> *In a rush? Skip to technical tutorial or live demo*

Working in this field, I often have developer friends reach out to me with questions about e-commerce projects.

I was talking with one of them last week. He seemed a bit distressed:

“I have this client that is asking for the moon in terms of performance for his online store. I really don’t know how I can meet his expectations.”

I sat there and smiled back at him because I knew the answer to his problem right away.

Many factors will influence your choice of stack for e-commerce. But, when speed is the main one, look no further than Hugo.

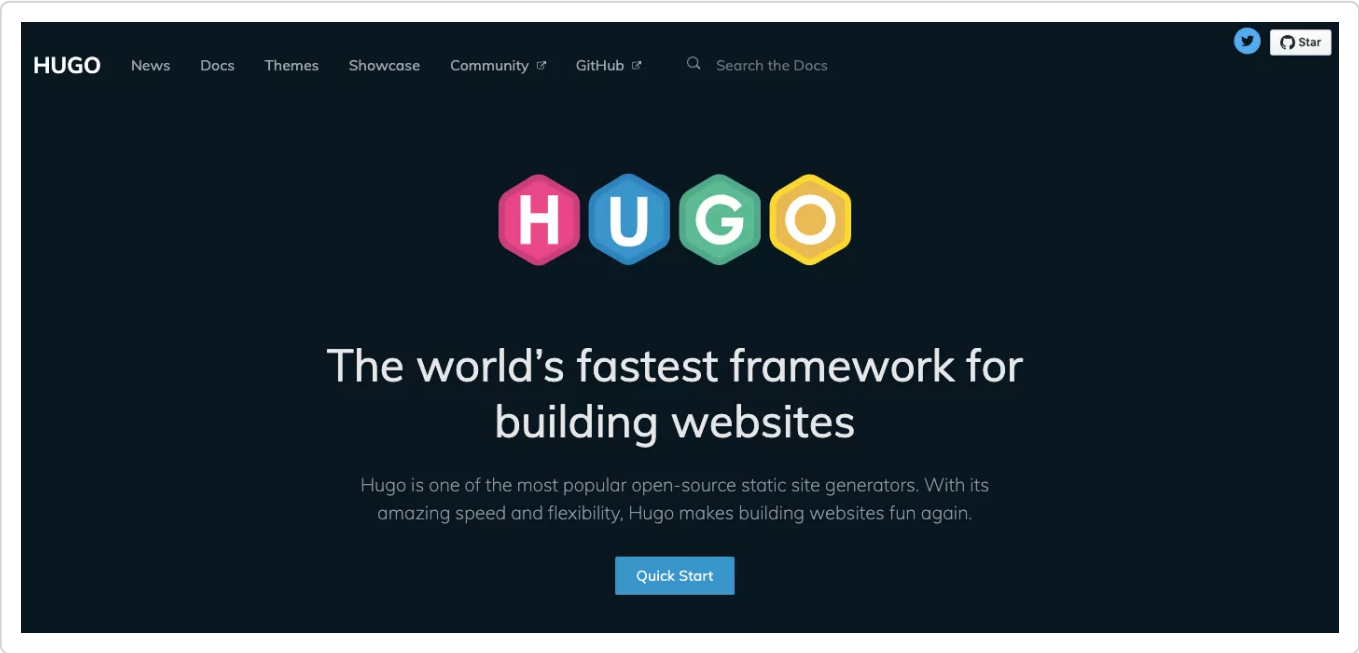
This conversation inspired me **to dig up our old Hugo e-commerce demo and to give it a fresh twist**, so here I am.

Here's what I'll do in the tutorial below:

- Create a Hugo website
- Add e-commerce functionalities with Snipcart
- Pair the static site to a CMS (Forestry)
- Deploy on Netlify

First, let's see what Hugo has been up to since our last visit.

What is Hugo?



In a nutshell, Hugo is a widely popular open-source static site generator written in Go. Released in 2013, it was one of the first SSGs launching the rise of the "static" web as we know it today.

> *Learn more about the Golang e-commerce ecosystem right here.*

It's still one of the more popular tools in this field, for many reasons. Hugo comes off as one of the most efficient ways we've seen to build, manage and update modern static sites. It's easy to install on any platform, plus you can host it anywhere. But above all, **its build times are just off the charts.**

To give you an idea, build times for Jekyll, another very popular SSG, are about ten times higher than those of Hugo.

It's hard for any other tools to compete with Hugo's speed when build times are under 1 second.

Other than speed, here's why you should consider Hugo for e-commerce projects (or any project, for that matter):

- It's flexible—it enables your website to scale thanks to many out-of-the-box functionalities.
- It's backed by an active community—it's probably the SSG with the most thriving community, always ready to help.
- It supports more than small blogs/stores—it has multiple output types and multilingual capacities; Hugo's ready for enterprise websites.

The crazy thing about Hugo is that it hasn't really changed since we wrote the first version of this post back in 2016. It was a solid tool right out of the gate. The community has evolved though, and so has the Hugo development experience. Take theming for instance; there are way more (and better) themes available now than back then.

Also, headless CMSs are making life easier for everyone.

Managing content with a headless CMS

Options for content management

A significant event that has happened in modern web development since we first wrote this post is the rise of the headless CMS.

It has made the JAMstack and static web way more accessible. If you think dealing with Markdown-driven static content files won't cut it for clients, these cool tools can help edit and manage content on top of Hugo. That's what I'll do in the demo below.

