How Nuxt Works?

Nuxt is a minimal but highly customizable framework to build web applications. This guide helps you better understand Nuxt internals to develop new solutions and module integrations on top of Nuxt.

The Nuxt Interface

When you start Nuxt in development mode with `nuxi dev` or building a production application with `nuxi build`, a common context will be created, referred to as `nuxt` internally. It holds normalized options merged with `nuxt.config` file, some internal state, and a powerful hooking system powered by unjs/hookable allowing different components to communicate with each other. You can think of it as **Builder Core**.

This context is globally available to be used with nuxt/kit composables. Therefore only one instance of Nuxt is allowed to run per process.

To extend the Nuxt interface and hook into different stages of the build process, we can use Nuxt Modules.

For more details, check out the source code.

The NuxtApp Interface

When rendering a page in the browser or on the server, a shared context will be created, referred to as `nuxtApp`. This context keeps vue instance, runtime hooks, and internal states like ssrContext and payload for hydration. You can think of it as **Runtime Core**.

This context can be accessed using `useNuxtApp()` composable within Nuxt plugins and `<script setup>` and vue composables. Global usage is possible for the browser but not on the server, to avoid sharing context between users.

To extend the `nuxtApp` interface and hook into different stages or access contexts, we can use Nuxt Plugins.

Check Nuxt App for more information about this interface.

`nuxtApp` has the following properties:

```
const nuxtApp = {
  vueApp, // the global Vue application: https://vuejs.org/api/application.html#application-api

  versions, // an object containing Nuxt and Vue versions

  // These let you call and add runtime NuxtApp hooks
  // https://github.com/nuxt/nuxt/blob/main/packages/nuxt/src/app/nuxt.ts#L18
  hooks,
  hook,
  callHook,

  // Only accessible on server-side
  ssrContext: {
    url,
    req,
    res,
    runtimeConfig,
    noSSR,
  },

  // This will be stringified and passed from server to client
  payload: {
    serverRendered: true,
    data: {},
    state: {}
  }

  provide: (name: string, value: any) => void
}
```

For more details, check out the source code.

Runtime Context vs. Build Context

Nuxt builds and bundles project using Node.js but also has a runtime side.

While both areas can be extended, that runtime context is isolated from build-time. Therefore, they are not supposed to share state, code, or context other than runtime configuration!

`nuxt.config` and Nuxt Modules can be used to extend the build context, and Nuxt Plugins can be used to extend runtime.

When building an application for production, `nuxi build` will generate a standalone build in the `.output` directory, independent of `nuxt.config` and Nuxt modules.

Edit on Github