Nuxt 3 Modules and Open-Source

More from the talk - Slides

Introduction

Hi Vue Toronto, thanks for having me! And let's start with some numbers, shall we?
So… what do you think 2 million refers to? That's the number of downloads Nuxt had during the last 30 days.
How about 15 million then? Well, you probably saw that one coming. That's the number of downloads Vue had during the last 30 days.
Alright, so Vue is 15 million, Nuxt is 2… what can 14 million be? Must be something big… Turns out that's the number of downloads Nuxt modules had over the last 30 days.
Interesting stat uh? What this means, is that either one person installed 14 million modules on their project last month, OR, that on average, every Nuxt project relies on 7 modules. Math cannot figure out which one is true…
But when you think about it, this definitely means that Nuxt modules are a central part of Nuxt, and I'd like to talk to you about them today, especially in the context of Nuxt 3. So let's see:

- What Nuxt 3 modules are about
- But more importantly, how you can make some and why you can be interested in doing so
- Finally, we will see how they can get you started and involved in open-source

So let's get started, shall we?

Personal Introduction

OK, but before that, I'm Lucie Haberer. I'm from France - as you probably guessed from my accent.
I've started my developer journey working with MediaWiki, the software Wikipedia runs on. But nowadays I'm working at Prismic as a Developer Experience Engineer… hence today's t-shirt.
By the way, Prismic is a Headless CMS & Page Builder. I'm having fun there, managing the company's open-source ecosystem with my team, while also doing streams, videos, and other cool things.
Finally, I'm a Nuxt contributor and ambassador. I've been working with Nuxt for almost 5 years now… so you start to see the link with this talk.
Anyway, enough me-chatting, let's dive into it.

What are Nuxt Modules

So… first question… what are Nuxt Modules - apart from the fact they get a lot of downloads?
I'll try to answer this question assuming you're new to Nuxt, so that everyone is able to follow along.
Nuxt is a meta-framework for Vue.js, and to quote its documentation:

“Modules are Nuxt.js extensions, which can extend its core functionality, and add endless integrations.”
Pretty powerful uh? Basically, this means that Nuxt modules are like addons for Nuxt.js.
They allow you to extend it in pretty much any way possible: from turning your website into a PWA, to flashing the connected lights in your room, when there’s an error within your code. - I certainly did that.
For a more concrete approach, Nuxt modules are registered inside your Nuxt config file. They can be:

- a package name
- a relative path
- or a module definition code directly.

Specific module options can be provided within the module declaration or through specific config keys.
Alright, so that leaves us with a relative understanding of Nuxt Modules. Let’s just add some details to that.
First of all, we’ll wind back a tiny bit and talk about modules with Nuxt 2.
Within Nuxt 2 the module ecosystem has grown rich. Talking about PWA earlier on, guess what? There’s a Nuxt module for that. It takes care of all the gotchas related to making that possible, and is easy to set up.
Another random example, you want to connect to your favorite CMS and don’t know what to do? Again, there’s a module for that, and it’s maintained by a wonderful person, what more to ask? OK, got it, there’s actually a whole lot of other wonderful people maintaining great CMS modules. Go check any of them out!
But you got my point. With Nuxt 2, you wanna do something? Guess what? There’s a module for that.
You can go check out this site, for the most comprehensive list of available modules for Nuxt 2… and Nuxt 3!
So let’s talk about Nuxt 3. Nuxt 3 modules are now a thing, and they are our main topic today. They are new, offer a nicer developer experience for module authors, and more!
The catch? Nuxt 3 module ecosystem is still relatively tiny today, but that will change, as Nuxt 3 is obviously the future of Nuxt.
Great! Before we move on, let’s recap what we discussed about Nuxt modules so far:

- Nuxt modules are like addons to the framework
- Both Nuxt 2 and Nuxt 3 modules are registered the same way within Nuxt config file
- Nuxt 2 module ecosystem is rich but about to be legacy
- Nuxt 3 module ecosystem is fancier but still a newborn today.

You can make your own!

Now that everyone knows the basics of Nuxt modules, I want to tell you something really important about them:
You can make your own!
Don’t believe me? Well, I can prove that!

Chances are, that if you're working with Nuxt, you're a developer. It happens that developers do write code, and Nuxt modules are made just of that.
And… I don't know… but between you and me, I can see a pretty straight line from you, to Nuxt modules, so you can make modules!
OK, but more realistically, why would you want to make Nuxt modules of your own? Especially with so many modules already out there?
Multiple reasons for that:
As we've seen, you can do pretty much anything with Nuxt modules. If you're working on multiple Nuxt projects, modules are a great way to abstract code from them, and share it reliably within all your projects.
This works great if you're working for an agency making websites for others, or an enterprise managing different internal applications. But for example, I personally extracted some code of mine into such modules with Nuxt 2 so that I had:

- One that managed Vue Meta the way I want it to be managed
- Another one that transformed Nuxt payloads in a particular way
- And a final one that outputs statistics about the project

Another great use-case for making your own Nuxt modules is for facilitating the integration with the tools you use.
While there are modules out there for Tailwind CSS or Sentry, there might not be one about a specific service you use. Maybe it happens to be new or very niche - this is especially true with Nuxt 3 because the ecosystem is fresh there.
In that case, why not create it? It doesn't need to be super fancy to get started!
But alright, sure! There are reasons to create Nuxt modules, but how do you do it? Well, that's what I'm about to tell you. Let's learn how to create Nuxt modules, in Nuxt 3.