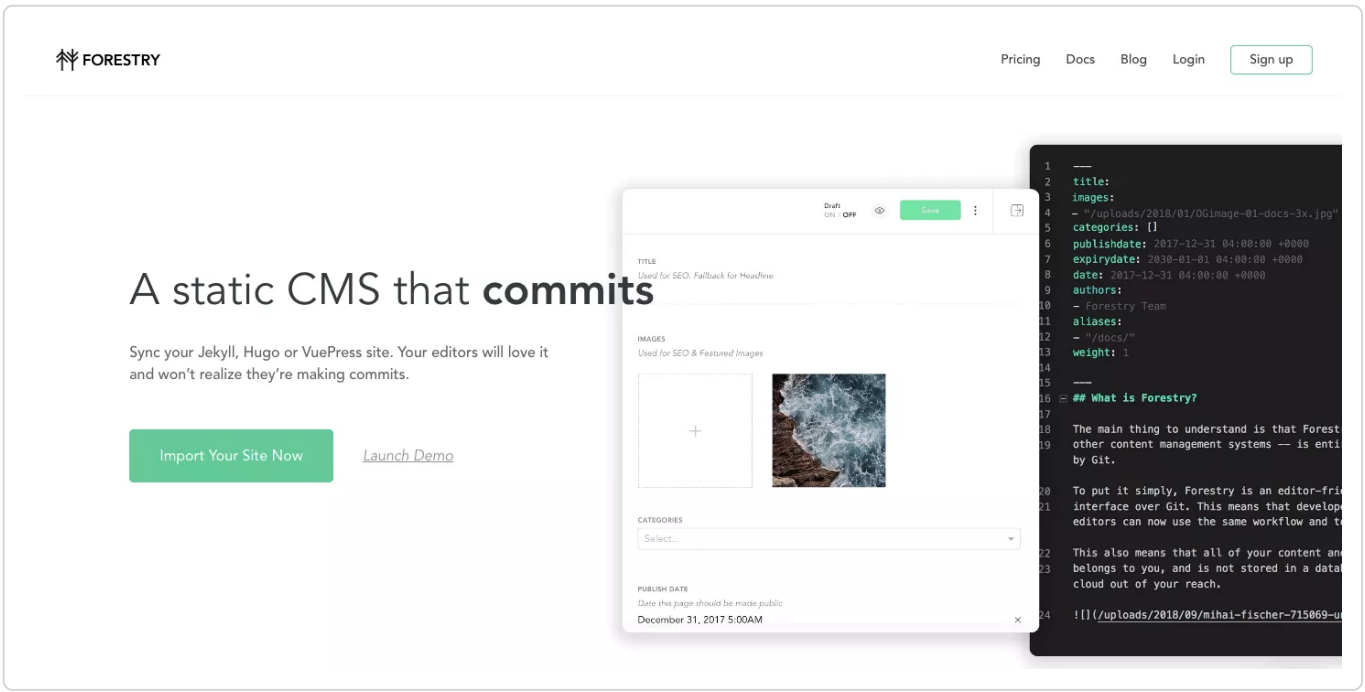
We suggest throwing one of the following headless CMS into the mix:

- Sanity.io
- DatoCMS
- Netlify CMS
- Forestry.io

The latest is the one I chose for this tutorial. It's pretty dope. Let's take a closer look!

> *There are way more headless options for e-commerce. If the ones mentioned here don't make it for you, consult this post we wrote on the subject.*

What is Forestry?



Forestry.io is a static, headless CMS that easily syncs with Hugo. It makes it the perfect choice for today's use case.

Although, you should know that Forestry is also built to support Jekyll and VuePress sites. You simply write frontend code with your SSG and push through Git. Forestry then pulls in the commits and updates the CMS. It's then easy for editors, marketing teams, and merchants to create & edit content in its rich UI.

This process brings many advantages compared to more traditional CMSs, like speed, reliability, continuous development, and a unified workflow.

It can be hosted anywhere you want. I'll be using Netlify down below.

Like many modern CMSs, it was built as an alternative to WordPress. It provided exactly what was needed to complete already powerful frontend tools such as Hugo & Jekyll—simple content management, roles & permissions, etc.

Okay, enough rambling, let's get practical.

Hugo tutorial: static e-commerce on top of Forestry.io

In this demo, I'll create a simple e-commerce store with the help of Hugo and Forestry.io. I'll also deploy it in a few clicks all thanks to Netlify. Let's do this!

Prerequisites

- Basic JS, HTML & CSS knowledge
- A Snipcart account (forever free in Test mode)

1. Installing Hugo

Before getting started, you'll need to install the Hugo CLI.

Since this entirely depends on your operating system, you can refer to the official documentation. Just keep in mind that if you want to make use of `Sass` like I will in this demonstration, you'll need to install the extended version of Hugo.

2. Creating a new site

Now that the Hugo CLI is installed and ready to go, you can create your new site with this simple command:

```
hugo new site <YOUR_PROJECT_NAME>
```

This should create a new directory with the following structure.

```
.
├── archetypes
├── content
├── data
├── layouts
├── static
├── themes
└── config.toml
```

To give you a brief idea of what the important directories do, the `data` directory is used to store any configuration files that we might want to use while generating our website. In this case, you'll

use it as a means to store a list of products.

The `layout` directory stores your `html` templates. You'll create a template for the homepage and partials for the header, footer, and products.

The `static` folder is used to store any static content such as images, stylesheets, or javascript files.

Later on, you'll also make changes in the `content` directory to pair it with Forestry.io, a headless CMS, and create an `assets` folder to leverage Hugo Pipes for our `scss` files.

