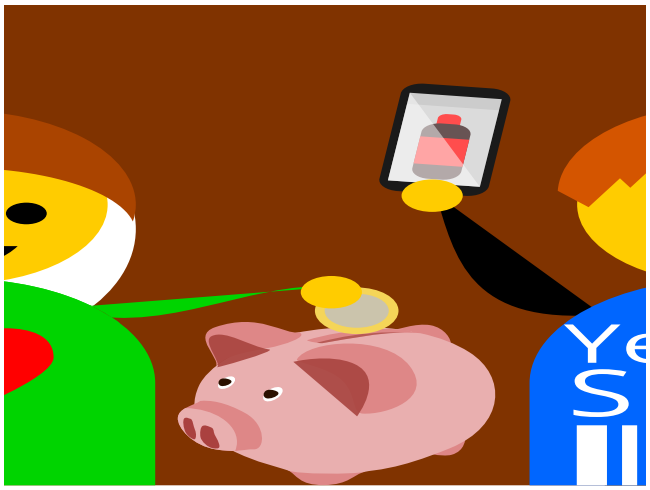
# Comprendre Vue.js : Nuxt en action

Dans le précédent article je vous ai présenté Nuxt. On a vu qu'il simplifie le développement d'une application Vue en gérant par exemple de façon automatisé le routage. Dans le présent article je vous propose de l'utiliser pour une petite application de recherches d'informations nutritives à partir d'un code-barre. Ce genre d'application se multiplie étant donné la sensibilisation croissante aux méfaits des additifs et à l'équilibre alimentaire. Il existe une base de donnée ouverte renseignée par les utilisateurs qui expose une API que nous allons utiliser. C'est d'ailleurs cette API qui est exploitée par toutes les applications qui existent actuellement.

# Open Food Facts



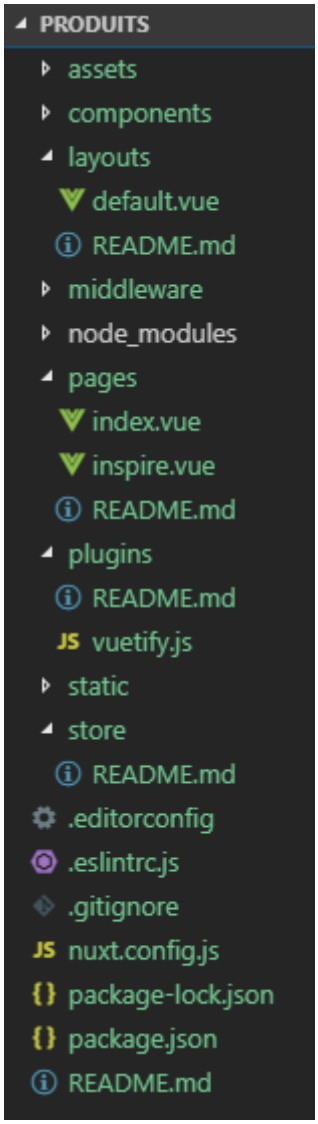
Open Food Facts est une initiative citoyenne mondiale indépendante de l'industrie à laquelle tout un chacun est invité à participer. Elle récence actuellement 452688 produits et la base s'agrandit tous les jours. Pour chaque produit on peut obtenir une foule d'informations comme les ingrédients utilisés, le nutriscore, le fabricant…

# On crée le projet

On crée un projet Nuxt en mode SPA :

```
npx create-nuxt-app produits
npx: installed 402 in 10.099s
> Generating Nuxt.js project in E:\laragon\www\vue-tuto\produits
? Project name produits
? Project description Tout savoir sur les produits
? Use a custom server framework none
? Use a custom UI framework vuetify
? Choose rendering mode Single Page App
? Use axios module yes
? Use eslint yes
? Use prettier no
? Author name bestmomo
? Choose a package manager npm
```

J’ai ajouté Vuetify, Axios et Eslint. On se retrouve avec cette architecture :




On n’a plus qu’à lancer :

