate/blob/main/src/parts/transpile.ts
import { useNuxt } from '@nuxt/kit'

export const setupTranspilation = () => {
  const nuxt = useNuxt()

  nuxt.options.build.transpile = nuxt.options.build.transpile || []

  if (nuxt.options.builder === '@nuxt/webpack-builder') {
    nuxt.options.build.transpile.push(
      'xstate',
    )
  }
}
```

```

```

```ts [module.ts]
import { useNuxt } from '@nuxt/kit'
import { setupTranspilation } from './setupTranspilation'

export default defineNuxtModule({
  setup() {
    setupTranspilation()
  }
})
```

```

```

::

```

## ## `tryUseNuxt`

Get the Nuxt instance from the context. It will return `null` if Nuxt is not available.

### ### Type

```

``ts
function tryUseNuxt(): Nuxt | null

interface Nuxt {
 options: NuxtOptions
 hooks: Hookable<NuxtHooks>
 hook: Nuxt['hooks']['hook']
 callHook: Nuxt['hooks']['callHook']
 addHooks: Nuxt['hooks']['addHooks']
 ready: () => Promise<void>
 close: () => Promise<void>
 server?: any
 vfs: Record<string, string>
 apps: Record<string, NuxtApp>
}
...

Examples

::code-group

``ts [requireSiteConfig.ts]
// https://github.com/harlan-zw/nuxt-site-config/blob/main/test/assertions.test.ts
import { tryUseNuxt } from '@nuxt/kit'

interface SiteConfig {
 title: string
}

export const requireSiteConfig = (): SiteConfig => {
 const nuxt = tryUseNuxt()
 if (!nuxt) {
 return { title: null }
 }
 return nuxt.options.siteConfig
}
...

``ts [module.ts]
import { useNuxt } from '@nuxt/kit'
import { requireSiteConfig } from './requireSiteConfig'

export default defineNuxtModule({
 setup(_, nuxt) {
 const config = requireSiteConfig()
 nuxt.options.app.head.title = config.title
 }
})
...

::

```

```

title: Pages
description: Nuxt Kit provides a set of utilities to help you create and use pages. You can use these utilities to manipulate the pages configuration or to define route rules.
links:
 - label: Source
 icon: i-simple-icons-github
 to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/pages.ts
 size: xs

```

## ## `extendPages`

In Nuxt 3, routes are automatically generated based on the structure of the files in the `pages` directory. However, there may be scenarios where you'd want to customize these routes. For instance, you might need to add a route for a dynamic page not generated by Nuxt, remove an existing route, or modify the configuration of a route. For such customizations, Nuxt offers the `extendPages` feature, which allows you to extend and alter the pages configuration.

**::tip**{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/extend-and-alter-nuxt-pages?friend=nuxt" target="\_blank"} Watch Vue School video about extendPages.  
::

### ### Type

```

````ts
function extendPages (callback: (pages: NuxtPage[]) => void): void

```

```

type NuxtPage = {
  name?: string
  path: string
  file?: string
  meta?: Record<string, any>
  alias?: string[] | string
  redirect?: RouteLocationRaw
  children?: NuxtPage[]
}
````

```

### ### Parameters

#### #### `callback`

**\*\*Type\*\*:** `(pages: NuxtPage[]) => void`

**\*\*Required\*\*:** `true`

A function that will be called with the pages configuration. You can alter this array by adding, deleting, or modifying its elements. Note: You should modify the provided pages array directly, as changes made to a copied array will not be reflected in the configuration.

### ### Examples

```

````ts
// https://github.com/nuxt-modules/prismic/blob/master/src/module.ts
import { createResolver, defineNuxtModule, extendPages } from '@nuxt/kit'

export default defineNuxtModule({
  setup(options) {
    const resolver = createResolver(import.meta.url)

    extendPages((pages) => {
      pages.unshift({
        name: 'prismic-preview',
        path: '/preview',
        file: resolver.resolve('runtime/preview.vue')
      })
    })
  }
})
````

```

## ## `extendRouteRules`

Nuxt is powered by the [Nitro](https://nitro.unjs.io) server engine. With Nitro, you can incorporate high-level logic directly into your configuration, which is useful for actions like redirects, proxying, caching, and appending headers to routes. This configuration works by associating route patterns with specific route settings.

**::tip**  
You can read more about Nitro route rules in the [Nitro documentation](https://nitro.unjs.io/guide/routing#route-rules).  
::

**::tip**{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/adding-route-rules-and-route-middlewares?friend=nuxt" target="\_blank"}

Watch Vue School video about adding route rules and route middlewares.

