# Pages Directory

Nuxt provides a file-based routing to create routes within your web application using Vue Router under the hood.

This directory is **optional**, meaning that `vue-router` won't be included if you only use app.vue (unless you set `pages: true` in `nuxt.config` or have a `app/router.options.ts`), reducing your application's bundle size.

## Usage

Pages are Vue components and can have any valid extension 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.

**pages/index.vue** pages/index.ts pages/index.tsx

pages/index.vue

```vue
<template>
  <h1>Index page</h1>
</template>
```

The `pages/index.vue` file will be mapped to the `/` route of your application.

If you are using app.vue, make sure to use the `<NuxtPage/>` component to display the current page:

app.vue

```vue
<template>
  <div>
    <!-- Markup shared across all pages, ex: NavBar -->
    <NuxtPage />
```

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

Pages **must have a single root element** to allow route 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:

**pages/working.vue** pages/bad-1.vue pages/bad-2.vue

pages/working.vue

```
<template>
  <div>
    <!-- This page correctly has only one single root element -->
    Page content
  </div>
</template>
```

# Dynamic Routes

If you place anything within square brackets, it will be turned into a dynamic route 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`.

## Example

```
> -| pages/
> ---| index.vue
> ---| users-[group]/
> -----| [id].vue
```

Given the example above, you can access group/id within your component via the `$route` object:

pages/users-[group]/[id].vue

```
<template>
  <p>{{ $route.params.group }} - {{ $route.params.id }}</p>
</template>
```

Navigating to `/users-admins/123` would render:

```
<p>admins - 123</p>
```

If you want to access the route using Composition API, there is a global `useRoute` function that will allow you to access the route just like `this.$route` in the Options API.

```
<script setup lang="ts">
const route = useRoute()

if (route.params.group === 'admins' && !route.params.id) {
  console.log('Warning! Make sure user is authenticated!')
}
</script>
```

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`. 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.

pages/[...slug].vue

```
<template>
  <p>{{ $route.params.slug }}</p>
</template>
```

Navigating to `/hello/world` would render:

```
<p>["hello", "world"]</p>
```

# Nested Routes

It is possible to display nested routes with `<NuxtPage>` .

Example:

```
> -| pages/
> ---| parent/
> ------| child.vue
> ---| parent.vue
```

This file tree will generate these routes:

```
[
  {
    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 `<NuxtPage>` component inside `pages/parent.vue` :

pages/parent.vue

```
<template>
  <div>
    <h1>I am the parent view</h1>
    <NuxtPage :foobar="123" />
  </div>
</template>
```

# Child Route Keys

If you want more control over when the `<NuxtPage>` 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`:

pages/parent.vue

```vue
<template>
  <div>
    <h1>I am the parent view</h1>
    <NuxtPage :page-key="route => route.fullPath" />
  </div>
</template>
```

Or alternatively:

pages/child.vue

```vue
<script setup lang="ts">
definePageMeta({
  key: route => route.fullPath
})
</script>
```

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

# 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 `<script>` and in `<script setup>`:

```vue
<script setup lang="ts">
definePageMeta({
  title: 'My home page'
})
</script>
```

This data can then be accessed throughout the rest of your app from the `route.meta` object.

```vue
<script setup lang="ts">
const route = useRoute()

console.log(route.meta.title) // My home page
```

```
</script>
```

If you are using nested routes, the page metadata from all these routes will be merged into a single object. For more on route meta, see the vue-router docs.

Much like `defineEmits` or `defineProps` (see Vue docs), `definePageMeta` is a **compiler macro**. It will be compiled away so you cannot reference it within your component. Instead, the metadata passed to it will be hoisted out of the component. Therefore, the page meta object cannot reference the component (or values defined on the component). However, it can reference imported bindings.

```
<script setup lang="ts">
import { someData } from '~/utils/example'

const title = ref('')

definePageMeta({
  title,  // This will create an error
  someData
})
</script>
```

# Special Metadata

Of course, you are welcome to define metadata for your own use throughout your app. But some metadata defined with `definePageMeta` has a particular purpose:

### `alias`

You can define page aliases. They allow you to access the same page from different paths. It can be either a string or an array of strings as defined here on vue-router documentation.

### `keepalive`

Nuxt will automatically wrap your page in the Vue `<KeepAlive>` component if you set `keepalive: true` in your `definePageMeta`. This might be useful to do, for example, in a parent route that has dynamic child routes, if you want to preserve page state across route changes.

When your goal is to preserve state for parent routes use this syntax: `<NuxtPage keepalive />`. You can also set props to be passed to `<KeepAlive>` (see a full list here).

You can set a default value for this property in your `nuxt.config`.

### `key`

See above.

### `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.

### `layoutTransition` and `pageTransition`

You can define transition properties for the `<transition>` component that wraps your pages and layouts, or pass `false` to disable the `<transition>` wrapper for that route. You can see a list of options that can be passed here or read more about how transitions work.

You can set default values for these properties in your `nuxt.config`.

### `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), or an array of strings/functions. More about named 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 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`:

index.d.ts

```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>` 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>
```

Learn more about `<NuxtLink>` usage.

# Programmatic Navigation

Nuxt 3 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.

```
<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>
```

# 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 in Docs > Guide > Going Further > 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 Layers to create groupings of your app's pages.

Example:

```
> -| nuxt.config.ts
> -| some-app/
```

```
> ---| nuxt.config.ts
> ---| pages
> -----| app-page.vue
```

```ts
// some-app/nuxt.config.ts
export default defineNuxtConfig({
})

// nuxt.config.ts
export default defineNuxtConfig({
  extends: ['./some-app'],
})
```

👉 Read more in Docs > Guide > Going Further > Layers.

Edit on Github