

RABEHANTA Loïc Faniry

RAMANANTSALAMA Andy Tahiana

RAMAMONJISOA Miangaly Andraina

Brainstorming work about next.js and nuxt.js

- Nuxt et Next c'est quoi ?

Tout deux sont des Framework permettant d'implémenter du côté du client une partie de la logique et de la navigation traditionnelle réalisés côté serveur. En d'autres termes, il consiste à faire du rendu Javascript côté serveur (SSR : server-side rendering), de gérer et générer les métas-datas pour chaque page et dans le cas où il n'y a pas de données dynamiques, à faire des rendus de pages statiques.

- Installation

NuxtJS


- ➔ Pour installer Nuxt JS il faut d'abord avoir au préalable :
 - Node v10.13
 - Un éditeur de texte (VSCode, Webstorm ...)
- ➔ Pour créer rapidement un nouveau projet, nous pouvons utiliser « create-nuxt-app ». Vérifiez que npx est installé ou npm v6.1 ou yarn.



```
Yarn  NPX  NPM  
yarn create nuxt-app <project-name>
```

- ➔ Quelques questions nous seront posées (nom, options Nuxt, framework UI, TypeScript, linter, framework de test, etc.). Pour en savoir plus sur toutes les options voir Create Nuxt App.

Une fois les questions répondues, toutes les dépendances seront installées. La prochaine étape est de naviguer dans le répertoire et de démarrer notre application :



```
Yarn  NPM  
cd <project-name>  
yarn dev
```

- ➔ Notre application est maintenant en cours d'exécution sur `http://localhost:3000`.
- ➔ Nous allons utiliser le terminal pour créer les répertoires et fichiers

Il y aura donc 2 étapes pour l'installation:

- Configuration du projet
 - Pour commencer, il faut créer un répertoire vide avec le nom du projet et naviguer à l'intérieur de celui-ci :

```
mkdir <nom-du-projet>
cd <nom-du-projet>
```

- Remplacer `<nom-du-projet>` avec le nom du projet.
Puis créer un fichier nommé « `package.json` » :

```
touch package.json
```

- Ouvrir le fichier « `package.json` » avec notre éditeur préféré et ajouter ce contenu JSON :

```
{
  "name": "my-app",
  "scripts": {
    "dev": "nuxt",
    "build": "nuxt build",
    "generate": "nuxt generate",
    "start": "nuxt start"
  }
}
```

« `scripts` » définit les commandes Nuxt que nous allons lancer avec la commande « `npm run <commande>` ».

Le fichier `package.json` est comme une carte d'identité pour notre projet. Si on ne sait pas ce qu'est le fichier `package.json`, nous vous recommandons grandement de lire la documentation de npm .

- Installation de nuxt
 - Une fois le fichier `package.json` créé, nous devons ajouter nuxt à votre projet avec la commande `npm` ou `yarn` comme ci-dessous :

Yarn NPM

```
yarn add nuxt
```

Yarn

- Cette commande va ajouter nuxt comme une dépendance du projet et va automatiquement l'ajouter à notre `package.json`. Le répertoire

node_modules va aussi être créé, c'est l'endroit où sont installés tous les packages et leurs dépendances.



Un fichier `yarn.lock` ou `package-lock.json` est aussi créé, ce qui assure une consistance et une compatibilité dans les dépendances installées par votre projet.

Next JS

➔ Pour installer Next JS il faut d'abord avoir au préalable :

- La dernière version de Windows 10 ou Windows 11
- Le sous-système Windows pour Linux (WSL) y compris une distribution Linux (comme Ubuntu) et s'assurer qu'elle s'exécute en mode WSL 2. Pour le vérifier, ouvrez PowerShell et entrez : `wsl -l -v`.
- Node.js sur WSL 2

