KERN046

Zeste de savoir

Améliorer son score SEO avec VueJS + NuxtJS

20 mai 2019

Table des matières

En pleine création de mon entreprise, je me suis mis à construire ma plateforme web avec VueJS , qui m'a énormément séduit par sa simplicité. Pour pouvoir faire vivre mon application et l'imposer sur le marché de la reconstitution médiévale, j'ai vite réalisé qu'il me faudrait utiliser une arme plus sophistiquée qu'un trébuchet : le SEO ! Je me suis donc mis en quête des bases et des bonnes pratiques qui me permettraient de courtiser les moteurs de recherche.

Nous allons donc traiter ici différents éléments du SEO : *sitemaps, balises META, données structurées*... et voir comment les mettre en place dans un environnement VueJS. Je ne suis pas un expert en SEO, cet article s'adresse aux personnes qui ont un minimum d'expérience en développement et aimeraient en apprendre plus sur le référencement de leur site. La mise en place des différents éléments se fait dans un environnement VueJS, mais les concepts évoqués s'appliquent bien entendu à tout projet web, quelles que soient les technologies utilisées !

1. Single-Page Application, le faux-ami du SEO

Ces dernières années, les *frameworks* Javascript à la tête du marché sont pour la plupart destinés à construire des *Single-Page Applications* (SPA). La différence avec un site web classique est que la navigation d'une page à une autre se fait sans chargement, c'est-à-dire que l'application ne charge que les éléments nouveaux de la page vers laquelle vous vous dirigez, sans nécessiter un rechargement complet de votre navigateur. Les éléments communs à l'ensemble des pages, comme les menus et les pieds de page, restent chargés pendant toute la navigation. C'est comme si votre application web n'avait qu'une seule et unique page.

Cela a de nombreux avantages sur lesquels nous ne passerons que très rapidement, mais principalement vous imaginez bien le gain de performances qu'il y a à ne pas devoir tout recharger à chaque page. À une époque où chaque dixième de seconde compte pour garder ses utilisateurs et un bon *ranking* SEO, chacun a intérêt à y prêter attention.

Cependant, quand vient le temps de se soucier de son référencement, on réalise alors qu'on est tombé dans un véritable traquenard : eh oui, si votre site se comporte comme s'il n'avait qu'une page, comment voulez-vous que les moteurs de recherche puissent correctement indexer vos produits, vos articles ? En effet, si un produit se situe à l'URL `/produits/mon-super-produit`, le moteur de recherche va charger le HTML de votre page. Dans une application VueJS, il n'y a qu'un seul fichier HTML minimaliste, qui permet seulement de charger le Javascript de votre application qui sert à générer dynamiquement son code HTML. Le moteur de recherche ne chargeant pas Javascript (apparemment Google serait capable de le faire, mais d'après mes

tests pas vraiment), il ne disposera d'aucune balise META permettant d'indexer correctement la page `/produits/mon-super-produit`.

Mais ne vous enfuyez pas, il existe une solution ! Le *Server-Side Rendering* (SSR) ! Cela permet de générer à l'avance le HTML final de vos pages, et donc vos balises META. Quand `/produits/mon-super-produit` sera appelé, le serveur répondra avec les métadonnées correspondant à votre produit, qui sera correctement indexé ! Sauvés !

Pour faire du SSR avec VueJS, nous allons utiliser NuxtJS , un *framework* complet avec de nombreux outils !

> *i* Il existe des solutions pour se passer de SSR, par exemple cette implémentation , qui fait du «*pre-rendering*», aussi appelé «*Static Site Generation*». Je vous laisse la découvrir et vous faire votre propre avis, mais dans le cadre de cet article nous en resterons à une approche orientée SSR.

