Comprendre Vue.js : Nuxt

On a vu au cours des précédents articles que Vue se présente comme un outil simple et puissant. D'autre part il bénéficie d'une communauté très active et ses possibilités s'élargissent rapidement. Dans le présent article je vous présente Nuxt, un framework dont l'ambition n'est rien moins que créer des applications Vue.js universelles.

L'idée est d'utiliser le **SSR** (Server Side Rendering). C'est quoi cette bête ? Imaginez que vous créez une application avec pas mal de manipulation de code côté client, vous risquez deux écueils : votre application risque de perturber les moteurs de recherche, ça peut poser des soucis parfois au navigateur qui se retrouve avec un gros boulot. La solution est juste ment le **SSR** ! On crée l'application côté serveur et on envoie ce qu'il faut au client. En fait on a un isomorphisme entre le serveur et le client, le code est interprété de la même façon des deux côtés.

Évidemment côté serveur on a besoin de Node.js et cette approche ne fait pas l'unanimité… Mais Nuxt a d'autres tours dans son sac : on peut créer une SPA (Single Page Application) ou générer une application statique !

Sous le capot Nuxt utilise ce qu'on a déjà vu : Vue 2.0, Vue-Router, Vuex…

On va un peu s'amuser avec Nuxt pour comprendre son fonctionnement et voir ce qu'il apporte réellement…

Installation

On crée une application Nuxt à partir de Vue Cli :

```
npx create-nuxt-app test
```

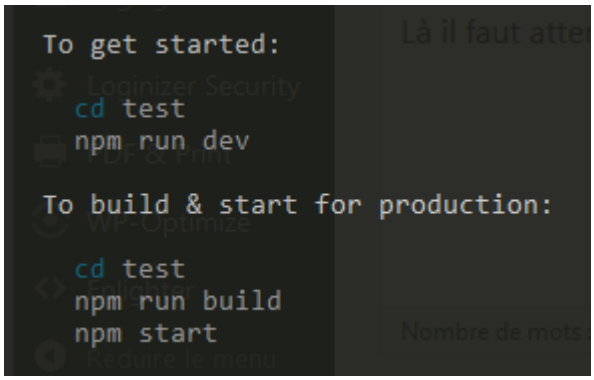
```
...
? Project name test
? Project description Un essai de Nuxt
```

```
? Use a custom server framework none
? Use a custom UI framework vuetify
? Choose rendering mode Universal
? Use axios module yes
? Use eslint yes
? Use prettier no
? Author name bestmomo
? Choose a package manager npm
```

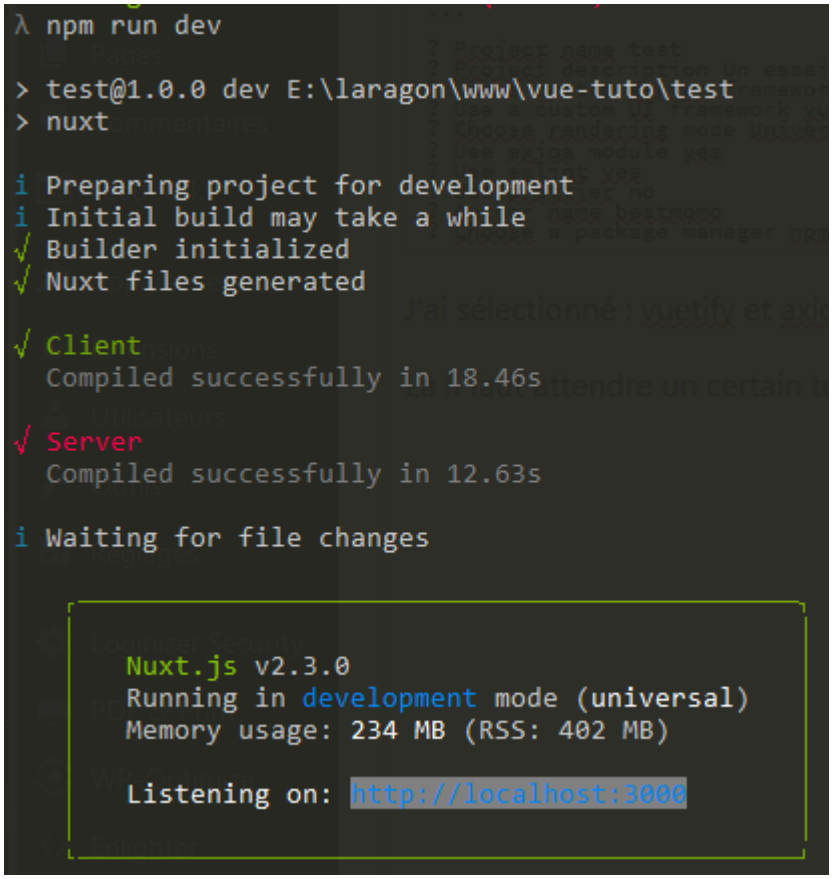
J’ai sélectionné : vuetify et axios.