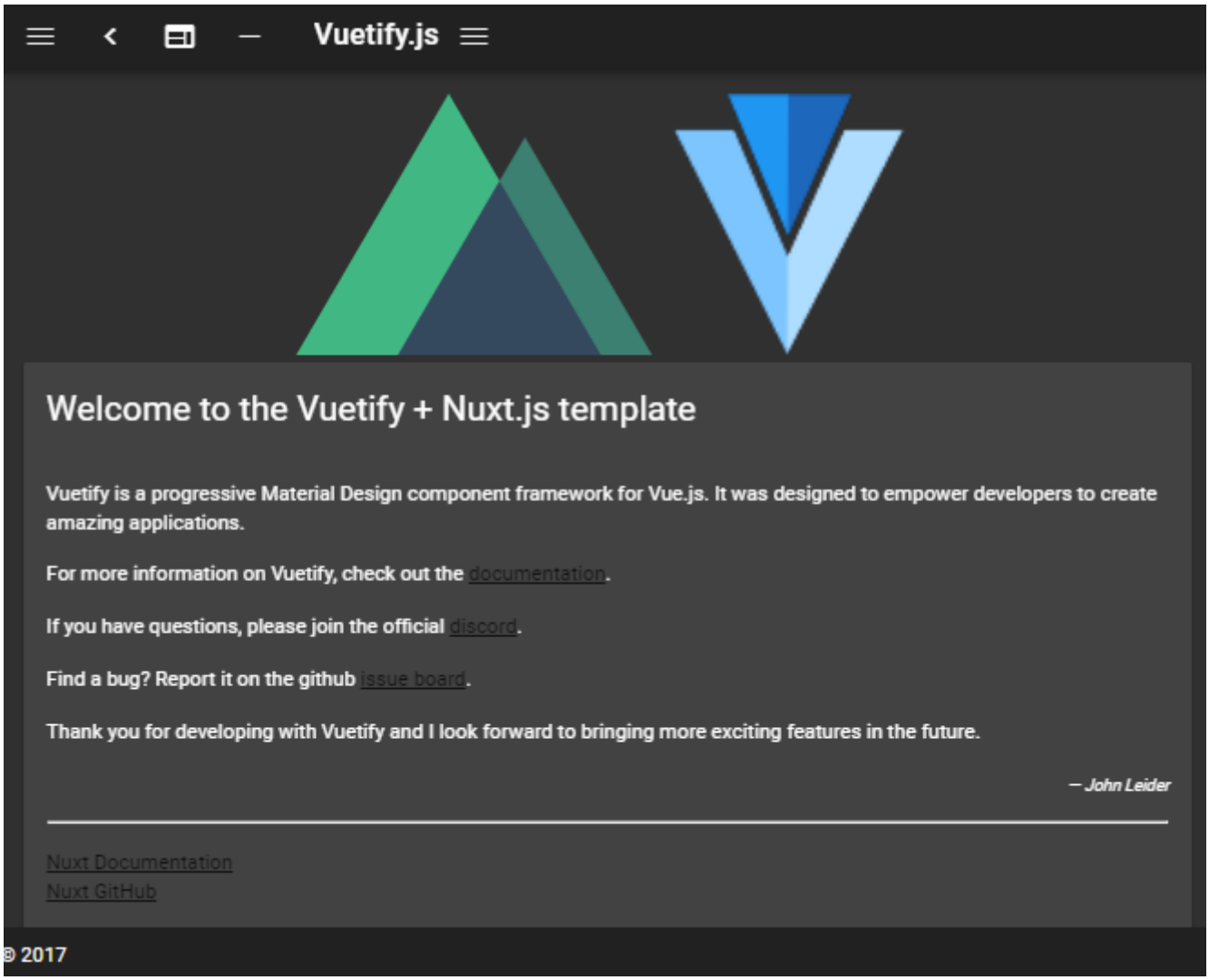
npm run dev

On attend que tout se construise…


Ça se termine par quelque chose comme ça :

```
Nuxt.js v2.3.2
Running in development mode (spa)
Memory usage: 154 MB (RSS: 298 MB)

Listening on: http://localhost:3000
```