Nuxt Modules in Nuxt 3

If you never built Nuxt 2 modules that's fine. If you did, modules in Nuxt 3 have evolved a bit from Nuxt 2 modules, but still feel pretty similar. Compared to them, they come with:

- First-class TypeScript support
- Stronger structure
- Nicer developer experience and defaults

Let's see more of that concretely!

Creating a module

We'll start by creating a module project, to do so we have to use *nuxi*, Nuxt 3 CLI.
Once our project is created, we can install dependencies with either NPM - or your favorite package manager, I'm not here to judge.
And because I don't have any trust in the wifi connection in this room, I already did that ahead of time in this project here.
Alright, so we created a module project, and installed dependencies, and we ended up with that.
Let's take a minute to analyze that project structure.
First, we have two directories:

- The *src* directory holds the source code of our module, this is where we will work.
- The *playground* directory next to it, contains a Nuxt 3 application configured with our module. We will use it while developing.

Then, let's have a look at the package.json. The first thing we can see here, is that it's already configured to be shipped properly to NPM, neat!
Moving on, we can see it has a few scripts:

- Prepack takes care of building our module. Nuxt 3 modules come with an in-house builder created by Nuxt that is already configured for you. That's great because you don't have to go through the headache of learning and configuring one.
- Then we have dev commands that basically launch or build our playground that we use to develop our module.
- The *dev:prepare* command is a bit special, basically it's used to build TypeScript types, but those will also be built as we start our dev server, so let's ignore it for today.

Finally, the noteworthy things in the dependencies are:

- @nuxt/kit, this is a collection of utilities to develop our modules, we'll use it later on.
- @nuxt/module-builder, this is the package providing the build command for our module we talked about before.

Regarding the other root files, those are pretty common ones. Feel free to fine-tune them to your liking.
One thing though, if you're building a module that's meant to be shared outside of your organization, I'd recommend sticking with Nuxt ESLint configuration. This will allow your module to share a consistent coding style with the other community modules out there.
Now that we're familiar with our project structure, let's focus on the *src* directory. Here we have a *module.ts* file. It is our module entry point.
You probably noticed that it's a TypeScript file. Actually, Nuxt 3 puts the incentive on TypeScript as it results in a much-improved developer experience for end-users.
But no worries, if you're not comfortable with TypeScript, it's still possible to write modules in plain JavaScript.
As this would get us a bit off-track for today, we'll stick with that file, but we'll stay away from TypeScript black magic: everything will look like JavaScript, no worries again.
If we now peek at this module file, what does it looks like? Oh, there's already a bunch of code there!
The first thing we can see is that it's importing a few utilities from @nuxt/kit, mainly *defineNuxtModule* which is used to, namely, define a module.
Then we have a TypeScript interface. It's a way to describe an object structure in TypeScript. Here it's defining our module options, but as promised, no black magic today so we can safely ignore it.
Finally, our module export. As suggested, our module is defined using the *defineNuxtModule* helper from @nuxt/kit, and our module is the object right here. Let's explain its structure:

- The *meta* property is an object containing our module meta, here it's name and the config key we'd like to use in Nuxt configuration to pass options to our module. Like this. It can contain other metadata like our module compatibility, but let's leave that out for today.

- Then we have *defaults*. This is an object defining our module default options which will be extended with options provided in Nuxt config. It can also be a function to programmatically define those defaults.
- Finally, we have the *setup* function. It's the code of our module that will get executed by Nuxt. This function can be asynchronous, but it's recommended to keep it synchronous when possible as it can slow down Nuxt start process otherwise. It receives two arguments: the options the user provided to your module, and a Nuxt object.
  Here, we can see that our setup function already has some example code in it. Let's ignore it for now. We'll come back to it later.

Alright, so that was a really dense section touring the new module anatomy in Nuxt 3. Let's recap the main takeaways from it:

- Creating a Nuxt module can be done with *nuxi init*
- Initialized modules come with their own defaults and bundler by Nuxt, it's nice to stick with them
- @nuxt/kit is a collection of utilities for developing Nuxt modules
- Modules are TypeScript by default
- And defined through an object passed to the *defineNuxtModule* helper provided by @nuxt/kit.