D’autre part j’ai choisi le mode universel, c’est à dire avec la génération côté serveur, l’autre choix est SPA (Single Page Application) avec juste une génération côté client.

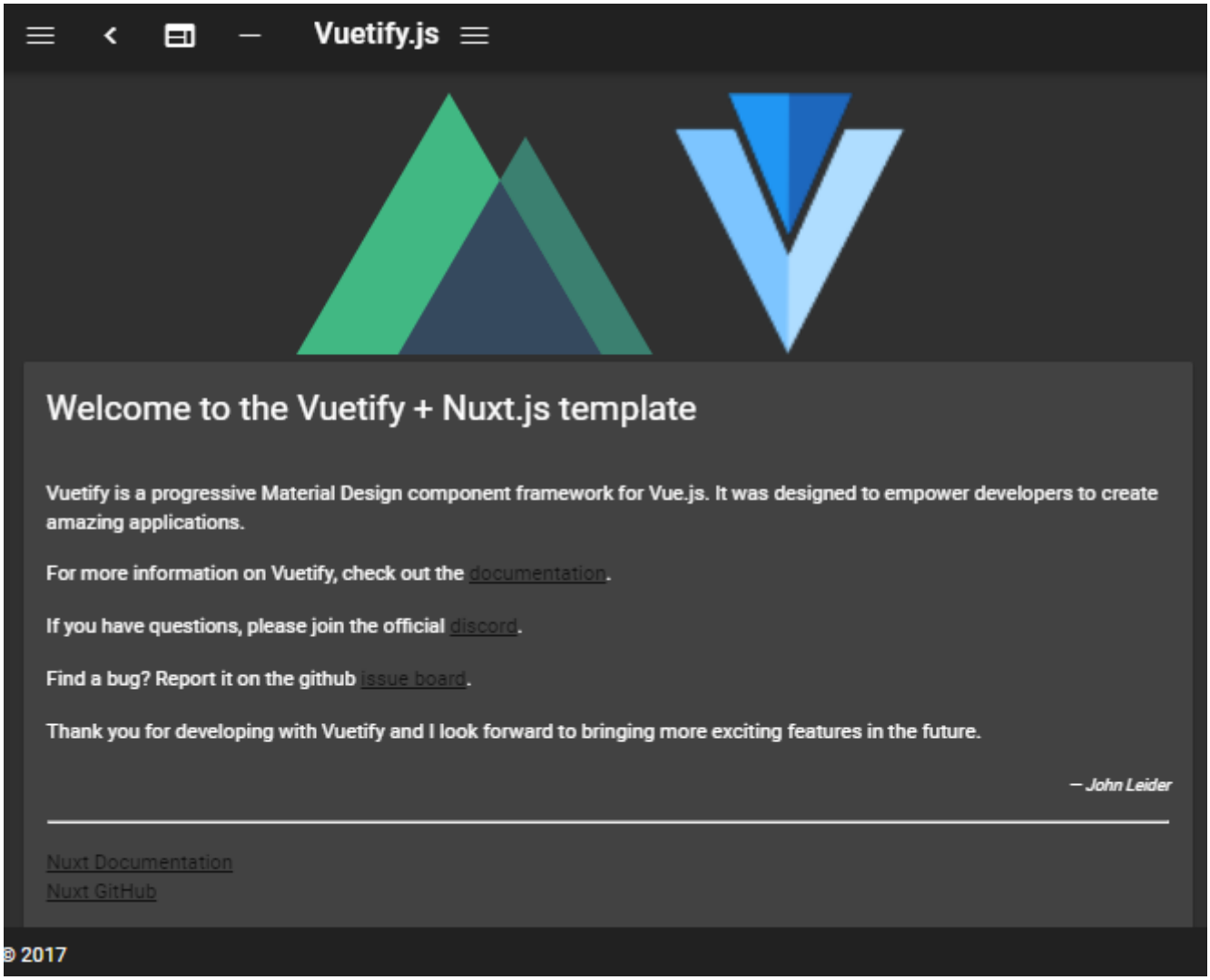
Là il faut attendre un certain temps que tout s’installe… Si tout va bien ça se termine avec ça :