Et on se retrouve avec la même page d'accueil que lors du précédent article :




# Le layout et la page d'accueil

On va commencer par changer le layout (**layouts/default.vue**) :

```
<template>
  <v-app dark>
    <v-navigation-drawer
      v-model="drawer"
```

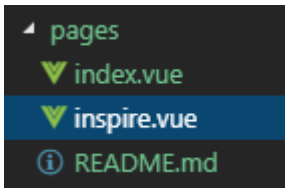
```
      clipped
      fixed
      app
    >
      <v-list>
        <v-list-tile
          to="/">
          <v-list-tile-action>
            <v-icon>find_in_page</v-icon>
          </v-list-tile-action>
          <v-list-tile-content>
            <v-list-tile-title>Rechercher</v-list-tile-title>
          </v-list-tile-content>
        </v-list-tile>
        <v-list-tile
          href="https://fr.openfoodfacts.org"
          target="_blank">
          <v-list-tile-action>
            <v-icon>help</v-icon>
          </v-list-tile-action>
          <v-list-tile-content>
            <v-list-tile-title>Comprendre</v-list-tile-title>
          </v-list-tile-content>
        </v-list-tile>
      </v-list>
    </v-navigation-drawer>
    <v-toolbar
      app
      fixed
      clipped-left>
      <v-toolbar-side-icon
        @click.stop="drawer = !drawer"/>
      <v-toolbar-title>Open Food facts</v-toolbar-title>
    </v-toolbar>
    <v-content>
      <v-container
        fluid
        fill-height>
        <nuxt />
      </v-container>
    </v-content>
  </v-app>
</template>
```

```
<script>
  export default {
    data: () => ({
      drawer: null
    })
  }
</script>
```

On change ainsi la mise en page globale :


Le lien **Comprendre** emmène sur le site d’open Food facts.

On peut supprimer la page **inspire** :



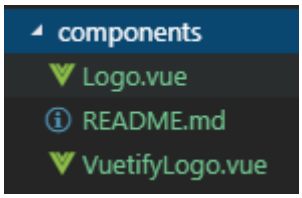

Et on met ce code dans **index.vue** :

```
<template>
  <v-layout
    column
    justify-center
    align-center>
    <v-flex
      xs12
      sm8
      md6>
      <div class="text-xs-center">
                                                    <img
src="https://static.openfoodfacts.org/images/misc/openfoodfacts-lo
go-fr-178x150.png">
      </div>
      <v-alert
```

```
        :value="alert"
        type="error">
        Aucun produit trouvé !
      </v-alert>
      <v-card>
        <v-card-title
          class="headline">
          Bienvenue sur Open Food Facts !
        </v-card-title>
        <v-card-text>
          <v-form @submit.prevent="submit">
            <v-text-field
              v-model="code"
              label="Entrez le code barre et appuyez sur 'Entrée'"
              type="number"
              required
            />
          </v-form>
        </v-card-text>
      </v-card>
    </v-flex>
  </v-layout>
</template>

<script>
export default {
  data() {
    return {
      code: '',
      alert: false
    }
  },
  methods: {
    submit() {
      // Ici il faut traiter la soumission
    }
  }
}
</script>
```


On va chercher le logo sur le site d’Open Food Facts. On crée un formulaire avec juste une zone de saisie pour le code barre et on prépare une alerte pour le cas où aucun produit n’est trouvé.

On peut supprimer les composants des logos devenus inutiles :




# Appel de l’API et Vuex

On va utiliser Axios pour aller chercher les informations du produit qui correspond au code barre entré. Sur le site on trouve ces informations pour l’API :

## API JSON expérimentale

Une API JSON est également disponible pour consulter les informations d'un produit. Cette API est notamment utilisée dans l'application mobile Open Food Facts pour iPhone et Android.