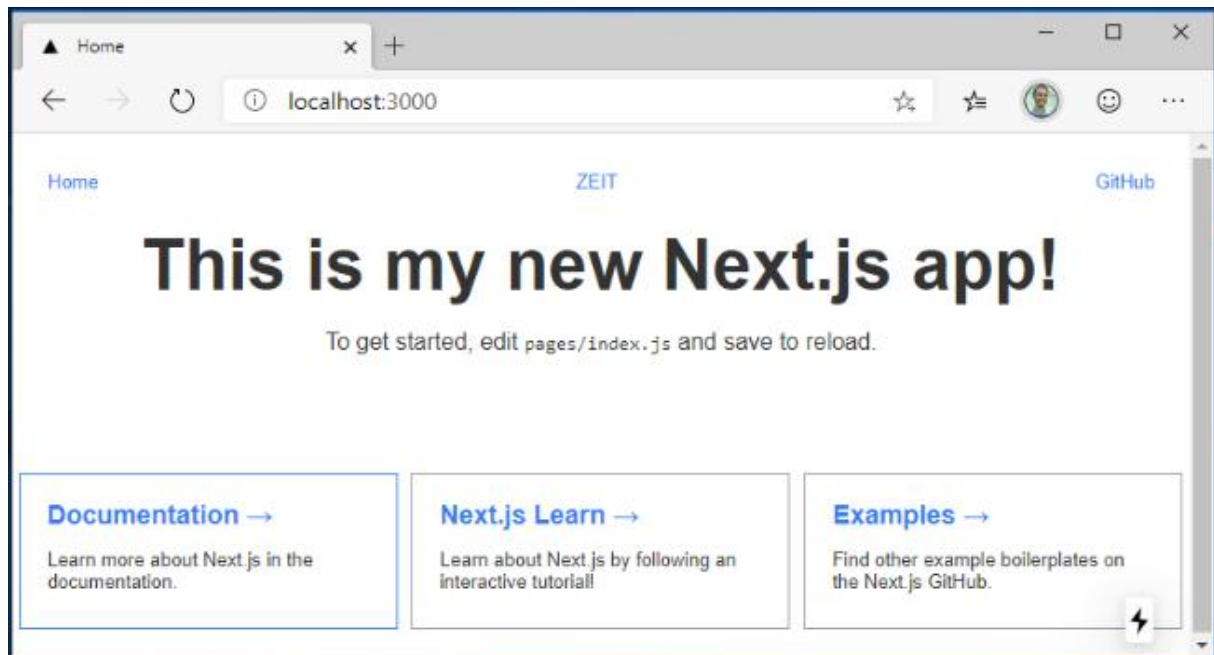
➔ L'installation de Next Js inclue l'installation de next, react et react-dom :

- Ouvrez une ligne de commande WSL
- Créez un dossier pour le nouveau projet : « `mkdir NextProjects` », puis accédez à ce répertoire : « `cd NextProjects` ».
- Installez Next.js et créez un projet (en remplaçant « `my-next-app` » par le nom que vous souhaitez donner à votre application) : « `npx create-next-app my-next-app` ».
- Une fois le package installé, modifiez les répertoires dans votre nouveau dossier d'application, « `cd my-next-app` », puis utilisez « code ». Pour ouvrir votre projet Next.js dans VS Code. Vous pourrez ainsi examiner l'infrastructure Next.js créée pour votre application. Notez que VS Code a ouvert votre application dans un environnement WSL-Remote (comme indiqué dans l'onglet vert situé en bas à gauche de votre fenêtre de VS Code). Cela signifie que lorsque vous utilisez VS Code à des fins d'édition sur le système d'exploitation Windows, il continue d'exécuter votre application sur le système d'exploitation Linux.
- Trois commandes sont à connaître une fois Next.js installé :
 - « `npm run dev` » pour exécuter une instance de développement avec rechargement à chaud, surveillance des fichiers et réexécution des tâches.
 - « `npm run build` » pour compiler votre projet.
 - « `npm start` » pour démarrer votre application en mode de production.

Ouvrez le terminal WSL intégré à VS Code (Afficher > Terminal). Assurez-vous que le chemin du terminal pointe vers le répertoire de votre projet (par ex. `~/NextProjects/my-next-app$`). Essayez ensuite d'exécuter une instance de développement de votre nouvelle application Next.js à l'aide de : « `npm run dev` »

- Le serveur de développement local démarre et une fois les pages de votre projet générées, votre terminal affiche « compiler avec succès -