On lance en mode développement :



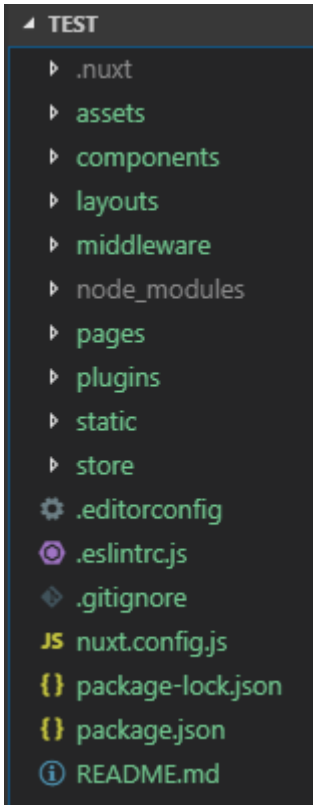
On se rend donc à l’adresse localhost:3000 :



J’ai cet aspect parce que j’ai ajouté Vuetify dans l’installation. C’est un superbe framework pour le Material Design.

La structure de l’application

On a cette structure de dossiers :

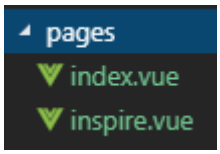


Faisons un peu le point :

- **assets** : les ressources non compilées comme les images, le CSS, SASS ou LESS
- **components** : les composants Vue.js
- **layouts** : les mises en page
- **middleware** : si on veut une action avant différents événements comme par exemple le changement de page
- **pages** : contient des composants Vue.js pour le routage, la structure de ce dossier va créer automatiquement les routes
- **plugins** : si on veut ajouter des plugins à Vue.js
- **static** : les fichiers statiques comme robots.txt
- **store** : le dossier pour Vuex

Les routes

Comme je l'ai dit ci-dessus le routage avec Nuxt est automatiquement créé pour vue-router en suivant l'arborescence du dossier **pages**. On a actuellement ces deux composants :



Si vous aller voir dans le fichier **nuxt/router.js** vous allez trouver ce code :

```
export function createRouter() {
  return new Router({
    mode: 'history',
    base: '/',
    linkActiveClass: 'nuxt-link-active',
    linkExactActiveClass: 'nuxt-link-exact-active',
    scrollBehavior,

    routes: [{
      path: "/inspire",
      component: _2ffe4ba3,
      name: "inspire"
    }, {
      path: "/",
      component: _ba1b444a,
      name: "index"
    }],

    fallback: false
  })
}
```

On voit les deux routes créées dans ce composant de Nuxt.

On va voir si la création dynamique fonctionne, on crée un nouveau dossier avec un composant :

Avec ce code (la syntaxe des composants utilisés est liée à Vuetify) :

```
<template>
  <v-layout>
    <v-flex text-xs-center>
      Un essai du routeur
    </v-flex>
  </v-layout>
```

```
</template>
```

On voit que le routeur se met à jour :

```
routes: [{
  path: "/inspire",
  component: _2ffe4ba3,
  name: "inspire"
}, {
  path: "/articles/essai",
  component: _5bd398d9,
  name: "articles-essai"
}, {
  path: "/",
  component: _ba1b444a,
  name: "index"
}],
```

Et si on entre l'adresse on a bien la page créée :

C'est vraiment pratique !