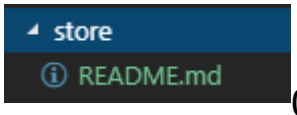
Cette API est expérimentale et peut être amenée à changer, merci de nous signaler si vous l'utilisez, afin que l'on puisse vous prévenir des éventuels changements.

URL de consultation : https://fr.openfoodfacts.org/api/v0/produit/*[code barre]*.json

Exemple : https://fr.openfoodfacts.org/api/v0/produit/3029330003533.json

Il est donc facile d'appeler cette API…

Maintenant où va-t-on placer les données reçues ? On veut les rendre disponibles pour plusieurs pages, donc plusieurs composants, dans ce cas le plus indiqué est de passer par Vuex. La bonne nouvelle c'est que Nuxt installe automatiquement Vuex ! On a un dossier **store** déjà présent :




On va créer un fichier **index.js** avec ce code :

```
import Vuex from 'vuex'

const createStore = () => {
  return new Vuex.Store({
    state: {
      data: {}
    },
    mutations: {
      setData (state, data) {
        state.data = data
      }
    }
  })
}

export default createStore
```

Juste un mutateur **setData** qui attend les informations pour les

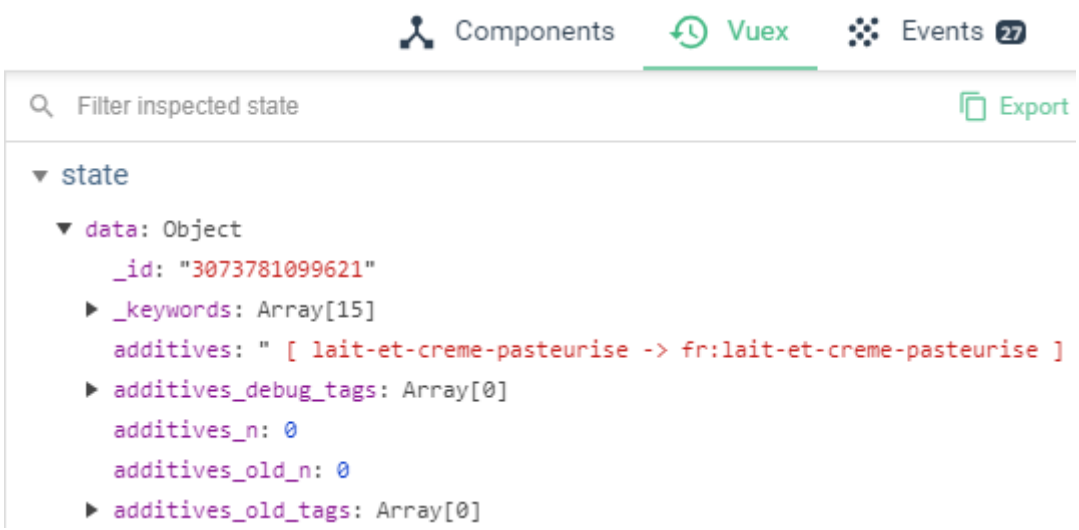
mettre dans **data**.

On va compléter la vue **index.vue** pour appeler l'API et remplir le store :

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

export default {
  data() {
    return {
      code: '',
      alert: false
    }
  },
  methods: {
    submit() {
      if(this.code != 0 && !isNaN(this.code)) {
axios.get(`https://fr.openfoodfacts.org/api/v0/produit/${this.code
}.json`)
        .then((res) => {
          if(res.data.status) {
            this.$store.commit('setData', res.data.product)
          } else {
            this.alert = true
          }
        })
      }
    }
  }
}
</script>
```

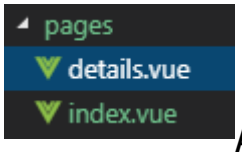
On utilise **Axios** et au retour on vérifie la donnée **status** qui nous indique si un produit a été trouvé. Si ce n'est pas le cas on change la valeur de et l'alerte devient visible :


Si le produit existe alors le **store** est rempli :




# La vue des détails

Maintenant qu'on a récupéré les informations il faut les afficher, on va créer une vue **details.vue** :




Avec ce code :

