Présentation Gwen – Wordpress Headless

Présentation générale :

Wordpress headless est une version du CMS Wordpress qui aide les développeurs web à travailler plus efficacement sur leurs sites Wordpress. C'est une configuration spéciale utilisée uniquement pour la gestion de contenu backend tandis que la gestion du contenu, c'est-à-dire la partie front end est gérée via une autre technologie, souvent un framework Javascript comme React, Vue ou Angular.

Pour accéder aux contenus, nous allons utiliser l'API REST de Wordpress qui se base sur un système de routage pour accéder aux différents contenus. La structure de l'API REST est facilement consultable sur la documentation officielle ou à la racine d'un projet wordpress via l'extension « wp-json » :

- ➔ https://developer.wordpress.org/rest-api/
- ➔ http://localhost:9000/wp-json

Le principe est simple, en appelant un « endpoint » via une requête HTTP, le serveur va nous renvoyer une réponse en JSON des données dont nous souhaitons disposer. Une API REST accepte tout un tas de paramètres permettant de filtrer la requête en limitant le nombre de réponses par exemple dans le cas où on souhaiterait obtenir tel nombre de réponses ou encore les trier dans l'ordre ascendant / descendant.

Comment utiliser Wordpress Headless :

Il existe deux solutions possibles :

- ➔ Via framework Javascript : React, Vue, Angular voir même Astro et utilisation de l'API REST.
- ➔ Plugin : GraphQL et utilisation du GraphQL.

Comment installer Wordpress Headless :

- ➔ Créer un theme vide avec les deux propriétés « theme name » et « author » dans un fichier style.css puis insérer le code suivant dans un fichier php qui permet une redirection vers notre hébergement front-end :

```
<script type="text/javascript">
    window.location = 'http://localhost:3000'; //On effectue une redirection sur le serveur front-end.
</script>
```

Remarque : Il est toujours possible d'accéder au panel admin pour modifier les contenus du site via l'éditeur Wysiwyg. L'accès doit se faire directement à l'URL de connexion sans passer par la racine du projet. Dans ce cas la redirection est effectuée immédiatement.

- ➔ Installer le Headless Mode Plugin (peu recommandé) : https://wordpress.org/plugins/headless-mode/

Avantages :

- ➔ Ainsi Wordpress Headless apporte une certaine flexibilité au niveau du front-end avec la possibilité d'utiliser des technologies modernes et cela permet également de séparer d'une part tout ce qui touche au front-end et d'une autre part tout ce qui touche au back-end. Ce qui permet à une équipe d'avancer en parallèle sur des tâches liées d'une part au front-end et au back-end.
- ➔ Gain de performance avec une architecture plus légère et plus moderne
- ➔ Scalabilité & interopérabilité : Permet des calls via API plus simples à mettre en place.
- ➔ Sécurité : exposition limitée : Le serveur WordPress n'est pas directement exposé à l'utilisateur final, réduisant les risques d'attaques par la limitation naturelle des points d'accès au back-end.

Désavantages :

- ➔ Complexité accrue dans la mise en place de l'outil, il faut maîtriser à la fois l'API REST de Wordpress tout en maîtrisant un framework Javascript.
- ➔ Nécessite des configurations spécifiques du point de vue du SEO.
- ➔ Les plugins, thèmes ne fonctionnent pas en Wordpress Headless et doivent être implémentés d'une autre manière…
- ➔ Prévoir un coût supplémentaire dans la mise en place de Wordpress Headless puisqu'il est nécessaire de disposer de deux hébergements, un gérant le front-end et un gérant le back-end.

Wordpress Headless a un intérêt concret dans le développement des SPA « Single Page Application » mais peut être étendu aux MPA et sites web de plus grande envergure.

Principe d'utilisation :

- ➔ **Démarrer deux hébergements :**
  - **Un back-end avec Wordpress** headless activé (se référer à la section « Comment installer Wordpress headless »
  - **Un front-end avec le framework que l'on souhaite** (VueJS et Nuxt son framework le plus connu afin de gérer l'interopérabilité avec le back-end Wordpress)
- ➔ Une fois cela fait notre back-end wordpress sera à même de répondre aux requêtes effectuées via l'API REST consultable via la documentation ou à la racine du back-end wordpress en spécifiant l'extension « wp-json ».

Pourquoi utiliser Nuxt complémentairement à VueJS ?