::

### ### Type

```
``ts
function extendRouteRules (route: string, rule: NitroRouteConfig, options: ExtendRouteRulesOptions): void

interface NitroRouteConfig {
 cache?: CacheOptions | false;
 headers?: Record<string, string>;
 redirect?: string | { to: string; statusCode?: HTTPStatusCode };
 prerender?: boolean;
 proxy?: string | ({ to: string } & ProxyOptions);
 isr?: number | boolean;
 cors?: boolean;
 swr?: boolean | number;
 static?: boolean | number;
}

interface ExtendRouteRulesOptions {
 override?: boolean
}

interface CacheOptions {
 swr?: boolean
 name?: string
 group?: string
 integrity?: any
 maxAge?: number
 staleMaxAge?: number
 base?: string
 headersOnly?: boolean
}

// See https://www.jsdocs.io/package/h3#ProxyOptions
interface ProxyOptions {
 headers?: RequestHeaders | HeadersInit;
 fetchOptions?: RequestInit & { duplex?: Duplex } & {
 ignoreResponseError?: boolean;
 };
 fetch?: typeof fetch;
 sendStream?: boolean;
 streamRequest?: boolean;
 cookieDomainRewrite?: string | Record<string, string>;
 cookiePathRewrite?: string | Record<string, string>;
 onResponse?: (event: H3Event, response: Response) => void;
}
``
```

### ### Parameters

#### `route`

**Type:** `string`

**Required:** `true`

A route pattern to match against.

#### `rule`

**Type:** `NitroRouteConfig`

**Required:** `true`

A route configuration to apply to the matched route.

#### `options`

**Type:** `ExtendRouteRulesOptions`

**Default:** `{}`

Options to pass to the route configuration. If `override` is set to `true`, it will override the existing route configuration.

### ### Examples

```
``ts
// https://github.com/directus/website/blob/main/modules/redirects.ts
import { createResolver, defineNuxtModule, extendRouteRules, extendPages } from '@nuxt/kit'
```

```
export default defineNuxtModule({
 setup(options) {
 const resolver = createResolver(import.meta.url)

 extendPages((pages) => {
 pages.unshift({
 name: 'preview-new',
 path: '/preview-new',
 file: resolver.resolve('runtime/preview.vue')
 })
 })

 extendRouteRules('/preview', {
 redirect: {
 to: '/preview-new',
 statusCode: 302
 }
 })

 extendRouteRules('/preview-new', {
 cache: {
 maxAge: 60 * 60 * 24 * 7
 }
 })
 }
})

```

### ## `addRouteMiddleware`

Registers route middlewares to be available for all routes or for specific routes.

Route middlewares can be also defined in plugins via [``addRouteMiddleware``](/docs/api/utils/add-route-middleware) composable.

**::tip**  
Read more about route middlewares in the [Route middleware documentation](/docs/getting-started/routing#route-middleware).  
**::**

**::tip**{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/adding-route-rules-and-route-middlewares?friend=nuxt" target="\_blank"}  
Watch Vue School video about adding route rules and route middlewares.  
**::**

### ### Type

```
````ts
function addRouteMiddleware (input: NuxtMiddleware | NuxtMiddleware[], options: AddRouteMiddlewareOptions): void

type NuxtMiddleware = {
  name: string
  path: string
  global?: boolean
}

interface AddRouteMiddlewareOptions {
  override?: boolean
  prepend?: boolean
}

```

Parameters

`input`

****Type**:** ``NuxtMiddleware | NuxtMiddleware[]``

****Required**:** ``true``

A middleware object or an array of middleware objects with the following properties:

- ``name`` (required)

****Type**:** ``string``

Middleware name.

- ``path`` (required)

****Type**:** ``string``

Path to the middleware.

- ``global`` (optional)

****Type**:** `boolean`

If enabled, registers middleware to be available for all routes.

`options`

****Type**:** `AddRouteMiddlewareOptions`

****Default**:** `{}`

- `override` (optional)

****Type**:** `boolean`

****Default**:** `false`

If enabled, overrides the existing middleware with the same name.

- `prepend` (optional)

****Type**:** `boolean`

****Default**:** `false`

If enabled, prepends the middleware to the list of existing middleware.

Examples

::code-group

```
``ts [runtime/auth.ts]
export default defineNuxtRouteMiddleware((to, from) => {
  // isAuthenticated() is an example method verifying if a user is authenticated
  if (to.path !== '/login' && isAuthenticated() === false) {
    return navigateTo('/login')
  }
})
``
```

```
``ts [module.ts]
import { createResolver, defineNuxtModule, addRouteMiddleware } from '@nuxt/kit'

export default defineNuxtModule({
  setup() {
    const resolver = createResolver(import.meta.url)

    addRouteMiddleware({
      name: 'auth',
      path: resolver.resolve('runtime/auth.ts'),
      global: true
    }, { prepend: true })
  }
})
``
```

::

```

---
title: "Layout"
description: "Nuxt Kit provides a set of utilities to help you work with layouts."
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/layout.ts
    size: xs
---

```

Layouts is used to be a wrapper around your pages. It can be used to wrap your pages with common components, for example, a header and a footer. Layouts can be registered using `addLayout` utility.

`addLayout`

Register template as layout and add it to the layouts.