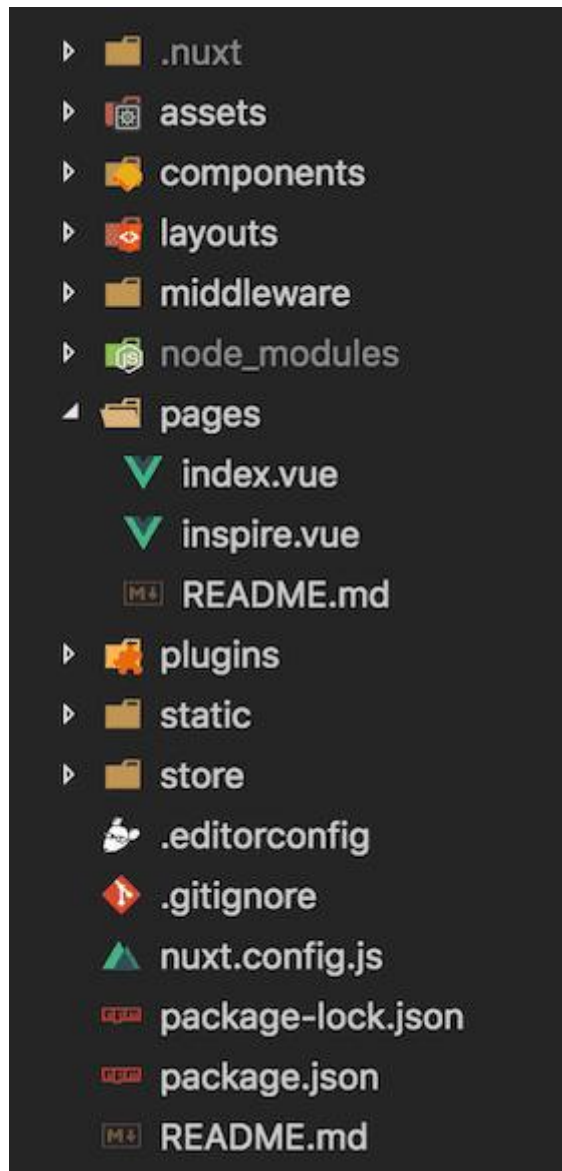
prêt sur `http://localhost:3000` ». Sélectionnez ce lien localhost pour ouvrir votre nouvelle application Next.js dans un navigateur web.



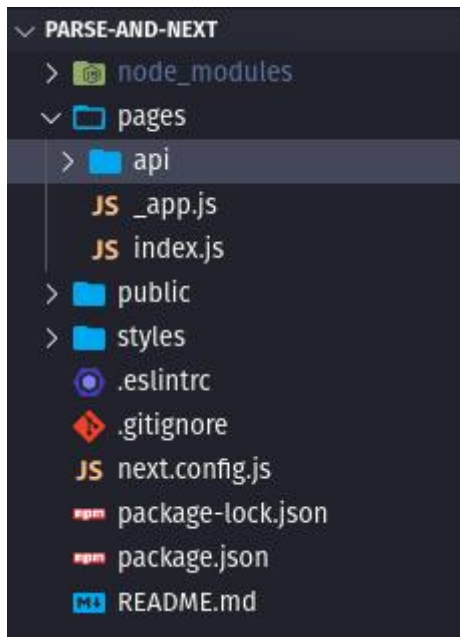
- Ouvrez le fichier « `pages/index.js` » dans votre éditeur VS Code. Recherchez le titre de la page `<h1 className='title'>Welcome to Next.js!</h1>` et remplacez-le par `<h1 className='title'>This is my new Next.js app!</h1>`. Avec votre navigateur Web toujours ouvert sur `localhost:3000`, enregistrez la modification. Vous pouvez constater que la fonctionnalité de rechargement à chaud compile et met à jour automatiquement votre modification dans le navigateur.
- Penchons-nous sur la manière dont Next.js gère les erreurs. Supprimez la balise de fermeture `</h1>` de manière à ce que votre code de titre se présente comme suit : `<h1 className='title'>This is my new Next.js app!`. Enregistrez cette modification. Vous pouvez constater qu'une erreur « Échec de la compilation » s'affiche dans votre navigateur et sur votre terminal pour vous indiquer qu'une balise de fermeture est attendue pour `<h1>`. Remplacez la balise de fermeture `</h1>` et enregistrez pour recharger la page.