2. Un sitemap pour les indexer tous

Une fois Nuxt installé, on va pouvoir s'intéresser à la première étape de notre quête vers le référencement : **le *sitemap*** ! Comme son nom l'indique, il s'agit d'une cartographie des pages de votre site, afin d'indiquer facilement aux moteurs plusieurs informations sur ce qu'ils doivent indexer. Il va d'abord fournir bien évidemment l'URL de chaque page, mais peut également indiquer la date de dernière modification de la page ou la priorité d'indexation. Tous ne sont pas utilisés par l'ensemble des moteurs de recherche (typiquement, Google n'utilise pas le champ *priority*). Vous pouvez également donner des informations sur des images ou des vidéos contenues dans votre page pour étoffer les résultats indexés.

Maintenant, au travail !

Tout d'abord, nous allons installer deux modules pour générer et diffuser facilement notre *sitemap* : @nuxtjs/sitemap et @nuxtjs/robots ! Configurons tout d'abord notre module `sitemap` dans notre fichier `nuxt.config.js` :

```
1  import { getRoutes } from './helpers/sitemap';
2
3  module.exports = {
4    mode: 'universal',
5    // ...
6
7    modules: [
8      '@nuxtjs/sitemap'
9    ],
10
11   sitemap: {
12     path: '/sitemap.xml', // L'emplacement de votre fichier
           sitemap.
```

```
    hostname: process.env.WEBSITE_URL, // L'adresse de votre site, que vous pouvez placer comme ici dans une variable d'environnement.
    cacheTime: 1000 * 60 * 15, // La durée avant que le sitemap soit regénéré. Ici 15mn.
    gzip: true,
    generate: false, // Génère une version statique du sitemap quand activé. À utiliser avec nuxt generate.
    exclude: [ // Les pages qu'on a pas trop envie de voir atterrir sur Google.
      '/login',
      '/admin/**'
    ],
    routes() {
      // Nous allons utiliser une fonction personnalisée pour charger nos routes dynamiques dans le sitemap.
      return getRoutes();
    }
  }
};
```

Cette configuration de base, sans même implémenter la fonction `routes()`, générera un *sitemap* basique avec des entrées correspondant aux pages de votre application Nuxt. L'implémentation de la fonction `routes()` permet d'y ajouter les pages dynamiques de votre application. Dans mon cas, il s'agit d'événements répertoriés sur mon site. Pour pouvoir les indexer, implémentons donc la fonction `getRoutes()`.

```
// ./helpers/sitemap.js
import config from '../config';
import fetch from 'isomorphic-fetch';
import h2p from 'html2plaintext';

export const getRoutes = () => {
    // Attention, cette fonction DOIT retourner une Promise.
    return new Promise(async (resolve, reject) => {
        // Je récupère les événements depuis mon API.
        const events = await fetchEvents();
        const routes = [];

        for (const event of events) {
            // Pour chaque événement, je renseigne les informations indexées à partir des données que j'ai.
            const route = {
                url: `/events/${event.slug}`,
                lastmodISO: event.updated_at,
                priority: 1
            };
```

```
20              // Spécificité liée à mon application, l'événement n'a
                    pas forcément sa propre image.
21              if (event.thumbnail !== null) {
22                  // En revanche, s'il en a une, j'en indexe les
                        informations.
23                  route.img = [
24                      {
25                          // La configuration de mon hostname ainsi
                                que mon architecture pour distribuer
                                les images sont très spécifiques à mon
                                infrastructure, il est évidemment
                                possible de simplifier.
26                          url:
                                `${config.apiUrl}:${config.apiPort}/pictures/${even
27                          caption: h2p(event.short_description), //
                                h2p me vient d'un module retirant les
                                balises HTML.
28                          title: event.label,
29                          geoLocation: `${event.address.town},
                                ${event.address.zip_code.country}`
30                      }
31                  ];
32              }
33              routes.push(route);
34          }
35          // Tout se passe bien, je résous ma Promise en renvoyant
                les routes ajoutées par ma fonction.
36          resolve(routes);
37      });
38  };
39
40  const fetchEvents = () => {
41      return fetch(`${config.apiUrl}/events`).then(response =>
            response.json());
42  };
```

Vous pouvez désormais lancer votre serveur Nuxt ! Une fois la compilation de vos sources terminée et le serveur en état d'écoute, vous pouvez vous rendre dans votre navigateur, sur l'URL `/sitemap.xml` de votre site.