```

::note
In Nuxt 2 `error` layout can also be registered using this utility. In Nuxt 3+ `error` layout [replaced](/docs/getting-started/error-handling#rendering-an-error-page) with `error.vue` page in project root.
::

```

Type

```

``ts
function addLayout (layout: NuxtTemplate | string, name: string): void

interface NuxtTemplate {
  src?: string
  filename?: string
  dst?: string
  options?: Record<string, any>
  getContents?: (data: Record<string, any>) => string | Promise<string>
  write?: boolean
}
``

```

Parameters

`layout`

****Type**:** `NuxtTemplate | string`

****Required**:** `true`

A template object or a string with the path to the template. If a string is provided, it will be converted to a template object with `src` set to the string value. If a template object is provided, it must have the following properties:

- `src` (optional)

****Type**:** `string`

Path to the template. If `src` is not provided, `getContents` must be provided instead.

- `filename` (optional)

****Type**:** `string`

Filename of the template. If `filename` is not provided, it will be generated from the `src` path. In this case, the `src` option is required.

- `dst` (optional)

****Type**:** `string`

Path to the destination file. If `dst` is not provided, it will be generated from the `filename` path and nuxt `buildDir` option.

- `options` (optional)

****Type**:** `Options`

Options to pass to the template.

- `getContents` (optional)

****Type**:** `(data: Options) => string | Promise<string>`

A function that will be called with the `options` object. It should return a string or a promise that resolves to a string. If `src` is provided, this function will be ignored.

- `write` (optional)

****Type**:** `boolean`

If set to `true`, the template will be written to the destination file. Otherwise, the template will be used only in virtual filesystem.

```

---
title: Plugins
description: Nuxt Kit provides a set of utilities to help you create and use plugins. You can add plugins or plugin templates to your module using these functions.
links:
  - label: Source
    icon: i-simple-icons-github
    to: https://github.com/nuxt/nuxt/blob/main/packages/kit/src/plugin.ts
    size: xs
---

```

Plugins are self-contained code that usually add app-level functionality to Vue. In Nuxt, plugins are automatically imported from the `plugins` directory. However, if you need to ship a plugin with your module, Nuxt Kit provides the `addPlugin` and `addPluginTemplate` methods. These utils allow you to customize the plugin configuration to better suit your needs.

`addPlugin`

Registers a Nuxt plugin and to the plugins array.

```

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/injecting-plugins?friend=nuxt" target="_blank"}
Watch Vue School video about addPlugin.
::

```

Type

```

````ts
function addPlugin (plugin: NuxtPlugin | string, options: AddPluginOptions): NuxtPlugin

```

```

interface NuxtPlugin {
 src: string
 mode?: 'all' | 'server' | 'client'
 order?: number
}

```

```

interface AddPluginOptions { append?: boolean }
```

```

Parameters

`plugin`

Type: `NuxtPlugin | string`

Required: `true`

A plugin object or a string with the path to the plugin. If a string is provided, it will be converted to a plugin object with `src` set to the string value. If a plugin object is provided, it must have the following properties:

- `src` (required)

Type: `string`

Path to the plugin.

- `mode` (optional)

Type: `'all' | 'server' | 'client'`

Default: `'all'`

If set to `'all'`, the plugin will be included in both client and server bundles. If set to `'server'`, the plugin will only be included in the server bundle. If set to `'client'`, the plugin will only be included in the client bundle. You can also use `.client` and `.server` modifiers when specifying `src` option to use plugin only in client or server side.

- `order` (optional)

Type: `number`

Default: `0`

Order of the plugin. This allows more granular control over plugin order and should only be used by advanced users. Lower numbers run first, and user plugins default to `0`. It's recommended to set `order` to a number between `-20` for `pre`-plugins (plugins that run before Nuxt plugins) and `20` for `post`-plugins (plugins that run after Nuxt plugins).