3. Using the built-in Hugo server

To start the static site generator, use the following command:

```
hugo server -D
```

This command should generate your pages. Watch for any incoming changes and deploy the site locally at the following URL: http://localhost:1313. Don't fret if the page is blank as it's exactly what it should look like at this stage.

The `-D` parameter will allow you to display drafted content. I'll talk more about this later on when we pair our site with a CMS.

4. Adding data

For this demonstration, I'll create a simple cheese shop that displays a list of products on the home page.

Therefore, add a `products.json` file inside the `data` directory describing our products according to Snipcart's product definition.

```
[
  {
    "id": "GOUDA_WHEEL",
    "name": "Gouda Wheel",
    "price": 299.95,
    "image": "/images/gouda.jpg",
    "description": "Want to step up your cheese game? Try our mild, yellow Gouda Wheel
    "product_url": "http://snipcart-hugo-forestery.netlify.com"
  },
  ...
]
```

5. Adding new layouts and partials

You won't be using a pre-made theme here. Therefore, you must create templates yourself.

> *You could also choose a pre-existing Hugo theme and skip this step.*

First, create an `index.html` file inside the `layouts` directory with the following code:

```
{{ partial "header.html" . }}
<main class="products">
  {{ range .Site.Data.products }}
    {{ partial "product.html" . }}
  {{ end }}
</main>
{{ partial "footer.html" . }}
```

As you can see, curly brackets (`{{ }}`) can be used to add content to your page programmatically.

In this case, you'll leverage this feature to ask Hugo to import the footer and header as well as list all the products stored in the predefined variable `.Site.Data` .

Partials need to be created inside a `layouts/partials` directory. Create a file name `header.html` that will include Snipcart's dependencies. Also, don't forget to add your API key if you choose to do so.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel='shortcut icon' type='image/x-icon' href='/images/icons/favicon.ico' />
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></sc
  <script src="https://cdn.snipcart.com/scripts/2.0/snipcart.js" data-api-key="YOUR_PU
  <link href="https://cdn.snipcart.com/themes/2.0/base/snipcart.min.css" rel="stylesh
  <title>CheeseCity</title>
</head>
<body>
  <header class="header">
    <a class="header__branding" href="/">
      <img class="header__logo" src="/images/icons/cheese.svg" alt="cheese logo">
      <h1 class="header__title">CheeseCity</h1>
    </a>
    <div class="header__cart">
      <a href="#" class="snipcart-checkout" style="text-decoration: none;">
        <div class="snipcart-summary">
          <img src="/images/icons/cart.svg" alt="">
          <span class="snipcart-total-price"></span>
        </div>
      </a>
    </div>
  </header>
```

You'll also create a `footer.html` file in the same way that we created your header.

```
  <footer class="footer">
    Powered by <a href="https://gohugo.io/">Hugo</a> & <a href="https:,
  </footer>
</body>
</html>
```

Lastly, let's create a `product.html` partial that will display what you want to showcase about your product. Keep in mind that your `Add to cart` button must once again follow Snipcart's product definition.

```
<div class="product">
  <h2 class="product__name">{{ .name }}</h2>
  <img class="product__image" src="{{ .image }}" alt="">
  <p class="product__description">{{ .description }}</p>
  <div class="product__button-container">
    <div class="product__price">${{ .price }}</div>
    <button
      class="snipcart-add-item buy-button"
      data-item-id="{{ .id }}"
      data-item-name="{{ .name }}"
      data-item-price="{{ .price }}"
      data-item-url="{{ .product_url }}"
      data-item-description="{{ .description }}">
      Add to cart
    </button>
  </div>
</div>
```

6. Add styling with Hugo pipes

At this stage, you can add an `assets` directory at the root of the project. This directory will hold all the files which need to be processed by Hugo Pipes. In this case, a directory named `scss` with your entry point stylesheet `main.scss` .

Once completed you can link your stylesheets in the `header.html` partial by adding the following code:

```
{{ $style := resources.Get "scss/main.scss" | resources.ToCSS | resources.Minify | re
  <link rel="stylesheet" href="{{ $style.Permalink }}">
```

It's as simple as that. If you look again in your browser, you should have a static e-commerce store!

7. Pairing the static site with a CMS

At this stage, you might want to add a CMS to your website. Thankfully, Hugo is an excellent match with most headless CMSs.

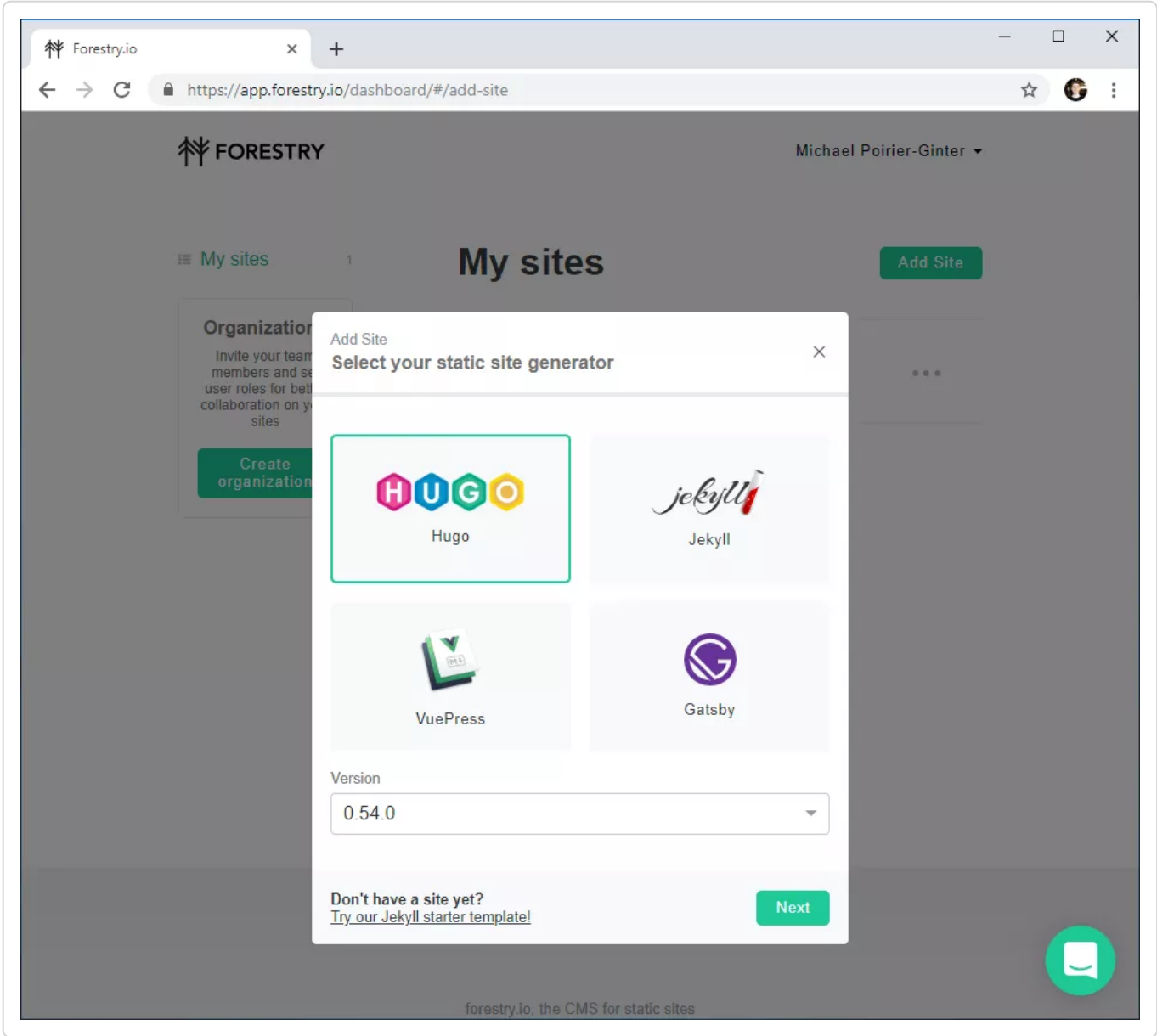
In fact, if your CMS exposes a REST endpoint, you can just change your loop in the `index.html` layout to the following:

```
{{ $products := getJSON "ENDPOINT_URL" }}
{{ range $products }}
  {{ partial "product.html" . }}
{{ end }}
```

As mentioned earlier, I'll use Forestry.io here, which will commit new content in the project directory, thus, regenerating the pages and preventing any unneeded HTTP calls.

7.1 Linking your repository with Forestry.io

Before getting started with Forestry, you'll need to make sure that your directory is stored in a Source Provider such as Github, Gitlab, or Bitbucket. Once this is done, you can sign into the Forestry.io website and follow the onscreen instructions.

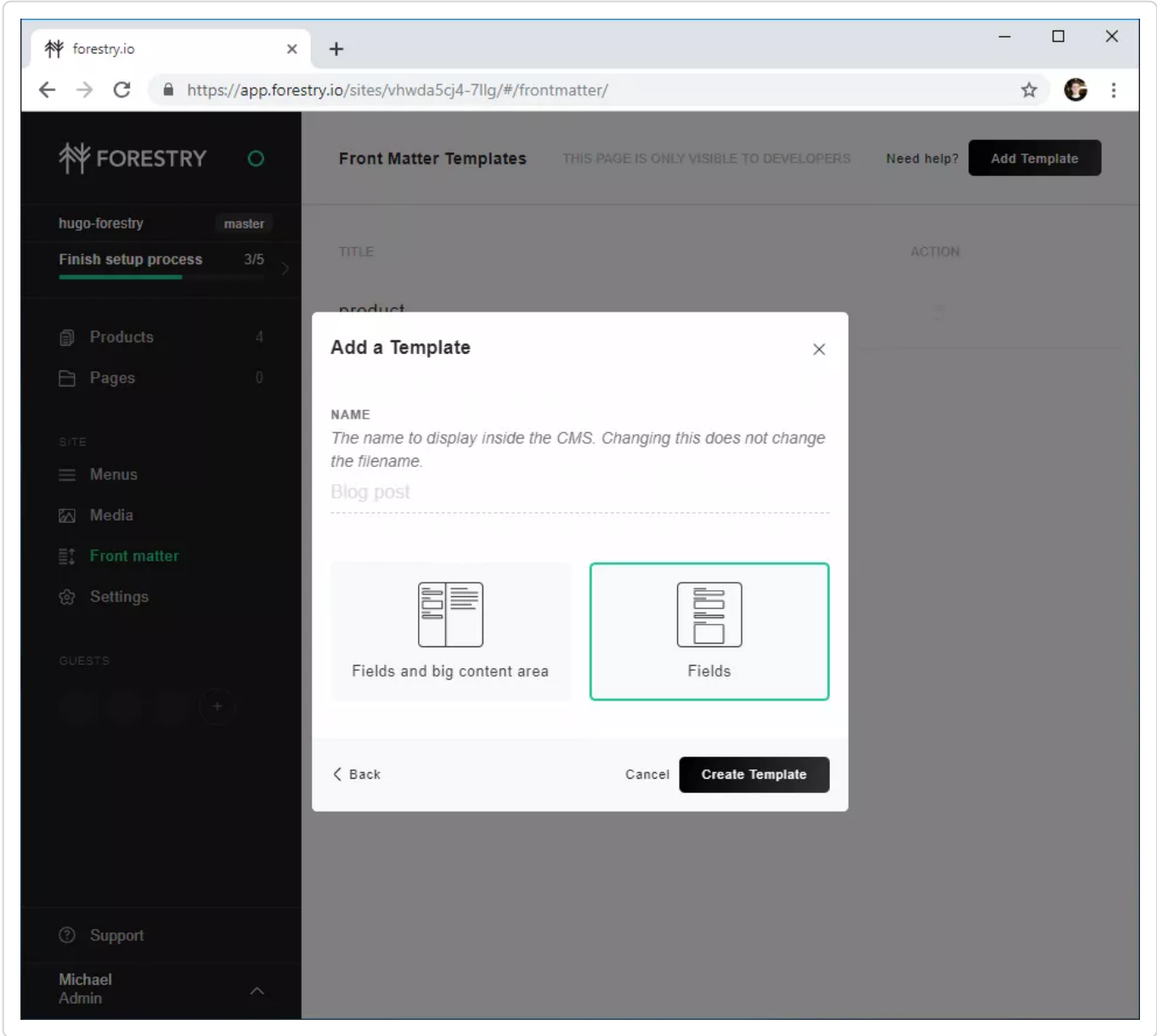


7.2 Setting up a new front matter

Once your repository is linked with Forestry, you can start defining your front matter inside the dashboard. In case you're wondering what that is, the front matter is what is used in Hugo to determine metadata attached to an instance of a content type.

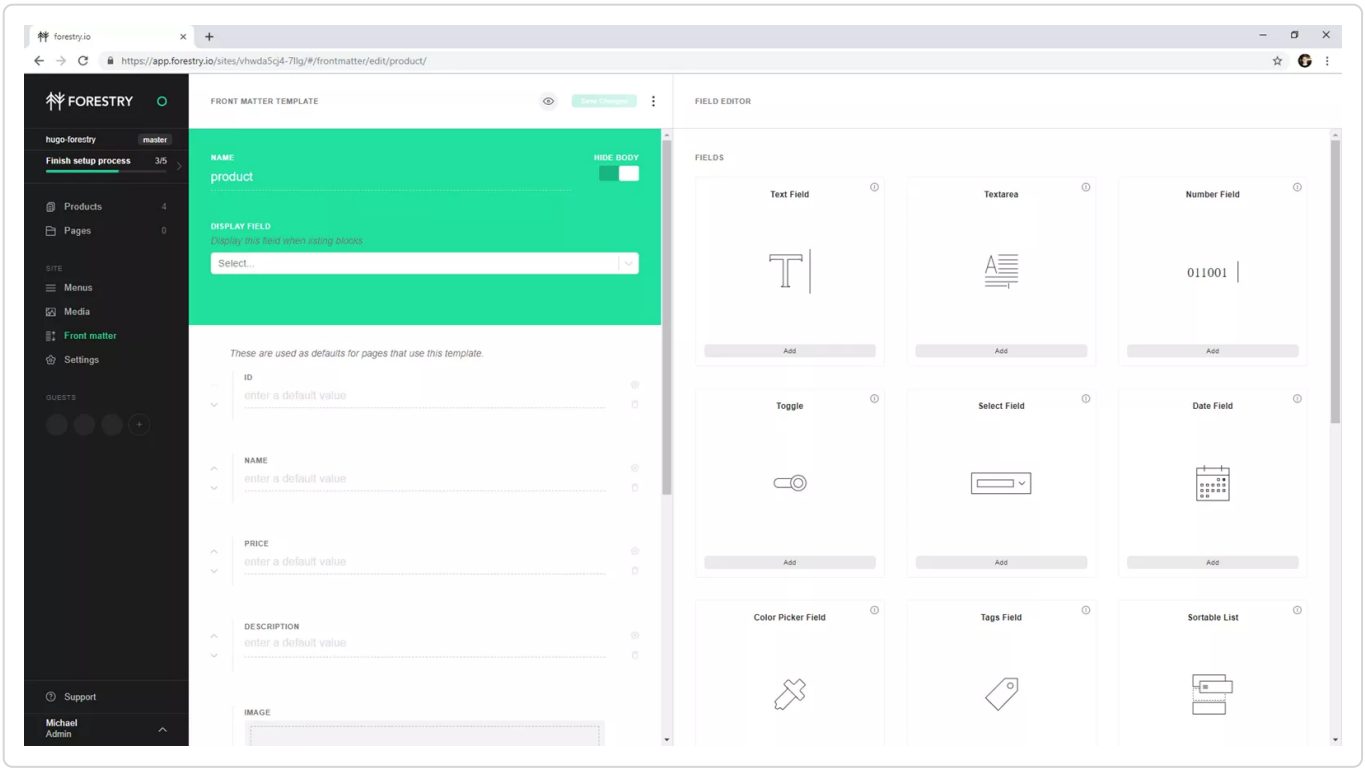
Adding a `JSON` file inside the `data` directory is excellent to get started rapidly, but if you want to create a site with multiple pages using content types is the way to go as it allows you to match types with custom layouts.

Here, you'll use it to define your product and pass that data to your `product.html` partial. You can do so by clicking on "Front matter" on the sidebar and then clicking "Add Template" in the top right corner.



Name it `product` and add the following fields: `id`, `name`, `price`, `description`, `image` and `product_url`.

Make sure that if you add anything else, it doesn't overwrite any predefined variables from Hugo. That's the reason why I used `product_url` rather than `url`.



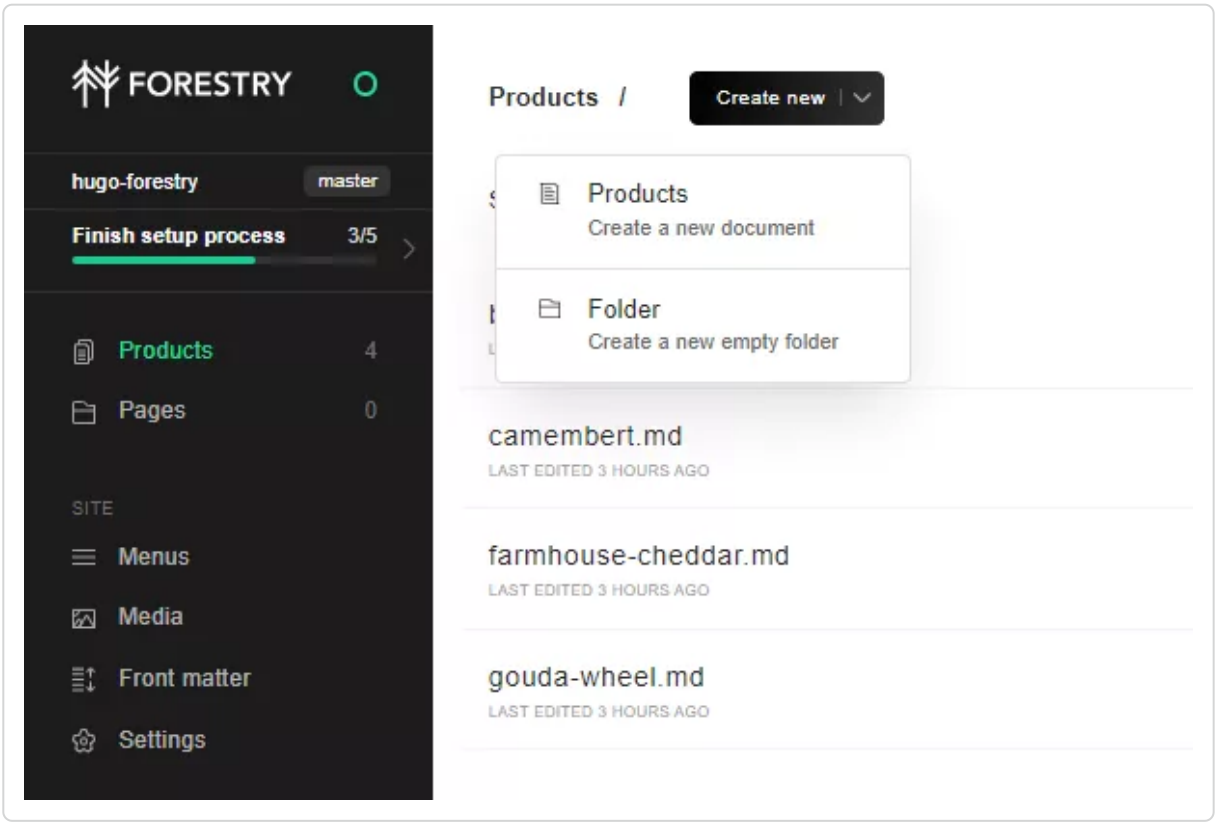
7.3 Creating a new section

Once you've created a new front matter, you can go in `settings > sidebar` and add a new section. You'll use this section, later on, to fetch all our products in our layout.

Click on `Add section` and create a new `Directory` with the following settings:

7.4 Adding a new document for each of your products

