CheatSheet

Storyblok Nuxt.js

by Lisi Linhart & Samuel Snopko

Setup

Start by creating a Nuxt app and installing the Storyblok Nuxt Package

```
# Create a Nuxt App
npx create-nuxt-app mynuxt

# Install Storyblok Nuxt
npm install storyblok-nuxt --save
```

HINT: Read the 5 minute tutorial on how to add Storyblok to Nuxt.js: www.storyblok.com/tp/headless-cms-nuxtjs

Set up the Storyblok Module

Inside the nuxt.config.js file, we need to add our preview token to the storyblok-nuxt module.

```
modules: [
    ['storyblok-nuxt', {
      accessToken: 'YOUR_PREVIEW_TOKEN',
      cacheProvider: 'memory'
    }]
 ]
```

Make Components Editable

To enable live editing we need to pass the v-editable property to the components that we want to edit inside Storyblok. This property only works on draft versions of the API content.

```
<template>
  <div v-editable="blok" class="teaser">
    <h1>{{blok.headline}}</h1>
  </div>
</template>
```

Getting Content from the API

With Nuxt's asynData function we can fetch our Storyblok data on the server side with the $storyapi object.

```
asyncData (context) {
    return context.app.
$storyapi.get('cdn/stories/home', {
       version: 'draft'
    }).then((res) => {
       return res.data
    })
}
```

Connecting the Storyblok Bridge

We can use the $storybridge object to recognise input events inside of Storyblok. This allows live updating of the content and resolving relations.

```
mounted () {
 this.$storybridge.on('input',
  (event) => {
  if (event.story.id === this.story.id) {
   this.$storybridge.resolveRelations(
    ['featured-articles.articles'],
     (event) => {
      this.story.content =
event.story.content
   })
  }
 })
}
```

HINT: Learn more about the Storyblok Nuxt package on Github: github.com/storyblok/storyblok-nuxt

Storyblok Bridge Publish Event

With the $storybridge it’s also possible to recognize publish events and reload the current path.

```
mounted () {
 this.$storybridge.on(['published',
'change'],
 (event) => {
   // window.location.reload()
   this.$nuxt.$router.go({
     path: this.$nuxt.
$router.currentRoute,
     force: true,
   })
 })
}
```

Getting the Correct Version

Inside of Storyblok we want to load the draft version, but in production we want to load the published version of our content. We can use the context object to find out what version to load.

```
asyncData (context) {
 let version = context.query._storyblok
|| context.isDev ? 'draft' : 'published'

const fullSlug = (context.route.path ==
'/' || context.route.path == '') ?
'home' : context.route.path

 return context.app.$storyapi.get(`cdn/
stories/${fullSlug}`, {
   version: version
 }).then((res) => {
   return res.data
 })
}
```

Creating Dynamic Components

In many cases we want to automatically load the right component depending on the Storyblok structure.

```
<template>
  <component
    v-for="blok in blok.body"
    :key="blok._uid"
    :blok="blok"
    :is="blok.component" />
</template>
```

HINT: Read more about dynamic components in the Vue documentation: vuejs.org/v2/guide/components-dynamic-async.html

In order for that to work we need to register new components in Nuxt. These components need to match the name of the components we created in Storyblok.

```
import Vue from 'vue'
import Page from '~/components/Page.vue'
import Teaser from '~/components/Teaser.vue'
import Grid from '~/components/Grid.vue'
import Feature from '~/components/Feature.vue'

Vue.component('page', Page)
Vue.component('teaser', Teaser)
Vue.component('grid', Grid)
Vue.component('feature', Feature)
```

HINT: Learn how to build a multilanguage website in our tutorial: storyblok.com/tp/nuxt-js-multilanguage-website-tutorial