Doing things with our module

Now that we're familiar with our module's anatomy. Let's make our module do a few things!
Let's start with a quick one by just logging some cheesy stuff.
In France, apéritif or apéro is a sacred moment of the day that happens at 7pm which basically is the French Happy Hour. Some will argue it's earlier, or that it's apéro anytime, but let's stick with 7pm and warn our developers about it.
And without much code, if I were to launch my dev server at apéro time, I'd get this important warning in my console. That's already a super useful module in my opinion.
OK, but I get you might not all be French here, difficult to relate… Also, logging ain't super exciting I admit, but you can still use it to provide information to developers using your module, share stats for nerds, etc.
Anyway, let's do something more advanced than logging: programmatically updating our Nuxt config. For example, we can have our module disabling SSR, just like this.
This is maybe a weird thing to do for sure, but really, you can edit anything in Nuxt config with a module.
Let's go more useful and see how our module could provide a default stylesheet. This allows me to present to you the runtime directory that I ignored until now.
The runtime directory contains pieces of code that will be used by your application directly, not by Nuxt.
With Nuxt modules it's expected to be a child directory of the *src* directory like I have here.
Inside it, I can create a *style.css* filelor, and add some style to it.
OK, so, now, how do I load it into Nuxt from my module? To do so, we'll use the *createResolver* helper from @nuxt/kit. This is just a helper that allows us to resolve file paths consistently.

Now, we can simply push our CSS file to Nuxt CSS assets array, and we end up with a big, flashy, red background. Great!
Let's keep working with the runtime directory, what about adding a Nuxt plugin from it this time? Our module could, for example, provide a plugin that injects some formatting utilities into our code.
To do so, we use our resolver again to resolve our plugin path. Then we need to do two things:

1. telling Nuxt to transpile our plugin
2. and registering our plugin with the *addPlugin* helper from @nuxt/kit.

Now that our plugin is available, we can inject utilities with it like this. And use them in our application thanks to our module.
Alright, so those are 3 things you can do with modules. Of course, you can do a lot more, like adding auto-imports for composables in your runtime directory, same for components, and many more.
There's one thing, however I haven't touched on so far that is quite central to Nuxt, it's hooks.
Hooks are:

- Windows to Nuxt internals which you can hook to.
- They allow you to get information and perform certain actions when Nuxt does something specific.
- I admit, hooks are a bit oriented towards advanced use-case but are really what make modules capable of anything.

There are two ways to register hooks, the first one is programmatic using *nuxt.hook*. For example, we can log how many routes are about to be generated this way.
The second way is a sugar, you can actually provide a map of hooks to hook to in your module definition directly like this.
Again, that was a dense section, so here are the main takeaways for you:

- We can run any type of code in our module *setup* function
- Lot of utilities are made available by @nuxt/kit to perform common tasks, feel free to explore them!
- The runtime directory is where we can put assets: css files, plugins, Vue components, for our module to inject, etc.
- Hooks are windows to Nuxt internals, they allow you to fine-tune and react to Nuxt lifecycles.

Next steps

OK, so, we've been through Nuxt 3 module structure, and how to make our module do a few things. What are the next steps from here?
First thing, and important one, we can just publish our module to npm. We just need to pick a version number and run the *npm publish* command.
Next up, if you were to write a module, chances are that you'd like to test it at some point. In that regard, the Nuxt team provides you with a test utils package package to help you in that endeavor.
Finally, this is not super official, but widely adopted by Nuxt community, it's the usage of *standard-version* CLI to publish your module. It's a really neat CLI that streamlines the

publishing process by updating smartly your package version number and generating a comprehensive changelog based on your commit messages. Feel free to check it out.

Perks to learning Nuxt modules

That's all folks! No. That's not all. Actually, I didn't say everything to you when explaining why learning to build Nuxt modules was great. There are some extra perks to it.
The first one, is that by learning Nuxt modules, you'll get a deeper understanding of Nuxt behaviors and internals. This will help you understand better how Nuxt works and troubleshoot issues you might encounter while developing with it.
A related fun fact, Nuxt itself is partly made out of modules, and part of it also relies on module APIs.
Then, as we've seen, Nuxt modules are really well structured. Being familiar with that structure is a valuable skill as this allows you to be familiar with any module codebase available out there.
Concretely, by being familiar with that structure, you can:

- Understand better what the modules you're using are doing. In that regard, feel free to be curious and look at the source code of the module you use. I learned a lot about Nuxt this way.
  Quickly, if we look at Nuxt color mode module, we can see that it's exposing its options to the runtime app by creating a file importable through the #color-mode-options alias. That's a neat pattern!
- Secondly, being familiar with Nuxt modules' structure allows you to contribute to the ecosystem. Found a bug while using one? Craving for a feature on your favorite? Go and open pull requests! You know what the codebase is about and the community is really friendly to newcomers.
  Personally, that's how I became comfortable with open-source and contributing code to codebases that I don't own. If you're looking to learn and want to be involved, that's an excellent way to get started!
  And again, all community modules are easily searchable on modules.nuxtjs.org, and there are plenty of modules to make or port to Nuxt 3!

Conclusion

So that was "Nuxt 3 Modules and Open-Source". Nuxt modules are a way to extend Nuxt functionalities endlessly and I hope you learned a few things about them. Most importantly:

- That you can make Nuxt modules, for your team or the whole community
- That modules are nicely structured and their developer experience greatly improved with Nuxt 3
- And that learning that structure is a great way to understand Nuxt better and get involved in the community.

You can find all the info about this talk, slides, source code, and extra links, at lucie.red/toronto.
Finally, you can also follow me, and my nonsense, on Twitter at lucie.red/twitter
Thanks everyone!