You should now see a new `Products` section on the sidebar. If you click on it, you should have a `Create new button` on the top left corner allowing you to catalog a new product.



You might notice that you can save the document as a `Draft`. If you choose to do so, that new piece of content will only be shown when then the `-D` flag is added to the `hugo server` command.

It's a great way to test out new content before exposing it to the public.

7.5 Updating the layout to display our new content

Now let's update the `product` loop to the following:

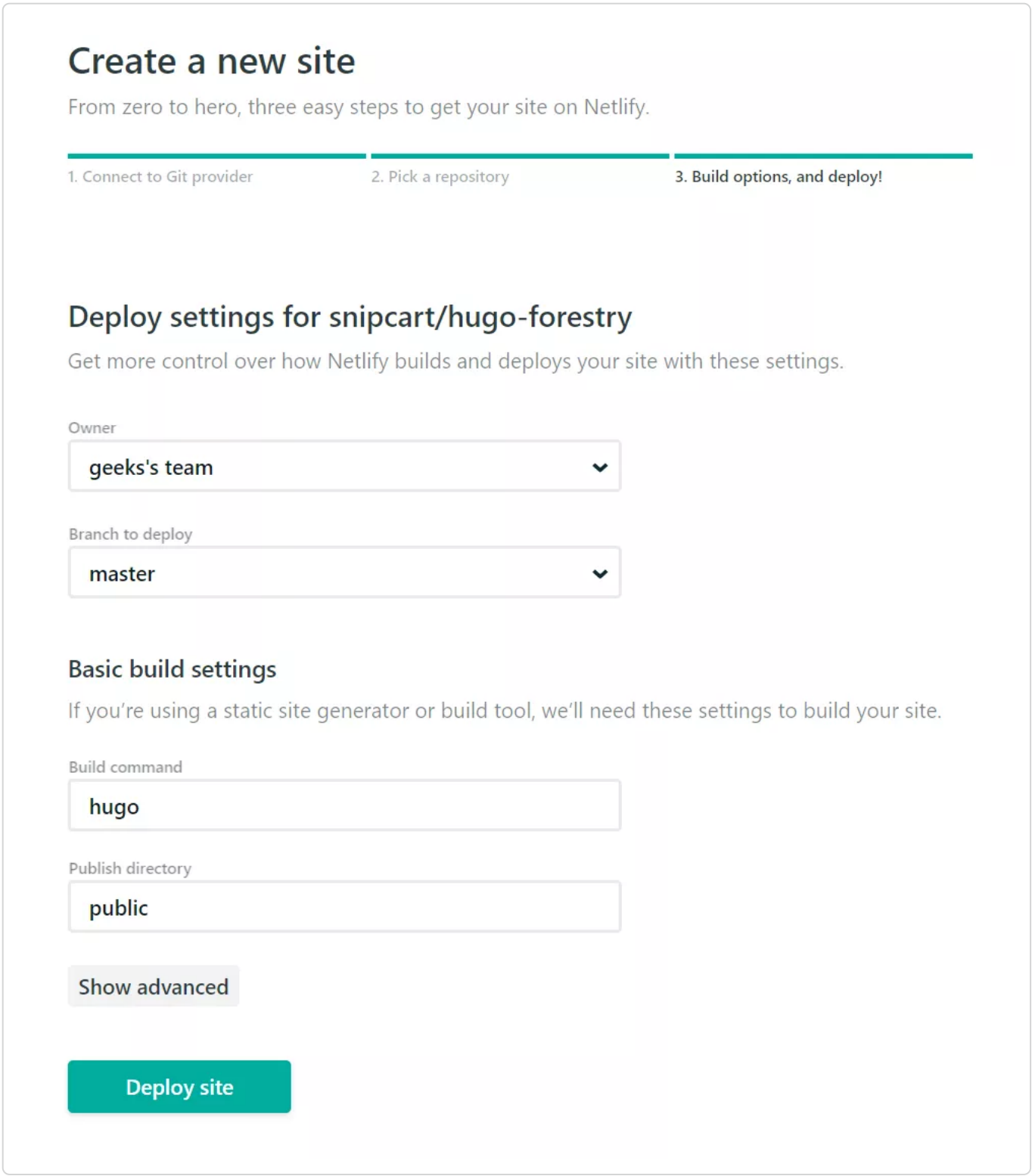
```
{{ range where .Site.RegularPages "Section" "products"}}
  {{ partial "product.html" .Params }}
{{ end }}
```

This loop will find any pages related to the `products` section you've created earlier in Forestry's dashboard. You then pass the `.Params` variable to our partial as this is where the front matter is stored.

8. Deploying the website with Netlify

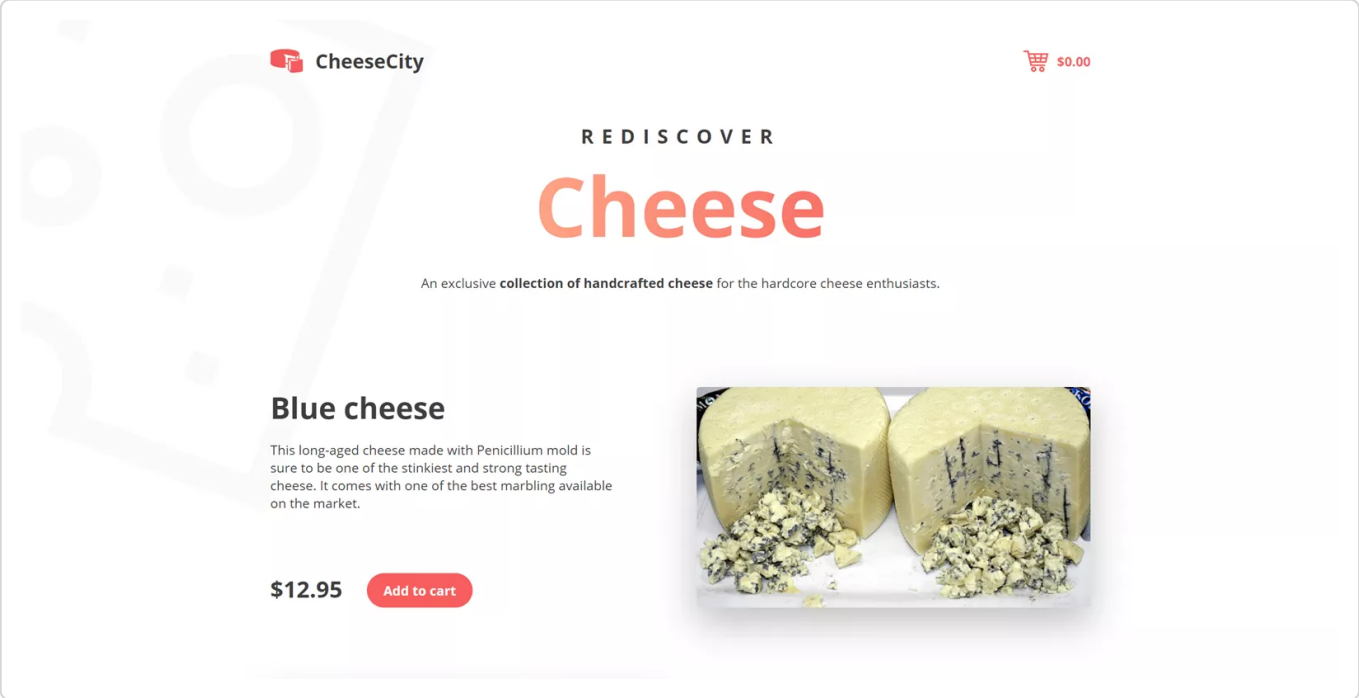
Finally, you'll deploy your site using Netlify. Thankfully, doing so is as easy as logging in and linking your repository.

Usually, Netlify will detect that you're creating a Hugo project and set up the following configuration automatically:



Simply press deploy, and you should be done!

Live demo & GitHub repo



> *See the Github repo here*

> *See live demo here*

Closing thoughts

Being relatively new to the Snipcart team, I've never had the occasion to try a static site generator before. I'm glad I had to this opportunity to work with Hugo. It's super easy to get started as it requires minimal setup and it's incredibly fast! Making changes was nearly instant, and it even had live reloading out of the box. What more could one ask for?

The hardest part was getting an understanding of the content type system, but once I got that "Ah-ha!" moment, everything went as smooth as expected.

It took me a little more than a day to get this demonstration working and integrate it with a CMS. Having said that, if I was to do it again, I'm pretty confident it would only take a few hours—Hugo is that simple to use.