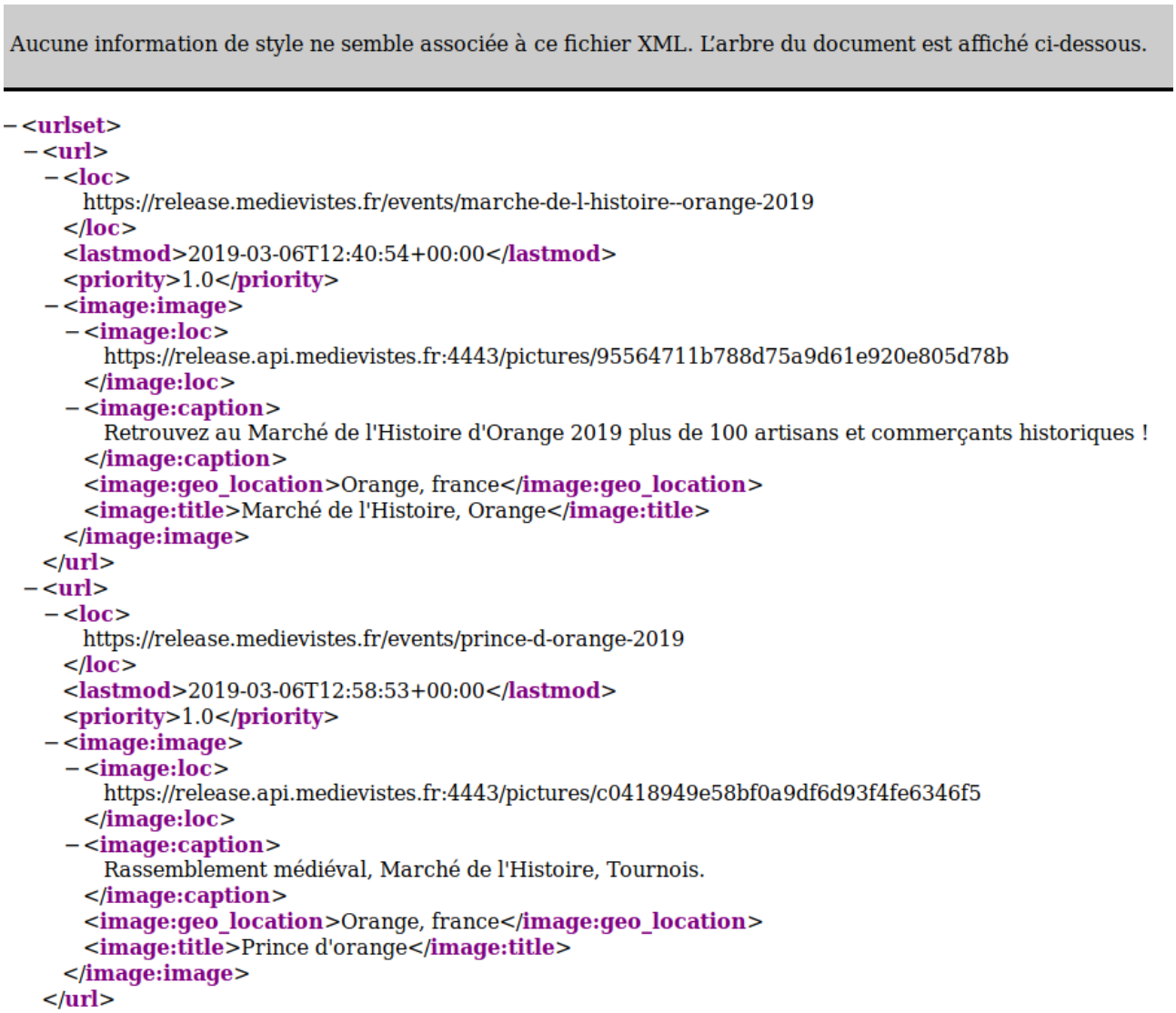

FIGURE 2. – Ça commence à ressembler à quelque chose !

