# Pages and Layouts

## app.vue

Nuxt 3 provides a central entry point to your app via `~/app.vue`. 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 about `app.vue`.

## Migration

1. 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.

# Layouts

If you are using layouts in your app for multiple pages, there is only a slight change required.

In Nuxt 2, the `<Nuxt>` 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 `<slot />`. This also allows advanced use cases with named and scoped slots. 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 `<Nuxt />` with `<slot />`
2. Use `definePageMeta` to select the layout used by your page.
3. Move `~/layouts/_error.vue` to `~/error.vue`. See the error handling docs. If you want to ensure that this page uses a layout, you can use the `<NuxtLayout>` component directly within `error.vue`:

```
<template>
  <div>
    <NuxtLayout name="default">
      <!-- -->
    </NuxtLayout>
  </div>
</template>
```

## Example: `~/layouts/custom.vue`

**layouts/custom.vue** pages/index.vue

layouts/custom.vue

```
  <template>
    <div id="app-layout">
      <main>
-       <Nuxt />
+       <slot />
      </main>
    </div>
  </template>
```

# Pages

Nuxt 3 ships with an optional `vue-router` integration triggered by the existence of a `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 `<Nuxt>` and `<NuxtChild>`. In Nuxt 3, these have been replaced with a single `<NuxtPage>` component.

## Page Keys and Keep-alive Props

If you were passing a custom page key or keep-alive props to `<Nuxt>`, you will now use `definePageMeta` to set these options.

See more about migrating Nuxt component hooks.

## 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. The `style` prop from `<Nuxt>` no longer applies to transition when used on `<slot>`, so move the styles to your `-active` class.

Read more about `pages/`.

## Migration

1. Rename any pages with dynamic parameters to match the new format.
2. Update `<Nuxt>` and `<NuxtChild>` to be `<NuxtPage>`.
3. If you're using the Composition API, you can also migrate `this.$route` and `this.$router` to use `useRoute` and `useRouter` composables.

## Example: Dynamic Routes

**Nuxt 2** Nuxt 3

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
```

## Example: Nested Routes and `definePageMeta`

**Nuxt 2** Nuxt 3

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>
```

## `<NuxtLink>` Component

Most of the syntax and functionality are the same for the global NuxtLink component. If you have been using the shortcut `<NLink>` format, you should update this to use `<NuxtLink>`.

`<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, in the docs.

# 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.

**Note:** Ensure to always `await` on `navigateTo` or chain it's result by returning from functions.

**Nuxt 2** Nuxt 3

Nuxt 2

```
<script>
export default {
  methods: {
    navigate(){
      this.$router.push({
        path: '/search',
        query: {
          name: 'first name',
          type: '1'
        }
      })
    }
  }
}
</script>
```

Edit on Github