```
<template>
  <v-layout>
    <v-flex
      xs12
      sm6
      offset-sm3>
      <v-card>
        <v-img :src="$store.state.data.image_url"/>
```

```
        <v-card-title primary-title>
          <div>
                                    <div   class="headline">{{
$store.state.data.product_name }}</div>
                                  <span  class="grey--text">{{
$store.state.data.generic_name }}</span>
          </div>
        </v-card-title>

        <v-card-text class="mb-4">
          <v-list>
            <v-list-tile>
                <v-list-tile-title><strong>Marque :</strong> {{
$store.state.data.brands }}</v-list-tile-title>
            </v-list-tile>
            <v-list-tile v-if="$store.state.data.serving_size">
               <v-list-tile-title><strong>Quantité :</strong> {{
$store.state.data.serving_size }}</v-list-tile-title>
            </v-list-tile>
            <v-list-tile>
                  <v-list-tile-title><strong>Conditionnement
:</strong> {{ $store.state.data.packaging_tags.join(', ') }}</v-
list-tile-title>
            </v-list-tile>
            <v-list-tile>
               <v-list-tile-title><strong>Magasins :</strong> {{
$store.state.data.stores }}</v-list-tile-title>
            </v-list-tile>
                                           <v-list-tile   v-
if="$store.state.data.manufacturing_places">
              <v-list-tile-title><strong>Fabrication :</strong> {{
$store.state.data.manufacturing_places }}</v-list-tile-title>
            </v-list-tile>
            <v-list-tile v-if="$store.state.data.labels">
                <v-list-tile-title>{{ $store.state.data.labels
}}</v-list-tile-title>
            </v-list-tile>
            <v-list-tile>
              <img
:src="nutriscore($store.state.data.nutrition_grades)"
                width="120px"
                class="pt-4">
            </v-list-tile>
```

```
          </v-list>
        </v-card-text>

      </v-card>
    </v-flex>
  </v-layout>
</template>

<script>
export default {
  name: 'Details',
  methods: {
    nutriscore: function(value) {
                                                    return
`https://static.openfoodfacts.org/images/misc/nutriscore-${value}.
svg`
    }
  }
}
</script>
```

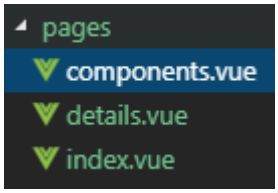
On sait que Nuxt va automatiquement créer une route **/details**. Donc dans la vue **index.vue** on appelle cette route :

```
if(res.data.status) {
  this.$store.commit('setData', res.data.product)
  this.$router.push('/details')
} else {
```

Et maintenant on obtient les détails du produit :


# Les ingrédients

On ne va pas s'arrêter en si bon chemin et on va ajouter une vue pour afficher les ingrédients :




Avec ce code :

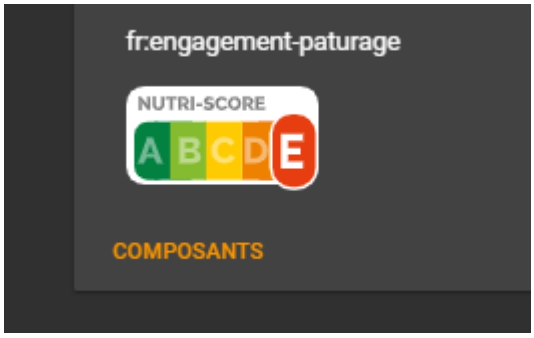
```
<template>
```

```
  <v-layout>
    <v-flex
      xs12
      sm6
      offset-sm3>
      <v-card>
        <v-img
          :src="$store.state.data.image_ingredients_url"/>

        <v-card-title primary-title>
          <div>
                                <div   class="headline">{{
$store.state.data.product_name }}</div>
                              <span  class="grey--text">{{
$store.state.data.generic_name }}</span>
          </div>
        </v-card-title>