Problème du rendu côté client (Client-Side Rendering - CSR) :

- ➔ Problème : Vue.js rend initialement les pages côté client, ce qui signifie que le contenu n'est pas présent dans le HTML initial envoyé par le serveur. Les moteurs de recherche peuvent avoir des difficultés à indexer correctement les pages, car ils doivent exécuter le JavaScript pour voir le contenu.

➔ Solution : Utiliser le rendu côté serveur (Server-Side Rendering - SSR) avec **Nuxt.js** (framework basé sur Vue.js), qui permet de pré-rendre les pages sur le serveur avant de les envoyer au client.

Problématique SEO :

Par défaut le **CSR** (Client Side Rendering) de Vue peut nuire au SEO. Effectivement et dans ce schéma, le HYML initial rendu est minimal, le contenu de l'application complet est rendu dynamiquement côté client via Javascript. Pour se faire, Nuxt a prévu des optimisations SEO conséquentes en plus d'un mode de rendering différent que nous verrons dans la suite de ce tutoriel : « l'universal rendering ».

➔ **Cas d'un onepage :**

- Dans le fichier « nuxt.config.ts », il est possible de définir les propriétés dans le head en créant un objet app, comprenant une propriété head comprenant elle-même les différentes balises types HTML homonymes :

nuxt.config.ts

```
export default defineNuxtConfig({
  app: {
    head: {
      charset: 'utf-8',
      viewport: 'width=device-width, initial-scale=1',
    }
  }
})
```

➔ **Cas d'un site avec plusieurs pages donc plusieurs components :**

- On va utiliser la fonction useHead en haut de chaque component vue qui prend en paramètres les différentes propriétés HTML homonymes. Seul est à noter pour exception la balise meta qui accepte un tableau avec name et content. Il est également possible de charger un script via la propriété script. C'est la méthode recommandée pour importer des librairies par CDN.

app.vue

```
<script setup lang="ts">
useHead({
  title: 'My App',
  meta: [
    { name: 'description', content: 'My amazing site.' }
  ],
  bodyAttrs: {
    class: 'test'
  },
  script: [ { innerHTML: 'console.log(\'Hello world\')' } ]
})
</script>
```

-

➔

➔ **Pour aller plus loin :**

- Si l'on souhaite aller plus loin, Nuxt propose également la fonction useSeoMeta qui permet de définir manuellement pour chaque composant (page) des balises SEO Meta personnalisés :

```
<script setup lang="ts">
useSeoMeta({
  title: 'My Amazing Site',
  ogTitle: 'My Amazing Site',
  description: 'This is my amazing site, let me tell you all about it.',
  ogDescription: 'This is my amazing site, let me tell you all about it.',
  ogImage: 'https://example.com/image.png',
  twitterCard: 'summary_large_image',
})
</script>
```

Il est même possible de rendre ces différentes balises SEO dynamiques à l'aide des références dynamiques :

app.vue

```
<script setup lang="ts">
const title = ref('Hello World')
</script>

<template>
  <div>
    <Head>
      <Title>{{ title }}</Title>
      <Meta name="description" :content="title" />
      <Style type="text/css" children="body { background-color: green; }" ></Style>
    </Head>

    <h1>{{ title }}</h1>
  </div>
</template>
```

Récupération de données avec Nuxt (intéressant pour l'API REST de WP) :

En vanilla Javascript, on utilise la célèbre fonction fetch avec une url ou une source de données, ce qui renvoit une promesse et permet de traiter une réponse à laquelle on souhaite accéder. Nuxt facilite le processus de récupération de données et utilise trois méthodes possibles de récupération de données :

➔ **Fonction useFetch :** La fonction useFetch est une solution simple permettant de récupérer des données. C'est littéralement un appel à la fonction fetch de Javascript :

app.vue

```vue
<script setup lang="ts">
const { data: count } = await useFetch('/api/count')
</script>

<template>
  <p>Page visits: {{ count }}</p>
</template>
```

➔ **$fetch,** $fetch est une autre solution permettant de récupérer des données. C'est un appel direct à la librairie « oFetch » (https://github.com/unjs/ofetch) inclue dans Nuxt qui permet des appels plus peformants aux API dont on a besoin. Cette méthode s'apparente aux appels AJAX et permettent de spécifier une method dans la requête définie, ce qui peut être intéressant dans le cadre de notre modèle Wordpress Headless reposant sur l'API REST de ce dernier.