- Structure de projet

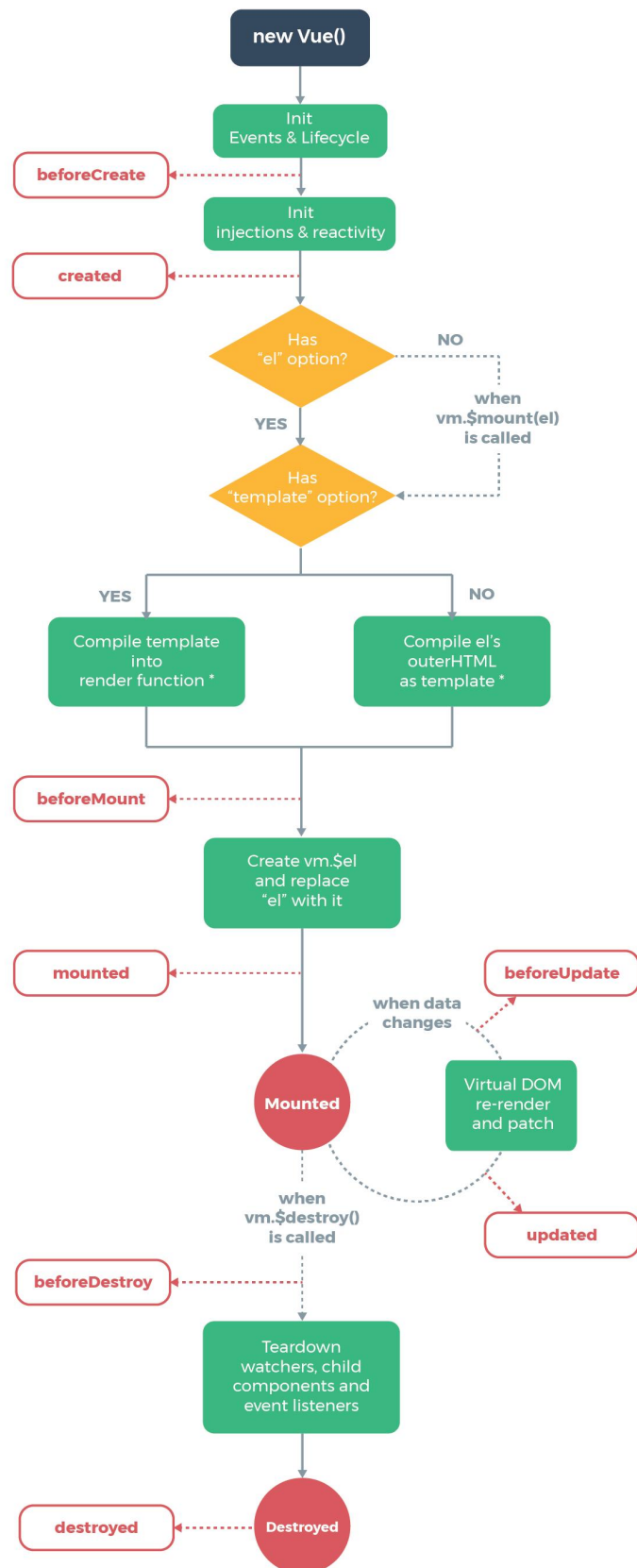
Nuxt Js



Next Js



- Cycle de vie
Le cycle de vie de Nuxt décrit les différentes étapes qui ont lieu après la phase de build, une fois que notre application a été bundlée, chunkée et minifiée. Ce qu'il se passe après cette phase dépend de si nous avons activé le rendu côté serveur ou pas. Si c'est le cas, il faut ensuite regarder le type de rendu serveur que nous avons choisi :
 - Rendu dynamique côté serveur (SSR) (nuxt start)
 - Génération statique de site (SSG) (nuxt generate).



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

➔ Binding de données

Dans Nuxt, nous avons 2 façons de récupérer de la data depuis une API. Nous pouvons utiliser la méthode `fetch` ou bien `asyncData`.

Nuxt prend en charge les modèles Vue traditionnels pour le chargement de données dans votre application côté client, comme la récupération de données dans le hook `mounted()` d'un composant. Cependant, les applications universelles doivent utiliser des hooks spécifiques à Nuxt pour pouvoir restituer les données lors du rendu côté serveur. Cela permet à votre page de s'afficher avec toutes ses données requises présentes.