Ça a l'air plutôt pas mal ! Il reste néanmoins un détail à régler : les moteurs de recherche ne prennent pas en compte automatiquement votre fichier `sitemap.xml`. En effet, il existe des cas où vous pourriez avoir un nom différent, ou même scindé votre *sitemap* en plusieurs fichiers distincts. Certains moteurs, comme Google, vous permettent d'ajouter votre fichier *sitemap* manuellement depuis une console d'administration. Mais nous souhaitons garder une démarche plus simple, qui permettra à chaque moteur de récupérer notre fichier. C'est là qu'entre en scène le second module que nous avons installé : @nuxtjs/robots !

Nous allons très simplement le configurer depuis notre fichier `nuxt.config.js` :

```
1 module.exports = {
2   mode: 'universal',
3
4   // ...
5
6   modules: [,
```

```
    '@nuxtjs/sitemap',
    '@nuxtjs/robots'
  ],

  robots: {
    Disallow: [
      '/login',
      '/admin',
    ],
    Sitemap: `${process.env.WEBSITE_URL}/sitemap.xml`
  }
}
```

Une fois le build Nuxt passé, vous pouvez vous rendre à l'URL `/robots.txt`. Vous devriez obtenir quelque chose de similaire à l'image suivante.

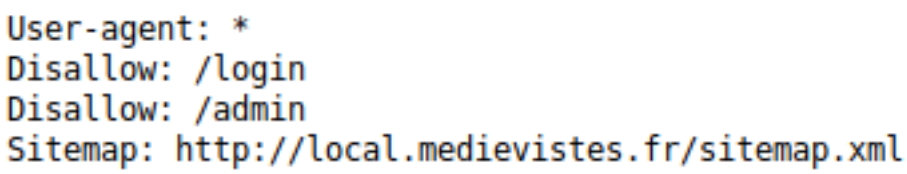

FIGURE 2. – Simple et efficace !

Nous avons maintenant un fichier *sitemap* correctement généré et distribué aux moteurs de recherche ! Mais la route est encore longue vers la gloire et la page 1 de Google !

3. Les métadonnées, indispensables pour bien identifier vos pages !

Maintenant, nous avons indiqué les URL de nos pages, avec quelques informations utiles à l'indexation, grâce à notre *sitemap*. Il est temps désormais de personnaliser les résultats que vont renvoyer les moteurs de recherche. Pour cela, nous allons recourir aux métadonnées.

Kézako ?

Les métadonnées sont des informations relatives au document HTML que votre serveur renvoie au navigateur, et ont plusieurs utilités ! Une métadonnée de base des pages web est l'encodage des caractères, qui est la plupart du temps l'UTF-8. Ou encore le *Content-Type* d'un contenu chargé par votre navigateur (est-ce un script JS ? Une feuille de style CSS ?). Il existe énormément de métadonnées avec plusieurs usages. Nous allons nous concentrer sur celles permettant d'améliorer les résultats des moteurs de recherche sur notre site.

Nuxt nous facilite la tâche pour définir des métadonnées et les personnaliser page par page. C'est assez logique par ailleurs, car rappelez-vous, c'est précisément pour cela que nous nous sommes

orientés vers du *Server-Side Rendering*. Avec seulement VueJS, il nous aurait été ~~impossible~~ *difficile* de personnaliser les métadonnées d'une page précise.