On peut aussi très facilement créer une route dynamique avec un paramètres en prévoyant un souligné, par exemple :

Et dans le routeur on retrouve le paramètre :

```
}, {
  path: "/articles/:id?",
  component: _633358a6,
  name: "articles-id"
}, {
```

Maintenant avec ce code dans le composant :

```
<template>
  <v-layout>
    <v-flex text-xs-center>
      On a l'id {{ this.$route.params.id }}
    </v-flex>
  </v-layout>
</template>
```

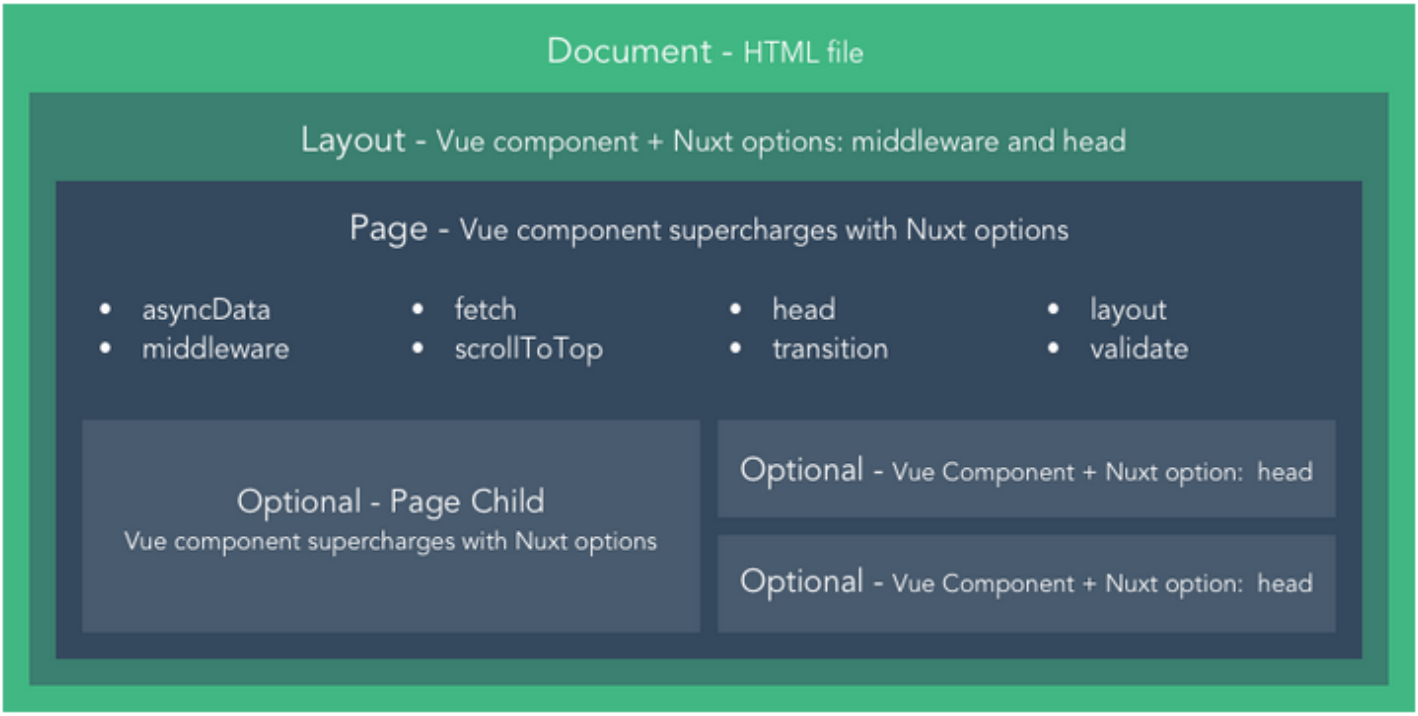
Ça fonctionne :

Il est aussi possible de créer de jolie transitions, je vous laisse consulter [la documentation](#).

De la même manière on peut accomplir une action avec le chargement des pages avec [un middleware](#).

Mise en page

Voyons maintenant la création des pages. Voici l'organisation globale :



Le document

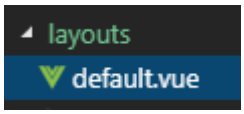
Le premier niveau est le document. par défaut on a un fichier **.nuxt/views/app.template.html** :

```
<!DOCTYPE html>
<html {{ HTML_ATTRS }}>
  <head>
    {{ HEAD }}
  </head>
  <body {{ BODY_ATTRS }}>
    {{ APP }}
  </body>
</html>
```

On peut intervenir à ce niveau pour une mise en forme qui concerne toute l'application en surchargeant ce fichier avec un fichier **app.html** à la racine mais c'est pas très utile.