Nuxt a deux hooks pour le chargement asynchrone des données :

- Le hook `fetch` (Nuxt 2.12+). Ce hook peut être placé sur n'importe quel composant et fournit des raccourcis pour le rendu des états de chargement (pendant le rendu côté client) et des erreurs.
- Le hook `asyncData`. Ce hook ne peut être placé que sur les composants page. Contrairement à `fetch`, ce hook n'affiche pas d'espace réservé de chargement pendant le rendu côté client : à la place, ce hook bloque la navigation de l'itinéraire jusqu'à ce qu'il soit résolu, affichant une erreur de page en cas d'échec.

Ces hooks peuvent être utilisés avec toute bibliothèque de récupération de données que vous choisirez. Nous vous recommandons d'utiliser `@nuxt/http` ou `@nuxt/axios` pour faire des requêtes HTTP aux API. Vous trouverez plus d'informations sur ces bibliothèques, telles que des guides de configuration, des en-têtes d'authentification, dans leur documentation respective.

Concernant le binding de donnée avec `next.js` :

Elle permet de différer les manières en fonction des cas d'utilisation.

- Pré-rendu côté serveur

```
export async function getServerSideProps(context) {
  return {
    props: {}, // will be passed to the page component as props
  }
}
```


- Pré-rendu au moment de la génération

```
export async function getStaticProps(context) {  
  return {  
    props: {}, // will be passed to the page component as props  
  }  
}
```

- Pré-rendu côté client

Pour cela, il y a deux options : useEffect ou SWR

➔ *Routing basique et dynamique, paramètre, validation*

Nuxt.js génère automatiquement la configuration pour vue-router en fonction de votre arborescence de fichiers Vue se trouvant au sein du répertoire pages.

Il existe plusieurs routes :

- *Routes basiques*
Cette arborescence :

```
pages/  
--| user/  
----| index.vue  
----| one.vue  
--| index.vue
```

Génèrera automatiquement :

```

router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'user',
      path: '/user',
      component: 'pages/user/index.vue'
    },
    {
      name: 'user-one',
      path: '/user/one',
      component: 'pages/user/one.vue'
    }
  ]
}

```

- Routes dynamiques :

Pour définir une route dynamique à l'aide d'un paramètre, vous devez définir un fichier .vue OU un répertoire préfixé par un souligné (_).

Cette arborescence :

```

pages/
--| _slug/
-----| comments.vue
-----| index.vue
--| users/
-----| _id.vue
--| index.vue

```

Génèrera automatiquement :

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'users-id',
      path: '/users/:id?',
      component: 'pages/users/_id.vue'
    },
    {
      name: 'slug',
      path: '/:slug',
      component: 'pages/_slug/index.vue'
    },
    {
      name: 'slug-comments',
      path: '/:slug/comments',
      component: 'pages/_slug/comments.vue'
    }
  ]
}
```

Comme vous pouvez le voir, la route nommée `users-id` contient le chemin `:id?` ce qui le rend optionnel. Si vous voulez le rendre obligatoire, créez un fichier `index.vue` dans le dossier `users/_id`.

Validation des paramètres de route :

Nuxt.js vous permet de définir une méthode de validation dans votre composant de routage dynamique.

Par exemple : `pages/users/_id.vue`

```
export default {
  validate ({ params }) {
    // Doit être un nombre
    return /\d+$/ .test(params.id)
  }
}
```

Si la méthode de validation ne renvoie pas true, Nuxt.js chargera automatiquement la page d'erreur 404.

Pour plus d'informations à propos de la méthode de validation, consultez la partie Pages de l'API pour La méthode validate

Next.js :

Next.js utilise le système de routage par fichier basé sur les **pages** pour structurer l'aspect du routage de l'application.

- **Routing basique**

Le routing basique

- **Paramètres**

- **Routing dynamique**

Pour ce cas, il suffit principalement d'ajouter des crochets à une page ([paramètre]) pour rendre dynamique le routing.

- **Paramètres**

```
export default function IndexPage() {
  return (
    <ul>
      <li>
        <Link href="/">
          <a> *NOM* </a>
        </Link>
      </li>
      <li>
        <Link href="/[nom1]" as="/<u>nom1</u>">
          <a> *NOM* </a>
        </Link>
      </li>
      <li>
```