Nous allons procéder en deux étapes : tout d'abord, nous allons définir des métadonnées générales dans notre fichier `nuxt.config.js`, en y implémentant la fonction `head()`, avant de personnaliser ces valeurs pour chacune de nos pages. Voici pour le fichier `nuxt.config.js` :

```
module.exports = {
  mode: 'universal',

  head: {
    title: 'Médiévistes',
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content:
          'width=device-width, initial-scale=1' },
      { hid: 'description', name: 'description', content:
          'Portail de la communauté médiévale française' },
      { hid: 'og:type', name: 'og:type', content: 'website' },
      { hid: 'og:url', name: 'og:url', content:
          process.env.WEBSITE_URL },
      { hid: 'og:title', name: 'og:title', content: 'Médiévistes'
          },
      { hid: 'og:site_name', name: 'og:site_name', content:
          'Médiévistes' },
      { hid: 'og:locale', name: 'og:locale', content: 'fr' },
      { hid: 'og:image', name: 'og:image', content:
          `${process.env.WEBSITE_URL}/Logo_Red_BG_512px.png` },
      { hid: 'google-site-verification', name:
          'google-site-verification',  content:
          process.env.GOOGLE_SITE_VERIFICATION_TOKEN },
      { name: 'msapplication-TileColor', content: '#2b5797' },
      { name: 'theme-color', content: '#fdf8f0' }
    ],
  }

  // ...
}
```

Les métadonnées principales ici sont `description` ainsi que l'ensemble des balises préfixées par `og:` (métadonnées basées sur la technologie Open-Graph développée par Facebook, consultez cet article pour en savoir plus). Il est possible d'en ajouter d'autres, comme les Twitter Cards . Un attribut à retenir de cette configuration dans Nuxt est le `hid` de chaque métadonnée : c'est ce qui va nous permettre d'écraser ces valeurs dans nos pages.

Prenons maintenant la page d'accueil de notre site, dans `./pages/index.vue` et implémentons-y également la méthode `head()`.

```
export default {
    name: 'homepage',

    // ...

    head() {
        return {
            // Le module i18n permet de traduire les valeurs des
                métadonnées selon la locale de l'utilisateur
            title: this.$i18n.t('pages.homepage.title'),
            meta: [
                {
                    'property': 'og:title',
                    'content':
                        this.$i18n.t('pages.homepage.title'),
                    'hid': 'og:title'
                },
                {
                    'property': 'og:description',
                    'content':
                        this.$i18n.t('pages.homepage.description'),
                    'hid': 'og:description'
                },
            ]
        };
    }
};
```

Ici, je donne donc un titre à ma page et j'écrase notamment les valeurs par défaut des métadonnées `og:title` et `og:description`. Voici un second exemple, qui utilise les données de l'application pour générer des métadonnées pertinentes.

```
import h2p from 'html2plaintext';

export default {
    name: 'page-event-details',

    head() {
        const meta = [
            {
                'property': 'og:title',
                'content': this.event.label,
                'hid': 'og:title'
            },
            {
                'property': 'og:description',
                'content': h2p(this.event.short_description),
```

```
16                  'hid': 'og:description'
17              },
18              {
19                  'property': 'description',
20                  'content': h2p(this.event.short_description),
21                  'hid': 'description'
22              },
23          ];
24          if (this.event.thumbnail !== null) {
25              meta.push({
26                  'property': 'og:image',
27                  'content':
                        `${this.$store.getters['api/urlWithPort']}/pictures/${this
28                  'hid': 'og:image'
29              });
30          }
31          return {
32              title: this.event.label,
33              meta
34          };
35      },
36
37      async asyncData({ app, params }) {
38          return {
39              // Récupère les données de l'événement de la page dans
                    l'API.
40              event: await app.$repositories.event.get(params.slug)
41          };
42      }
43  }
```

Grâce à ces métadonnées, j'obtiens l'affichage suivant en partageant mon lien sur Discord :