Les layouts

Au second niveau on a les layouts (mises en page) qui permettent une mise en page personnalisée. On en a une par défaut à l'installation :



Comme j'ai ajouté Vuetify ce layout est assez fourni avec une navigation élégante. L'emplacement pour les pages se situe ici :

```
<v-content>
  <v-container>
    <nuxt />
  </v-container>
</v-content>
```

c'est le composant **nuxt** qui permet l'affichage de la page.

Les pages

Enfin les pages ont aussi la liberté d'avoir leur propre présentation. Ce sont évidemment des composants de Vue. Mais le plus important à savoir pour les pages c'est qu'elles disposent de clés spéciales qui rendent le développement plus facile :

- **asyncData** : pour récupérer des données du serveur avec Axios si on utilise pas Vuex :

```
export default {
  async asyncData ({ params }) {
    let { data } = await axios.get(`https://depot/articles/${id}`)
    return { titre: data.titre}
  }
}
```

- **fetch** : pour remplir le store avant le chargement de la page (documentation ici).
- **head** : pour définir des metas pour la page
- **layout** : pour choisir un layout pour la page
- **transition** : pour avoir un effet de transition pour la page (documentation ici)
- **middleware** : pour définir un middleware pour la page

La liste n'est pas complète !

Un exemple

Les données

Arrivé à ce stade on se dit qu'un petit exemple ne ferait pas de mal… On va créer un fichier de données **db.json** à la racine avec ce code :

```
{
  "continents" : [
    { "id": 1, "name": "Europe" },
    { "id": 2, "name": "Amérique" },
    { "id": 3, "name": "Asie" }
```

```
  ],
  "countries" : [
    { "id": 1, "continentID": 1, "name": "France"},
    { "id": 2, "continentID": 1, "name": "Angleterre"},
    { "id": 3, "continentID": 1, "name": "Espagne"},
    { "id": 4, "continentID": 1, "name": "Italie"},
    { "id": 5, "continentID": 1, "name": "Etats-Unis"},
    { "id": 6, "continentID": 2, "name": "Vénézuéla"},
    { "id": 7, "continentID": 2, "name": "Canada"},
    { "id": 8, "continentID": 2, "name": "Colombie"},
    { "id": 9, "continentID": 2, "name": "Chili"},
    { "id": 10, "continentID": 3, "name": "Chine"},
    { "id": 11, "continentID": 3, "name": "Inde"},
    { "id": 12, "continentID": 4, "name": "Corée"},
    { "id": 13, "continentID": 5, "name": "Russie"}
  ]
}
```

Et on lance un serveur pour notre petite API (si vous n'avez pas **json-server** installé alors installez-le) :

```
json-server db.json --port 3001
```

J'ai changé le port par défaut qui est 3000 pour ne pas entrer en conflit avec Nuxt qui l'utilise aussi.

On peut maintenant accéder aux continents avec http://localhost:3010/continents et à un continent spécifique avec http://localhost:3010/continents/id. Et c'est la même chose pour les pays.

