Error handling

Learn how to catch errors in different lifecycle.

Handling Errors

Nuxt 3 is a full-stack framework, which means there are several sources of unpreventable user runtime errors that can happen in different contexts:

1. Errors during the Vue rendering lifecycle (SSR + SPA)
2. Errors during API or Nitro server lifecycle
3. Server and client startup errors (SSR + SPA)
4. Errors downloading JS chunks

Errors During the Vue Rendering Lifecycle (SSR + SPA)

You can hook into Vue errors using `onErrorCaptured`.

In addition, Nuxt provides a `vue:error` 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`. It will receive all Vue errors, even if they are handled.

Example With Global Error Reporting Framework

```
export default defineNuxtPlugin((nuxtApp) => {
  nuxtApp.vueApp.config.errorHandler = (error, context) => {
    // ...
  }
})
```

Server and Client Startup Errors (SSR + SPA)

Nuxt will call the `app:error` hook if there are any errors in starting your Nuxt application.

This includes:

- running Nuxt plugins
- processing `app:created` and `app:beforeMount` hooks
- rendering your Vue app to HTML (on the server)
- mounting the app (on client-side), though you should handle this case with `onErrorCaptured` or with `vue:error`
- processing the `app:mounted` hook

Errors During API or Nitro Server Lifecycle

You cannot currently define a server-side handler for these errors, but can render an error page (see the next section).

Errors downloading 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 for ideas.

Rendering an Error Page

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.

This 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

An error 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`).

The lifecycle hooks are listed here.

You can customize this error page by adding `~/error.vue` in the source directory of your application, alongside `app.vue` . 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.

The error page has a single prop - `error` which contains an error for you to handle.

The `error` object provides the fields: `url` , `statusCode` , `statusMessage` , `message` , `description` and `data` . If you have an error with custom fields they will be lost; you should assign them to `data` instead. 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.

```
export default defineNuxtPlugin(nuxtApp => {
  nuxtApp.hook('vue:error', (err) => {
    //
  })
})
```

When you are ready to remove the error page, you can call the `clearError` helper function, which takes an optional path to redirect to (for example, if you want to navigate to a 'safe' page).

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.

If you are running on Node 16 and you set any cookies when rendering your error page, they will overwrite cookies previously set. We recommend using a newer version of Node as Node 16 will reach end-of-life in September 2023.

Example

error.vue

```vue
<script setup lang="ts">
const props = defineProps({
  error: Object
})

const handleError = () => clearError({ redirect: '/' })
</script>

<template>
  <button @click="handleError">Clear errors</button>
</template>
```

Error Helper Methods

useError

- `function useError (): Ref<Error | { url, statusCode, statusMessage, message, description, data }>`

This function will return the global Nuxt error that is being handled.

👉 Read more in Docs > API > Composables > Use Error.

createError

- `function createError (err: { cause, data, message, name, stack, statusCode, statusMessage, fatal }): Error`

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.

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

pages/movies/[slug].vue

```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>
```

showError

- `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`.

It is recommended instead to use `throw createError()`.

👉 Read more in Docs > API > Utils > Show Error.

`clearError`

- `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 in Docs > API > Utils > Clear Error.

Rendering Errors Within Your App

Nuxt also provides a `<NuxtErrorBoundary>` 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.)

> If you navigate to another route, the error will be cleared automatically.

Example

pages/index.vue

```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 }}
      <button @click="clearError">
        This will clear the error.
```


Read and edit a live example in Docs > Examples > Advanced > Error Handling.

Edit on Github