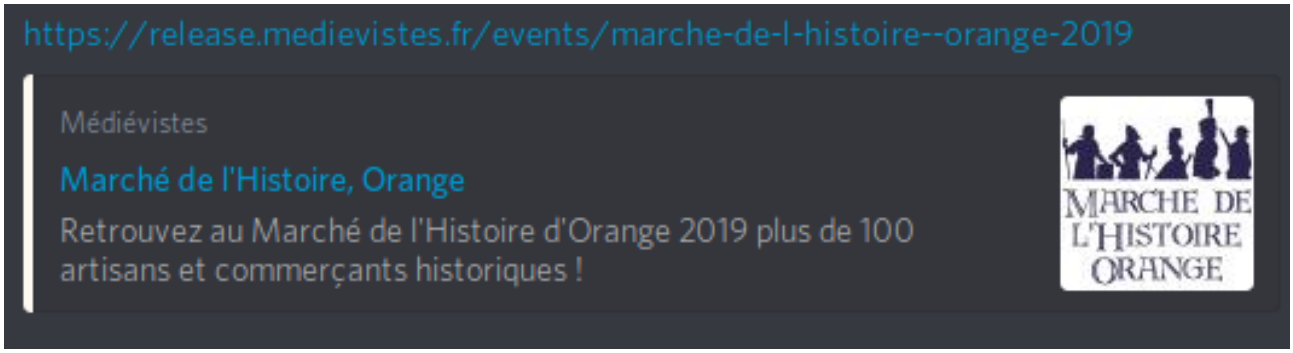

FIGURE 3. – Je sais pas pour vous, mais moi j'étais content

Voilà pour les métadonnées ! Évidemment je n'ai fait que survoler le sujet, je vous laisse approfondir grâce à des auteurs plus spécialisés et à vos propres expérimentations !

4. Les données structurées

Pour rédiger ce petit article sur le SEO et VueJS, j'ai bien évidemment voulu me renseigner davantage sur les techniques SEO pour pouvoir vous livrer un contenu un poil approfondi. Je suis rapidement tombé face à face avec la notion de *données structurées*, qui m'a semblé être l'arme ultime pour parfaire le référencement naturel de mon application.

Bon ben dis-nous ce que c'est !?

Sur une page de votre application, plusieurs éléments différents peuvent être présents : des blocs de texte, des images, des vidéos... Avec les données structurées, vous allez pouvoir préciser ce que représentent ces éléments en termes de données. Ce bloc de texte est-il une adresse postale ou le nom d'un acteur connu ? Cette image est-elle une photo de profil ou l'illustration d'un événement ?

A quoi ça sert ? Eh bien avec ça, vous allez pouvoir présenter sur la page de résultats des moteurs de recherche des extraits enrichis (*rich snippets*) en plus des données classiques (titre, URL, description).

Microdata ou JSON-LD ?

Je savais que vous alliez me poser la question ! Ce sont deux moyens distincts de représenter des données structurées. Ils font la même chose. Voyons leurs différences.

Microdata : il s'agit d'attributs HTML avec lesquels on décrit le contenu de balises. Exemple :

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1 itemprop="name">Avatar</h1>
  <span>Director: <span itemprop="director">James Cameron</span>
      (born August 16, 1954)</span>
  <span itemprop="genre">Science fiction</span>
  <a href="../movies/avatar-theatrical-trailer.html"
      itemprop="trailer">Trailer</a>
</div>
```

Les attributs *itemscope*, *itemtype* et *itemprop* servent à indiquer ce qu'est la donnée affichée par ce balisage HTML : il s'agit d'un film, dont on précise le nom, le directeur, le genre et la bande-annonce.

JSON-LD : c'est le format aujourd'hui recommandé par Google et le W3C. Sans trop de surprises, c'est du JSON inclus dans l'entête de la page au sein d'un script JS. La principale différence est que les données ne sont pas schématisées dans le document HTML et sont centralisées dans l'en-tête.

Mais alors, lequel choisir ?

En vérité, malgré le fait que les *microdata* soient désignées comme ancien format et que Google recommande JSON-LD, je n'ai pas trouvé ce choix si évident que ça dans notre cas. Il aurait pu être intéressant, au vu de l'approche **Component** de VueJS, d'utiliser les *microdata* pour qu'un composant ait son marquage *microdata* quel que soit l'endroit où il est utilisé. Mais il est

aussi possible que l'on veuille personnaliser page par page les données enrichies que l'on veut faire ressortir du lot.

JSON-LD ne m'a pas l'air limpide dans sa gestion des différents schémas et la documentation semble, au premier coup d'œil, souvent mêlée aux documents de spécification nécessaires pour le standardiser, ce qui la rend assez indigeste. Cependant, pour pouvoir adopter une approche page par page, nous allons nous concentrer sur JSON-LD. Je vous invite à vous faire votre propre avis sur les *microdata* et à consulter le site schema.org .

Intégrons donc JSON-LD dans nos pages !

En l'occurrence, je vais tenter d'intégrer un *rich snippet* pour la page de détail d'un événement. Il est important de consulter la documentation apportée par Google (que je trouve bien plus claire et intuitive que le site officiel de JSON-LD d'ailleurs), notamment la documentation du type Event ainsi que le guide qualité des données structurées .