```

::warning
Don't use `order` unless you know what you're doing. For most plugins, the default `order` of `0` is sufficient. To append a plugin to the end of the plugins array, use the `append` option instead.
::

```

`options`

Type: `AddPluginOptions`

****Default**:** `{}`

Options to pass to the plugin. If `append` is set to `true`, the plugin will be appended to the plugins array instead of prepended.

Examples

::code-group

```
``ts [module.ts]
import { createResolver, defineNuxtModule, addPlugin } from '@nuxt/kit'
```

```
export default defineNuxtModule({
  setup() {
    const resolver = createResolver(import.meta.url)

    addPlugin({
      src: resolver.resolve('runtime/plugin.js'),
      mode: 'client'
    })
  }
})
``
```

```
``ts [runtime/plugin.js]
// https://github.com/nuxt/nuxters
export default defineNuxtPlugin((nuxtApp) => {
  const colorMode = useColorMode()

  nuxtApp.hook('app:mounted', () => {
    if (colorMode.preference !== 'dark') {
      colorMode.preference = 'dark'
    }
  })
})
``
```

::

`addPluginTemplate`

Adds a template and registers as a nuxt plugin. This is useful for plugins that need to generate code at build time.

::tip{icon="i-ph-video-duotone" to="https://vueschool.io/lessons/injecting-plugin-templates?friend=nuxt" target="_blank"} Watch Vue School video about addPluginTemplate.

Type

```
``ts
function addPluginTemplate (pluginOptions: NuxtPluginTemplate, options: AddPluginOptions): NuxtPlugin
```

```
interface NuxtPluginTemplate<Options = Record<string, any>> {
  src?: string,
  filename?: string,
  dst?: string,
  mode?: 'all' | 'server' | 'client',
  options?: Options,
  getContents?: (data: Options) => string | Promise<string>,
  write?: boolean,
  order?: number
}
```

```
interface AddPluginOptions { append?: boolean }
```

```
interface NuxtPlugin {
  src: string
  mode?: 'all' | 'server' | 'client'
  order?: number
}
``
```

Parameters

`pluginOptions`

****Type**:** `NuxtPluginTemplate`

****Required**:** `true`

A plugin template object with the following properties:

- `src` (optional)

****Type**:** `string`

Path to the template. If `src` is not provided, `getContents` must be provided instead.

- `filename` (optional)

****Type**:** `string`

Filename of the template. If `filename` is not provided, it will be generated from the `src` path. In this case, the `src` option is required.

- `dst` (optional)

****Type**:** `string`

Path to the destination file. If `dst` is not provided, it will be generated from the `filename` path and next `buildDir` option.

- `mode` (optional)

****Type**:** `'all' | 'server' | 'client'`

****Default**:** `'all'`

If set to `'all'`, the plugin will be included in both client and server bundles. If set to `'server'`, the plugin will only be included in the server bundle. If set to `'client'`, the plugin will only be included in the client bundle. You can also use `.client` and `.server` modifiers when specifying `src` option to use plugin only in client or server side.

- `options` (optional)

****Type**:** `Options`

Options to pass to the template.

- `getContents` (optional)

****Type**:** `(data: Options) => string | Promise<string>`

A function that will be called with the `options` object. It should return a string or a promise that resolves to a string. If `src` is provided, this function will be ignored.

- `write` (optional)

****Type**:** `boolean`

If set to `true`, the template will be written to the destination file. Otherwise, the template will be used only in virtual filesystem.

- `order` (optional)

****Type**:** `number`

****Default**:** `0`

Order of the plugin. This allows more granular control over plugin order and should only be used by advanced users. Lower numbers run first, and user plugins default to `0`. It's recommended to set `order` to a number between `-20` for `pre`-plugins (plugins that run before Nuxt plugins) and `20` for `post`-plugins (plugins that run after Nuxt plugins).

:::warning

Don't use `order` unless you know what you're doing. For most plugins, the default `order` of `0` is sufficient. To append a plugin to the end of the plugins array, use the `append` option instead.

:::

`options`

****Type**:** `AddPluginOptions`

****Default**:** `{}`

Options to pass to the plugin. If `append` is set to `true`, the plugin will be appended to the plugins array instead of prepended.

Examples

:::code-group

```
```ts [module.ts]
// https://github.com/vuejs/vuefire
import { createResolver, defineNuxtModule, addPluginTemplate } from '@nuxt/kit'

export default defineNuxtModule({
 setup() {
```

```

const resolver = createResolver(import.meta.url)

addPluginTemplate({
 src: resolve(templatesDir, 'plugin.ejs'),
 filename: 'plugin.mjs',
 options: {
 ...options,
 ssr: nuxt.options.ssr,
 },
})
}
})
...

``ts [runtime/plugin.ejs]
import { VueFire, useSSRInitialState } from 'vuefire'
import { defineNuxtPlugin } from '#imports'

export default defineNuxtPlugin((nuxtApp) => {
 const firebaseApp = nuxtApp.$firebaseApp

 nuxtApp.vueApp.use(VueFire, { firebaseApp })

 <% if(options.ssr) { %>
 if (import.meta.server) {
 nuxtApp.payload.vuefire = useSSRInitialState(undefined, firebaseApp)
 } else if (nuxtApp.payload?.vuefire) {
 useSSRInitialState(nuxtApp.payload.vuefire, firebaseApp)
 }
 <% } %>
})
...

::

```