definePageMeta

`definePageMeta` is a compiler macro that you can use to set metadata for your **page** components located in the `pages/` directory (unless set otherwise). This way you can set custom metadata for each static or dynamic route of your Nuxt application.

pages/some-page.vue

```vue
<script setup lang="ts">
definePageMeta({
  layout: 'default'
})
</script>
```

👉 Read more in Docs > Guide > Directory Structure > Pages > #page Metadata.

Type

```ts
definePageMeta(meta: PageMeta) => void

interface PageMeta {
  validate?: (route: RouteLocationNormalized) => boolean | Promise<boolean> | Partial<NuxtError>
  redirect?: RouteRecordRedirectOption
  name?: string
  path?: string
  alias?: string | string[]
  pageTransition?: boolean | TransitionProps
  layoutTransition?: boolean | TransitionProps
  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
  [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/` directory.

  `path`

  - **Type**: `string`
    You may define a path matcher, 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`
    Set to `true` when you want to preserve page state across route changes or use the `KeepAliveProps` 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`
  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 | Array<MiddlewareKey | NavigationGuard>`
  Define anonymous or named middleware directly within `definePageMeta`. Learn more about route middleware.

`pageTransition`

- **Type**: `boolean | TransitionProps`
  Set name of the transition to apply for current page. You can also set this value to `false` to disable the page transition.

`redirect`

- **Type**: `RouteRecordRedirectOption`
  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 docs) 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**.

Examples

Basic Usage

The example below demonstrates:

- how `key` can be a function that returns a value;
- how `keepalive` property makes sure that the `<modal>` component is not cached when switching between multiple components;
- adding `pageType` as a custom property:

pages/some-page.vue

```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:

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>
```

Defining Layout

You can define the layout that matches the layout's file name located (by default) in the `layouts/` directory. You can also disable the layout by setting the `layout` to `false` :

pages/some-page.vue

```
<script setup lang="ts">
definePageMeta({
  // set custom layout
  layout: 'admin'
```

```
  // ... or disable a default layout
  layout: false
})
</script>
```

Edit on Github