pages/todos.vue

```vue
<script setup lang="ts">
async function addTodo() {
  const todo = await $fetch('/api/todos', {
    method: 'POST',
    body: {
      // My todo data
    }
  })
}
</script>
```

➔ **La fonction useAsyncData()**, plus compliquée à comprendre mais permet d'appeler de manière asynchrones des données. Plus difficile à comprendre mais permet d'appliquer une logique asynchrone et de retourner des données une fois que la promesse a été résolue. Cette fonction permet donc une meilleure gestion des erreurs. Voir doc : (https://nuxt.com/docs/getting-started/data-fetching)

pages/users.vue

```vue
<script setup lang="ts">
const { data, error } = await useAsyncData('users', () => myGetFunction('users'))

// This is also possible:
const { data, error } = await useAsyncData(() => myGetFunction('users'))
</script>
```

Au moment de la mise en production :

Un projet vue demande à être buildé via la commande « npm run build ». Le contenu « buildé » se trouve dans le répertoire « dist » et est prêt à être mis en production sur l'hébergement dédié à la partie front-end.

Nuxt et le « universal rendering » :

Nuxt par défaut utilise le universal rendering qui est un mélange du client-side rendering et du server-side rendering, ce qui combine l'avantage des deux à la fois. Parmi les avantages nous retrouvons une performance augmentée et un contenu SEO-friendly puisque le HTML est livré au navigateur tel quel, les crawlers peuvent ainsi facilement indexer le contenu. Toutefois cela a certains désavantages notamment du point de vue du coût même si l'universal rendering tend à réduire les requêtes au serveur. Il est possible toutefois d'opter pour d'autres techniques plus spécifiques de rendering comme le « edge-side-rendering » : https://nuxt.com/docs/guide/concepts/rendering#edge-side-rendering

Le principe du universal rendering est ainsi le suivant : quand le navigateur envoi une requête au serveur, le serveur renvoi au client une page HTML pré-rendu. Par la suite vient la dernière phase « d'hydratation » où le client télécharge le javascript vue. De là, l'application est 100% interactive et prend compte des différentes directives de vue (affichage conditionnel, boucles, syntaxe moustache etc.)

Dans certains cas, nous souhaitons que certains morceaux de notre code soit rendu côté client uniquement. Pour cela, nous englobons notre code dans une balise <client-only><client-only>. Le contenu à l'intérieur des balises <client-only><client-only/> n'est ainsi pas rendu et ne fait pas parti du HTML envoyé par le serveur au navigateur. Lorsque le navigateur reçoit le HTML et exécute le JS de Vue et Nuxt, le contenu à l'intérieur de ces balises est rendu dynamiquement. Cela permet de montrer le contenu qu'une fois que le chargement client est terminé, une fois que tout le javascript de vue et nuxt a été téléchargé.

Question de coût et d'hébergement.

L'architecture Wordpress Headless pose une question de coûts puisqu'il est nécessaire de déployer deux hébergement, un dédié au front-end géré par notre framework vueJS et Nuxt et un second dédié au back-end géré par Wordpress directement.

Comme dit précédemment, Wordpress Headless nécessite également des performances accrues par rapport à tous les appels vers l'API REST pouvant être générés. De plus, l'ajout de Nuxt pour l'optimisation SEO et tout ce que rajoute Nuxt en fonctionnalités utilise le SSR. **Il est ainsi nécessaire d'avoir un serveur capable de faire tourner une architecture node.js.**

Voici un exemple d'hébergement de serveur nodejs chez OVH :
https://www.ovhcloud.com/fr/web-hosting/nodejs-hosting/