```
module.exports = {
    name: 'page-event',

    head() {
        const meta = [
            {
                'property': 'og:title',
                'content': this.event.label,
                'hid': 'og:title'
            },
            {
                'property': 'og:description',
                'content': h2p(this.event.short_description),
                'hid': 'og:description'
            },
            {
                'property': 'description',
                'content': h2p(this.event.short_description),
                'hid': 'description'
            },
        ];
        if (this.event.thumbnail !== null) {
            meta.push({
                'property': 'og:image',
                'content':
                    `${this.$store.getters['api/urlWithPort']}/pictures/${this
                'hid': 'og:image'
            });
        }
        return {
            __dangerouslyDisableSanitizers: ['script'],
            script: [{ innerHTML:
                JSON.stringify(this.structuredData), type:
                'application/ld+json' }],
```

```
            title: this.event.label,
            meta
        };
    },

    async asyncData({ app, params }) {
        const event = await
            app.$repositories.event.get(params.slug);

        return {
            event,
            structuredData: {
                "@context": "http://schema.org",
                "@type": "Event",
                "name": event.label,
                "description": event.description,
                "startDate": event.begin_at,
                "endDate": event.end_at,
                "location": {
                    "@type": "Place",
                    "address": {
                        "@type": "PostalAddress",
                        "streetAddress": event.address.street_name,
                        "postalCode": event.address.zip_code.code,
                        "addressLocality": event.address.town,
                        "addressCountry":
                            app.i18n.t(`location.countries.codes.${event.addres
                    }
                }
            }
        };
    }
};
```

Et voilà le travail ! Plus qu'à laisser le temps aux moteurs de recherche d'indexer tout cela et nous devrions voir les résultats apparaître avec les nouvelles données !

Des outils peuvent vous aider à valider vos schémas JSON-LD, par exemple cet outil en ligne de Google , que vous pouvez même utiliser en local avec Localtunnel (je n'ai pas essayé ce dernier personnellement). Vous voici armés de pied en cap !

Et voilà, nous en avons fini pour cet article ! Évidemment le spectre du SEO est bien plus large, et pour approfondir, de nombreux sujets peuvent être épluchés sur les sites spécialisés. Par exemple, pour les mobiles, les pages AMP semblent être un bon moyen de favoriser son référencement naturel.

Nous avons pu voir que Nuxt permettait une manipulation relativement souple des différentes techniques d'enrichissement du contenu dans un environnement VueJS, et permet de bien mettre en avant les données contenues dans nos applications !

J'espère que cet article vous a intéressé, en tout cas ce fut un véritable plaisir de le rédiger. N'hésitez pas à faire vos retours !

Un grand merci aux personnes ayant relu l'article durant la bêta, et en particulier à @Taurre qui s'est démené pour trouver un validateur, ainsi qu'à @Heziode qui a bien voulu remplir cet office !