I think Hugo is an excellent fit for a lot of websites. That said, to show how powerful Hugo is, I think it would have been interesting to create a website with multiple pages to fully explore Hugo's template Lookup functionality.

If you've enjoyed this post, please take a second to share it on Twitter. Got comments, questions? Hit the section below!

Michael Poirier-Ginter

FULL STACK DEVELOPER

Michael has been programming for over 4 years and has recently obtained a Web Development college degree in Québec City. A full stack developer, his go-to technologies are React and Go, but he's also efficient with JavaScript, Vue.js, & .NET. Michael's been known to kick serious a** at CS:GO.

Follow him on LinkedIn.

Suggested posts:

19 Comments Snipcart 1 Login

Recommend 2 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

vinamelody • 18 hours ago

Hi, would be nice if I can use Snipcart and Hugo without Forestry for this :)

Reply • Share ›

Mathieu Dionne Mod ➚ vinamelody • 3 hours ago

Hi there, you can use Snipcart with Hugo without Forestry, or any other CMS. Snipcart lives in your site's frontend, no matter the tools you built it with.

Reply • Share ›

Patrick Coleman • 5 months ago

Really good article from Michael, I had the chance to meet him during a conference.

Reply • Share ›

Alex Planting • 5 months ago

I would not use this in an production environment. It is possible to alter the price before adding in to the cart. Not sure if that is desirable. The idea behind it is nice however but how would you solve the issue of price changing by a visitor?

Reply • Share ›

Mathieu Dionne Mod ➚ Alex Planting • 5 months ago

Hi Alex, it's impossible for a user to alter the price of an item before adding it to the cart.

For each transaction, Snipcart crawls back the merchant's website to look for altered information, thus verifying the original information published by the merchant.

All security matters are covered here: https://snipcart.com/security

Thanks for reading!

Reply • Share ›

JasonStoltzfus • 6 months ago

I had to change this line to an uppercased "Data" instead of "data" to get this to work:

```
{{ range .Site.Data.products }}
```

Reply • Share ›

Michael Poirier-Ginter Mod → JasonStoltzfus • 6 months ago

You're right! Thanks for letting us know, we'll update the post for the others to see :)

Reply • Share ›

Francais • 9 months ago • edited

Good day everyone! can you help me, please? i have done like in tutorial and now i want to get custom fields for options of chose,

```
<button class="snipcart-add-item waves-effect waves-light
btn" id="my-button" data-item-id="{{ .id }}" data-item-name="
{{ .name }}" data-item-price="{{ .price }}" data-item-url="{{
.url }}" data-item-custom1-name="{{ .vybor }}" data-item-
custom1-options="{{ .options }}">
shopping_cart
{{ .price }}
</button>



<select id="vybor">

<option>{{ .options }}</option>
```

see more

Reply • Share ›

Charles Ouellet Mod → Francais • 9 months ago

Hey there,

I think you should remove this line:

<option>{{ .options }}</option>

If this doesn't work, I suggest you contact our support team via the Intercom widget in the dashboard!

Reply • Share ›

Saku Mättö • a year ago

I think your syntax is in the wrong order -> hugo site new should read hugo new site

Reply • Share ›

Charles Ouellet Mod ➚ Saku Mättö • 9 months ago

Fixed! Thanks for reporting this.

^ | v • Reply • Share ›

Kamal • 2 years ago

How can I use these codes in content, so each page have different product shown! as this is in template is that mean each product have different template? but actually it should be one template but different content

^ | v • Reply • Share ›

Franck LN Mod ➚ Kamal • 2 years ago

Hey Kamal! Mind contacting our devs through Intercom with precisions? Not sure we fully understand your issue, and they'll be able to help one on one!

1 ^ | v • Reply • Share ›

Argus • 2 years ago

I would like to know how to paginate those products if you have a huge ptoducts.json file

^ | v • Reply • Share ›

Maxime LaBoissonniere ➚ Argus • 2 years ago

Hi there!

First, really sorry for the delay on this your comment slipped through our hands.

There's couple of ways of doing this depending of your use case, if you can poke us directly through Intercom we will happily help you out. :)

Thanks,
Max

^ | v • Reply • Share ›

Bjørn Erik Pedersen • 3 years ago

You don't need getJSON (it is ineffective) if your JSON lives in

COMPANY

- About
- Partnership program
- Terms of service