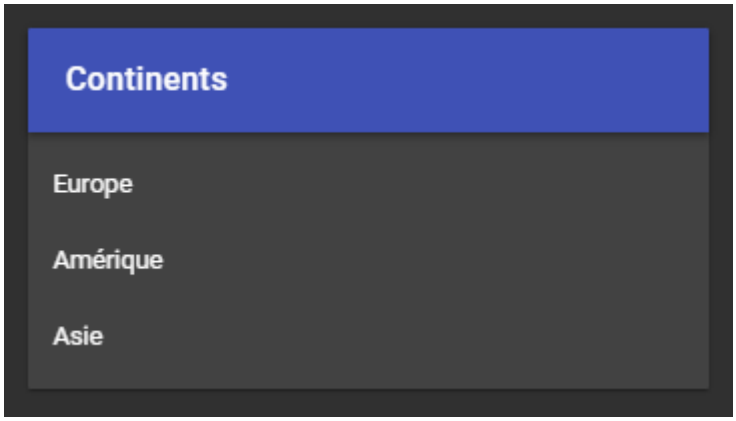
Les continents

On crée un fichier **pages/continents.vue** avec ce code :

```
<template>
  <v-layout>
    <v-flex
      xs12
      sm6
      offset-sm3>
      <v-card>
        <v-toolbar
```

```
          color="indigo"
          dark>
          <v-toolbar-title>Continents</v-toolbar-title>
        </v-toolbar>
        <v-list>
          <v-list-tile
            v-for="continent in continents"
            :key="continent.id"
          >
            <v-list-tile-content>
               <v-list-tile-title>{{ continent.name }}</v-list-
tile-title>
            </v-list-tile-content>
          </v-list-tile>
        </v-list>
      </v-card>
    </v-flex>
  </v-layout>
</template>

<script>
import axios from 'axios'

export default {
  name: 'Continents',
  async asyncData () {
    return axios.get('http://localhost:3010/continents')
    .then((res) => {
      return {
        continents: res.data
      }
    })
  }
}
</script>
```

On utilise Axios avec une promesse, quand les données arrivent on renseigne la variable **continents** qui vient fusionner avec les données du composant (si elles existaient !). Ensuite dans le template on utilise **v-for** pour afficher tous les noms des continents avec l’url **…/continents** :



On voit qu’on s’en sort très simplement !

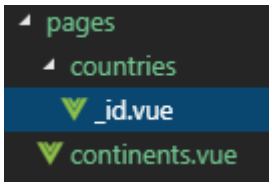
Maintenant j’aimerais qu’en cliquant sur un nom de continent j’ouvre la page des pays de ce continent. On a le choix entre utiliser un composant **nuxt-link** (méthode préconisée par la documentation), soit passer par une méthode. Je vais opter pour cette seconde solution pour le pas perturber le style de la page :

```
<template>

  ...

            <v-list-tile-content>
              <v-list-tile-title
                @click="countries(continent.id)"
              >{{ continent.name }}</v-list-tile-title>
            </v-list-tile-content>

  ...

</template>

<script>

  ...

  methods: {
    countries(id) {
      this.$router.push(`/countries/${id}`)
    }
  }
}
</script>
```

On récupère l’id du continent et on utilise la route **countries** avec ce paramètre.

Les pays

Maintenant on va créer le composant pour afficher les pays. Comme on a une route dynamique on adopte ce que je vous ai déjà décrit plus haut :

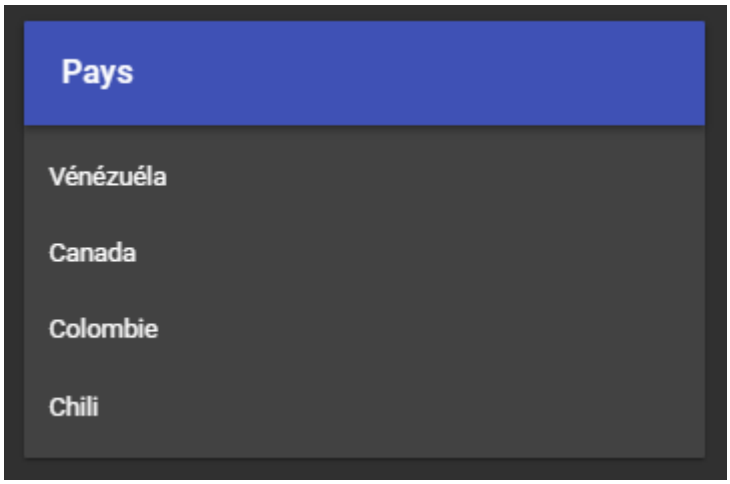


Avec ce code :

```
<template>
  <v-layout>
    <v-flex
      xs12
      sm6
      offset-sm3>
      <v-card>
        <v-toolbar
          color="indigo"
          dark>
          <v-toolbar-title>Pays</v-toolbar-title>
        </v-toolbar>
        <v-list>
          <v-list-tile
            v-for="country in countries"
            :key="country.id"
          >
            <v-list-tile-content>
              <v-list-tile-title>{{ country.name }}</v-list-tile-
title>
            </v-list-tile-content>
          </v-list-tile>
        </v-list>
      </v-card>
    </v-flex>
  </v-layout>
</template>
```

```
<script>
import axios from 'axios'

export default {
  name: 'Countries',
  async asyncData ({ params }) {
                                                                    return
axios.get(`http://localhost:3010/countries?continentID=${params.id
}`)
    .then((res) => {
      return {
        countries: res.data
      }
    })
  }
}
</script>
```

On utilise encore Axios pour envoyer une requête, cette fois paramétrée. Ensuite on utilise **v-for** pour afficher les pays.