        <v-card-text>
          <p><strong>Liste  des ingrédients :</strong></p>
          <p>{{ $store.state.data.ingredients_text_fr }}</p>
        </v-card-text>

        <v-card-actions>
          <v-btn
            flat
            to="/details"
            color="orange">Détails</v-btn>
        </v-card-actions>
      </v-card>
    </v-flex>
  </v-layout>
</template>

<script>
export default {
  name: 'Components'
}
</script>
```

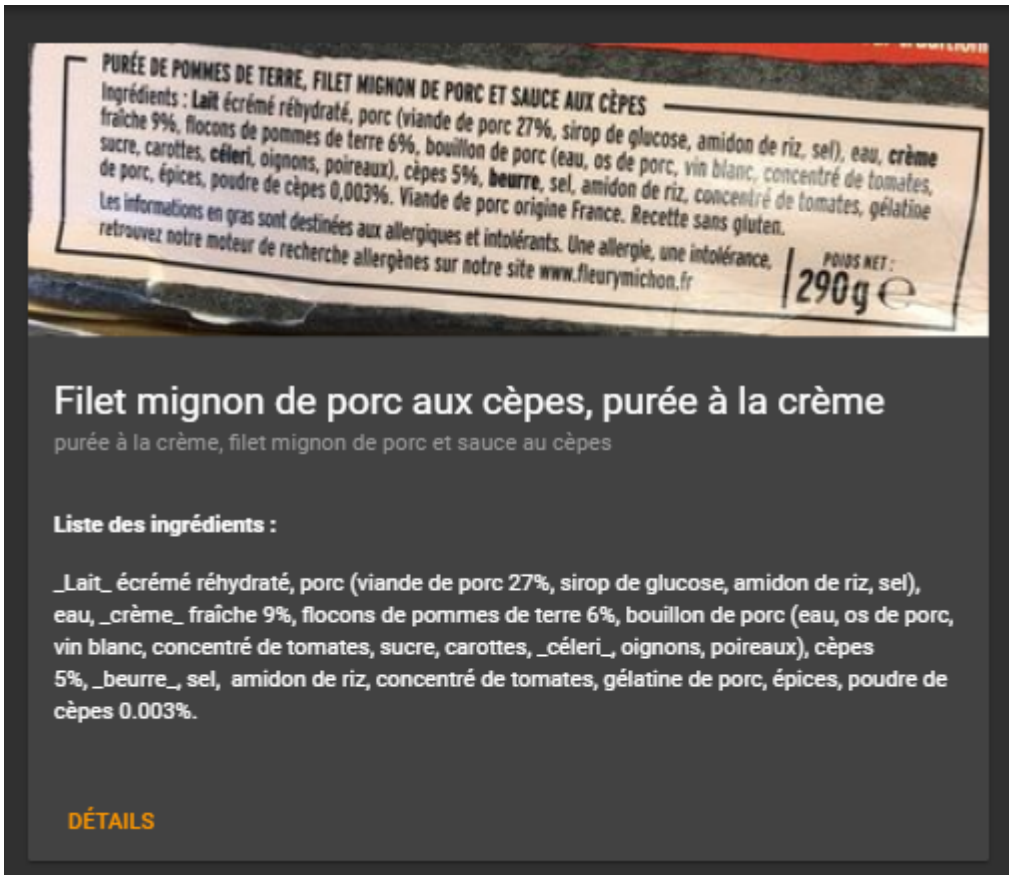
Et dans la vue **details** on ajoute un bouton pour accéder à cette nouvelle vue :

```
<v-card-actions>
```

```
  <v-btn
    flat
    to="/components"
    color="orange">Composants</v-btn>
</v-card-actions>
```




On a donc maintenant accès aux ingrédients des produits :




# Conclusion

On voit ainsi qu'il est très facile de créer des vues avec des routes qui se créent automatiquement. D'autre part Vuex nous permet d'accéder aux données à partir de n'importe quel composant de l'application, ce qui est bien pratique.