```

        <Link href="/chemin [nom2]" as="/chemin/nom2">
          <a> *NOM* </a>
        </Link>
      </li>
    </ul>
  )
}

```

*les parties en vertes doivent être remplacées par les indications correspondants. Et comme il s'agit d'un routing dynamique, il va falloir définir l'attribut de « as » :

```

import { useRouter } from "next/router"

export default function DynamicPage() {
  const router = useRouter()
  const {
    query: { id },
  } = router
  return <div>The dynamic route is {id}</div>
}

```

La seconde option:

```

export default function MyDynamicPage({ example }) {
  return <div>My example is {example}</div>
}

MyDynamicPage.getInitialProps = ({ query: { example } })
=> {
  return { example }
}

```

- ➔ Navigation
- ➔ Layout
- ➔ Style
- ➔ Composant
- ➔ Fetching des données

Le hook fetch de Nuxt est appelé après que l'instanciation du composant soit faite côté serveur : this est disponible à l'intérieur.

```

export default {
  async fetch() {
    console.log(this)
  }
}

```

À chaque fois que nous avons besoin d'avoir de la data asynchrone. « fetch » est appelé côté serveur lors du render de la route et côté client lors de la navigation.

Cela expose `$fetchState` au niveau du composant et avec les propriétés suivantes :

- « `pending` » est un Booléen, il permet d'afficher un placeholder quand `fetch` est appelé côté client.
- « `error` », il peut être soit à `null` soit à `Error` et nous permet d'afficher un message
- « `Timestamp` » est l'horodatage du dernier `fetch`, cela peut s'avérer utile pour faire du cache avec `keep-alive`

Nous avons aussi accès à `this.$fetch()`, utile si nous voulons appeler le hook `fetch` dans notre composant.

```
components/NuxtMountains.vue

<template>
  <p v-if="$fetchState.pending">Récupération des montagnes... ⚠️</p>
  <p v-else-if="$fetchState.error">Une erreur est survenue :(</p>
  <div v-else>
    <h1>Montagnes Nuxt</h1>
    <ul v-for="mountain of mountains">
      <li>{{ mountain.title }}</li>
    </ul>
    <button @click="$fetch">Actualiser</button>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        mountains: []
      }
    },
    async fetch() {
      this.mountains = await fetch(
        'https://api.nuxtjs.dev/mountains'
      ).then(res => res.json())
    }
  }
</script>
```

Next.js :

Le fetching permet de récupérer des données pouvant rendre le contenu différent selon l'utilisation. Tout cela s'exécute avec `fetch`.

- ➔ API Routes
- ➔ Authentication
- ➔ Middleware

Le middleware vous permet de définir une fonction personnalisée qui sera exécutée avant le rendu d'une page ou d'un groupe de pages.

Tous les middlewares devraient être placés dans le répertoire `middleware/`. Le nom du fichier étant le nom du middleware (`middleware/auth.js` deviendra le middleware `auth`).

Un middleware reçoit le contexte comme premier argument :

```
export default function (context) {  
  context.userAgent = process.server ? context.req.headers['user-agent'] : navigator.userAgent  
}
```

Le middleware sera exécuté en série dans l'ordre suivant :

- `nuxt.config.js`
- Mises en page correspondantes
- Pages correspondantes

Un middleware peut être asynchrone, retourner une Promise ou utiliser une fonction de rappel en second argument :

`middleware/stats.js`

```
import axios from 'axios'  
  
export default function ({ route }) {  
  return axios.post('http://my-stats-api.com', {  
    url: route.fullPath  
  })  
}
```

Puis, dans `nuxt.config.js`, pour une mise en page ou une page, utilisez le mot-clé `middleware` :

`nuxt.config.js`

```

module.exports = {
  router: {
    middleware: 'stats'
  }
}

```

Next.js :

- **Authentification des pages générées statistiquement**

```

import useUser from '../lib/useUser'
import Layout from '../components/Layout'

const Profile = () => {
  // Fetch the user client-side
  const { user } = useUser({ redirectTo: '/login' })

  // Server-render loading state
  if (!user || user.isLoggedIn === false) {
    return <Layout>Loading...</Layout>
  }