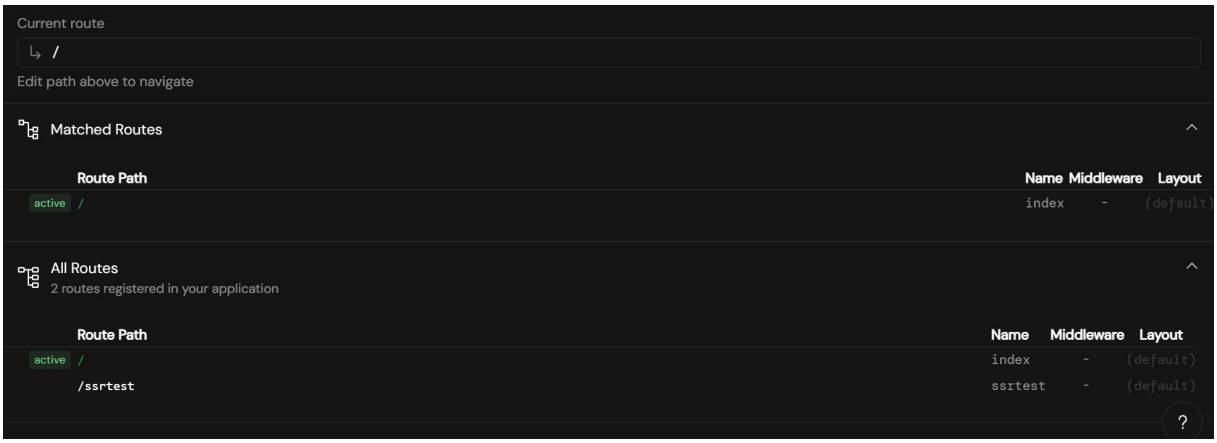
Extensions Nuxt :

Pour rajouter des fonctionnalités à Nuxt, il est possible d'ajouter des extensions appelés sur le site « modules » qui permettent d'enrichir l'expérience développeur. Il existe une exnteions tailwind CSS qui permet d'ajouter directement le framework CSS au fonctionnement du framework Vue. Cela évite ainsi de passer par une installation manuelle via NPM ou via CDN.

Toutes ces extensions sont là pour se faciliter la vie des développeurs qui souhaitent disposer d'outils de développement leur permettant de développer des applications web plus facilement.

Nuxt devtools :

Toujours dans un but de faciliter les processus de développement Nuxt propose « nuxt devtools » qui est une extension nativement installée sur Nuxt et qui permet d'accélérer le processus de développement d'une application Nuxt. Cela permet par exemple d'ajouter d'autres extensions ou encore de voir quels sont les routes et middlewares actif.



Middlewares :

Comme dans PHP, il est possible d'intégrer des middlewares afin par exemple de surveiller un statut d'authentification éventuel lorsque cela est nécessaire pour accéder à certaines pages.
https://nuxt.com/docs/guide/directory-structure/middleware

Conclusion :

WordPress Headless offre une flexibilité et des performances accrues pour les développeurs souhaitant tirer parti de technologies modernes pour le front-end tout en conservant la robustesse de WordPress pour la gestion du contenu. Bien que sa mise en place puisse être plus complexe et coûteuse, les avantages en termes de scalabilité, sécurité et interopérabilité en font une option attrayante pour de nombreux projets web modernes.

Wordpress headless avec d'autres frameworks JS :

Pour React, l'approche est la même, on configure notre wordpress headless de la même façon. Toutefois c'est toute une autre expérience, Vue et React se ressemble plus ou moins pour leur système de components mais au-delà la syntaxe change complètement au niveau du front. Il est également possible de se pencher vers angular qui dispose d'une syntaxe nettement plus exigeante.

Aujourd'hui il y a également astro (https://astro.build/) qui défie les performances des frameworks comme vue, react ou angular et promet des performances inégalées. Cette solution peut également être envisagée dans le cadre du développement d'un projet wordpress headless.

Documentation intéressante :

https://www.youtube.com/watch?v=dCxSsr5xuL8 : Nuxt in 100 seconds.

https://www.youtube.com/watch?v=PpyXtoM5HWQ : SEO & Metas with Nuxt 3.

https://fr.vuejs.org/guide/scaling-up/ssr : SSR avec Vue.

https://www.youtube.com/watch?v=YDvDDk7tkKI&list=PLa9NMvQUMD5eH9-Fney5J6Hvhz4qOS-1q&index=1 : Playlist react wordpress headless.

https://www.youtube.com/watch?v=YGhtEWeIKFo : Utilisation d'angular en wordpress headless.

Documentation généraliste sur Wordpress Headless :

https://www.hostinger.fr/tutoriels/wordpress-headless

https://www.agencedebord.com/blog/wordpress-headless-avantages-et-cas-dutilisation

https://raidboxes.io/fr/blog/webdesign-development/headless-cms-wordpress/

Wordpress headless avec VueJS : https://kinsta.com/fr/blog/wordpress-headless/

Wordpress headless avec React : https://kinsta.com/blog/wordpress-react/