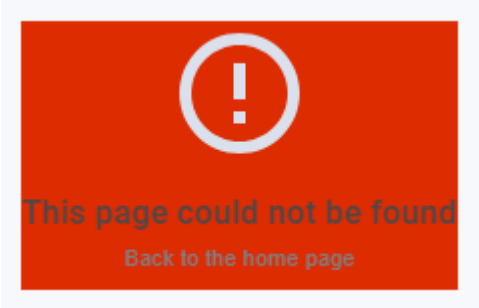
Toujours aussi simple !

Mais que se passe-t-il si on utilise un paramètre qui n’est pas un nombre ? On va se retrouver avec une page vide. Il est possible d’effectuer une validation avec la méthode **validate** :

```
export default {
  name: 'Countries',
  validate ({ params }) {
    // Doit être un nombre
    return /^\d+$/.test(params.id)
```

```
  },
```

Maintenant si le paramètre n’est pas un nombre on obtient la page d’erreur par défaut de Nuxt :



On peut créer sa page personnalisée en ajoutant un composant **layouts/error.vue**.

Les plugins

Il y a quand même un petit souci avec notre exemple : on importe deux fois Axios. Il serait bien de ne le faire qu’une fois… Nuxt permet cela. On a un fichier **nuxt.config.js** avec toute la configuration par défaut. On va juste déclarer axios dans ce fichier :

```
build: {
  vendor: ['axios'],
  ...

}
```

On peut ensuite l’importer dans plusieurs modules, il ne sera chargé qu’une fois !

C’est le même principe si on veut utiliser un autre plugin, on l’ajoute dans **vendor**.

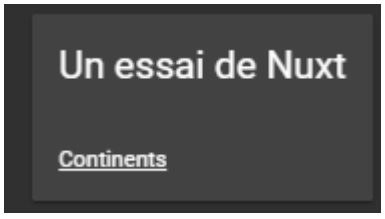
Générer un projet

Il y a deux possibilités pour la génération :

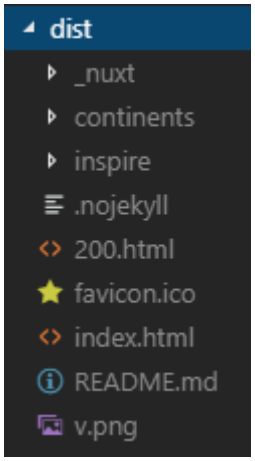
- **next build** : application avec un serveur web
- **next generate** : application statique

On va utiliser cette seconde possibilité pour notre exemple.Mais on va quand même préparer le terrain en modifiant le composant **index.vue** :

```
<template>
  <v-layout
    column
    justify-center
    align-center>
    <v-flex
      xs12
      sm8
      md6>
      <v-card>
         <v-card-title class="headline">Un essai de Nuxt</v-card-
title>
        <v-card-text>
          <nuxt-link
            class="white--text"
            to="/continents">
            Continents
          </nuxt-link>
        </v-card-text>
      </v-card>
    </v-flex>
  </v-layout>
</template>
```



On lance maintenant la génération et on se retrouve avec un dossier **dist** :



Et ça fonctionne !

Conclusion

Nuxt est un framework plutôt intéressant et bien conçu, il mérite d'être utilisé ! Même si vous n'êtes pas partant pour du SSR parce que Node n'est pas votre tasse de thé vous pouvez très bien l'utiliser en mode SPA (donc juste côté client) ou en génération statique comme on l'a vu ci-dessus.

Et Laravel ? Oui après tout c'est un blog sur Laravel ici ! Eh bien Laravel et Nuxt peuvent faire bon ménage même si a priori la cohabitation peut sembler délicate. Heureusement quelqu'un s'est attelé à la tâche et nous a pondu de supers packages ! On va juste attendre qu'ils soient actualisé pour la version 2. Mais de toute façon on peut très bien faire les développements séparés : SPA avec Nuxt et l'API avec Laravel ou Lumen.