Vue.js Development

Nuxt uses Vue as a frontend framework and adds features such as component auto-imports and file-based routing. Nuxt 3 integrates Vue 3, the new major release of Vue that enables new patterns for Nuxt users.

> While an in-depth knowledge of Vue is not required to use Nuxt, we recommend that you read the documentation and go through some of the examples on vuejs.org.

Nuxt has always used Vue as a frontend framework. We chose to build Nuxt on top of Vue for these reasons:

- The reactivity model of Vue, where a change in data automatically triggers a change in the interface.
- The component-based templating, while keeping HTML as the common language of the web, enables intuitive patterns to keep your interface consistent, yet powerful.
- From small projects to large web applications, Vue keeps performing well at scale to ensure that your application keeps delivering value to your users.

Vue with Nuxt

Single File Components

Vue's single-file components (SFC, or `*.vue` files) encapsulate the markup (`<template>`), logic (`<script>`) and styling (`<style>`) of a Vue component. Nuxt provides a zero-config experience for SFCs with Hot Module Replacement that offers a seamless developer experience.

Components Auto-imports

Every Vue component created in the `components/` **directory** of a Nuxt project will be available in your project without having to import it. If a component is not used anywhere, your production's code will not include it.

Vue Router

Most applications need multiple pages and a way to navigate between them. This is called **routing**. Nuxt uses a `pages/` **directory** and naming conventions to directly create routes mapped to your files using the official Vue Router library.

Example

Open on StackBlitz | Open on CodeSandbox

The `app.vue` file is the entry point, which represents the page displayed in the browser window.

Inside the `<template>` of the component, we use the `<Welcome>` component created in the `components/` **directory** without having to import it.

Try to replace the `<template>` 's content with a custom welcome message. The browser window on the right will automatically render the changes without reloading.

> 💡 If you're familiar with Vue, you might be looking for the `main.js` file that creates a Vue app instance. Nuxt automatically handles this behind the scenes.

If you were a previous user of Nuxt 2 or Vue 2, keep reading to get a feel of the differences between Vue 2 and Vue 3, and how Nuxt integrates those evolutions.

Otherwise, go to the next chapter to discover another key feature of Nuxt: Rendering modes.

Differences with Nuxt 2 / Vue 2

Nuxt 3 is based on Vue 3. The new major Vue version introduces several changes that Nuxt takes advantage of:

- Better performance
- Composition API

- TypeScript support

Faster Rendering

The Vue Virtual DOM (VDOM) has been rewritten from the ground up and allows for better rendering performance. On top of that, when working with compiled Single-File Components, the Vue compiler can further optimize them at build time by separating static and dynamic markup.

This results in faster first rendering (component creation) and updates, and less memory usage. In Nuxt 3, it enables faster server-side rendering as well.

Smaller Bundle

With Vue 3 and Nuxt 3, a focus has been put on bundle size reduction. With version 3, most of Vue's functionality, including template directives and built-in components, is tree-shakable. Your production bundle will not include them if you don't use them.

This way, a minimal Vue 3 application can be reduced to 12 kb gzipped.

Composition API

The only way to provide data and logic to components in Vue 2 was through the Options API, which allows you to return data and methods to a template with pre-defined properties like `data` and `methods`:

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment(){
      this.count++
    }
  }
}
</script>
```

The Composition API introduced in Vue 3 is not a replacement of the Options API, but it enables better logic reuse throughout an application, and is a more natural way to group code by concern in complex components.

Used with the `setup` keyword in the `<script>` definition, here is the above component rewritten with Composition API and Nuxt 3's auto-imported Reactivity APIs:

```
<script setup lang="ts">
const count = ref(0);
const increment = () => count.value++;
</script>
```

The goal of Nuxt 3 is to provide a great developer experience around the Composition API.

- Use auto-imported Reactivity functions from Vue and Nuxt 3 built-in composables.
- Write your own auto-imported reusable functions in the `composables/` directory.

TypeScript Support

Both Vue 3 and Nuxt 3 are written in TypeScript. A fully typed codebase prevents mistakes and documents APIs usage. This doesn't mean that you have to write your application in TypeScript to take advantage of it.

🔎 Read the details about TypeScript in Nuxt 3

Edit on Github