Middleware Directory

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.

> Route middleware run 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/` **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** (with a `.global` suffix) and will be automatically run on every route change.

The first two kinds of route middleware can be **defined in** `definePageMeta`.

Format

Route middleware are navigation guards that receive the current route and the next route as arguments.

```
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 (to: RouteLocationRaw | undefined | null, options?: { replace: boolean, redirectCode: number, external: boolean } )` - Redirects to the given route, within plugins or middleware. It can also be called directly to perform page navigation.
2. `abortNavigation (err?: string | Error)` - Aborts the navigation, with an optional error message.

Unlike navigation guards in the vue-router docs, a third `next()` argument is not passed, and redirects or route cancellation is handled by returning a value from the middleware. Possible return values are:

- nothing - does not block navigation and will move to the next middleware function, if any, or complete the route navigation
- `return navigateTo('/')` or `return navigateTo({ path: '/' })` - redirects to the given path and will set the redirect code to `302` Found 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 if the redirect happens on the server side

👉 Read more in Docs > API > Utils > Navigate To.

- `return abortNavigation()` - stops the current navigation
- `return abortNavigation(error)` - rejects the current navigation with an error

👉 Read more in Docs > API > Utils > Abort Navigation.

We recommend using the helper functions above for performing redirects or stopping navigation. Other possible return values described in the vue-router docs may work but there may be breaking changes in future.

What Order Middleware Runs In

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:

middleware/ directory

```
middleware/
--| analytics.global.ts
--| setup.global.ts
--| auth.ts
```

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.

middleware/ directory

```
middleware/
```

```
--| 01.setup.global.ts
--| 02.analytics.global.ts
--| auth.ts
```

> 💡 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:

```
export default defineNuxtRouteMiddleware(to => {
  // skip middleware on server
  if (process.server) return
  // skip middleware on client side entirely
  if (process.client) return
  // or only skip middleware on initial client load
  const nuxtApp = useNuxtApp()
  if (process.client && nuxtApp.isHydrating && nuxtApp.payload.serverRendered) return
})
```

Adding Middleware Dynamically

It is possible to add global or named route middleware manually using the `addRouteMiddleware()` helper function, such as from within a plugin.

```
export default defineNuxtPlugin(() => {
  addRouteMiddleware('global-test', () => {
    console.log('this global middleware was added in a plugin and will be run on every route chang
  }, { global: true })
```

```
  addRouteMiddleware('named-test', () => {
    console.log('this named middleware was added in a plugin and would override any existing middl
  })
})
```

Example: A Named Route Middleware

```
> -| middleware/
> ---| auth.ts
```

In your page file, you can reference this route middleware

```
<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.

Read and edit a live example in Docs > Examples > Routing > Middleware.

Edit on Github