  // Once the user request finishes, show the user
  return (
    <Layout>
      <h1>Your Profile</h1>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </Layout>
  )
}

export default Profile

```

Avant le contenu mis en cache, il y a l'exécution du middleware permettant de personnaliser les fichiers et les pages statiques.

Parmi les middlewares, il y a par exemple l'authentification, les tests A/B, les pages localisées, la protection contre les bots.

Une fois que NextJS est installé, il faut créer un fichier middleware.ts ou middleware.js à la racine ou dans le source de répertoire de même niveau de la page. Et enfin apporter une fonction middleware :


```
// middleware.ts
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

// This function can be marked `async` if using `await` inside
export function middleware(request: NextRequest) {
  return NextResponse.redirect(new URL('/about-2', request.url))
}

// See "Matching Paths" below to learn more
export const config = {
  matcher: '/about/:path*',
}
```

- ➔ Connexion à la base de données
- ➔ Déploiement

Nuxt.js permet de choisir entre trois modes de déploiement pour votre application : rendu côté serveur, application monopage ou généré de manière statique.

- Déploiement pour un rendu côté serveur (universelle)

Pour déployer, au lieu d'exécuter nuxt, vous voulez probablement faire d'abord la construction. Par conséquent, la construction et le démarrage sont des commandes distinctes :

```
nuxt build
nuxt start
```

Le fichier package.json suivant est recommandé :

```
{
  "name": "my-app",
  "dependencies": {
    "nuxt": "latest"
  },
  "scripts": {
    "dev": "nuxt",
    "build": "nuxt build",
    "start": "nuxt start"
  }
}
```

Note : nous recommandons d'ajouter `nuxt` dans `.npmignore` ou `.gitignore`.

- Déploiement pour une génération statique

Nuxt.js vous offre la possibilité d'héberger votre application web sur tout hébergement statique.

Pour générer votre application web en fichiers statiques :

```
npm run generate
```

Il créera un dossier `dist` avec tout à l'intérieur prêt à être déployé sur un hébergement de site statique.

Si vous avez un projet avec des routes dynamiques, regardez la configuration de la commande `generate` afin de dire à Nuxt.js comment générer ces routes dynamiques.

- Déploiement pour une application monopage (SPA)

`nuxt generate` a toujours besoin du SSR pendant le temps de génération afin de prérendre nos pages dans le but d'obtenir un chargement de page rapide et du contenu solide pour la SEO. Le contenu est généré lors de la phase de `build`. Il ne faut pas l'utiliser, par exemple, pour les applications où le contenu dépend de l'authentification de l'utilisateur ou pour une API en temps réel (du moins pour le premier chargement).

L'idée de l'application monopage est simple ! Quand le mode SPA est activé en utilisant `mode: 'spa'` ou l'option `--spa`, la génération se lance automatiquement après le `build` mais cette fois sans contenu de page et seulement avec les meta, ressources et liens communs.

Donc pour un déploiement en mode SPA, vous devez :

- Changez le mode dans `nuxt.config.js` pour `spa`.
- Lancez `npm run build`.
- Déployez le dossier `dist/` créé sur votre hébergement statique comme Surge, GitHub Pages ou nginx.

Une autre possibilité de déploiement est que nous pouvons utiliser Nuxt comme un middleware dans des frameworks si le mode est `spa`. Ceci aide à réduire le temps de chargement et à utiliser Nuxt dans des projets où le SSR n'est pas possible.

Qu'est-ce qui différencie alors ces 2 Framework ?

- ➔ Installation : Nuxt JS propose plus d'options, préparé et est plus facile à manipulé durant l'installation, notamment l'installation directe des typescript, des frameworks css comme Bootstrap, bulma...
Contrairement à cela NextJS nécessite des pièces supplémentaires à installer.
- ➔ Création de la page index et la récupération du contenu : Nuxt dispose d'un accesseur particulier (`$content`) qui permet de gérer directement le contenu, et ce grâce à une syntaxe ostensiblement de celle de MongoDB. Tout cela rend la gestion du contenu beaucoup plus facile et que ce que nous avons dans Next, où il faut coder l'accès au contenu soi-même.
- ➔ Création d'une page d'un article : Nuxt s'installe un testeur Tailwind qui permet de voir directement quels sont les styles qui ont été activés par la configuration sur cette page. Nuxt propose un éditeur intégré pour modifier à la volée le contenu texte également. Ainsi Nuxt propose une meilleure expérience développeur.