Comprendre Vue.js : Vuex

Dans le précédent article on a vu la réalisation d'un quiz. Les données (questions et réponses) ont été placées dans la propriété **data** du composant. C'est simple et efficace. Mais imaginez qu'on ait plusieurs composants qui se partagent ces informations. On n'a pas trop creusé la communication entre composants mais on a vu déjà les **props** pour transmettre une information du composant parent vers l'enfant. On verra que dans l'autre sens on utilise des événements. Ça peut devenir rapidement assez lourd. Il serait bien de disposer des données dans un endroit accessible pour tous les composants…

La solution préconisée est d'utiliser la librairie Vuex. Elle permet de stocker des données mais pas seulement ! On peut gérer très précisément comment ces données sont lues (accesseurs) et modifiées (mutateurs). On peut aussi définir certaines actions. Enfin on peut facilement déboguer avec les outils de Vue.

On va reprendre le quiz du précédent article et cette fois placer les données dans une instance de Vuex. Pour compléter on ira récupérer ces données sur un serveur.

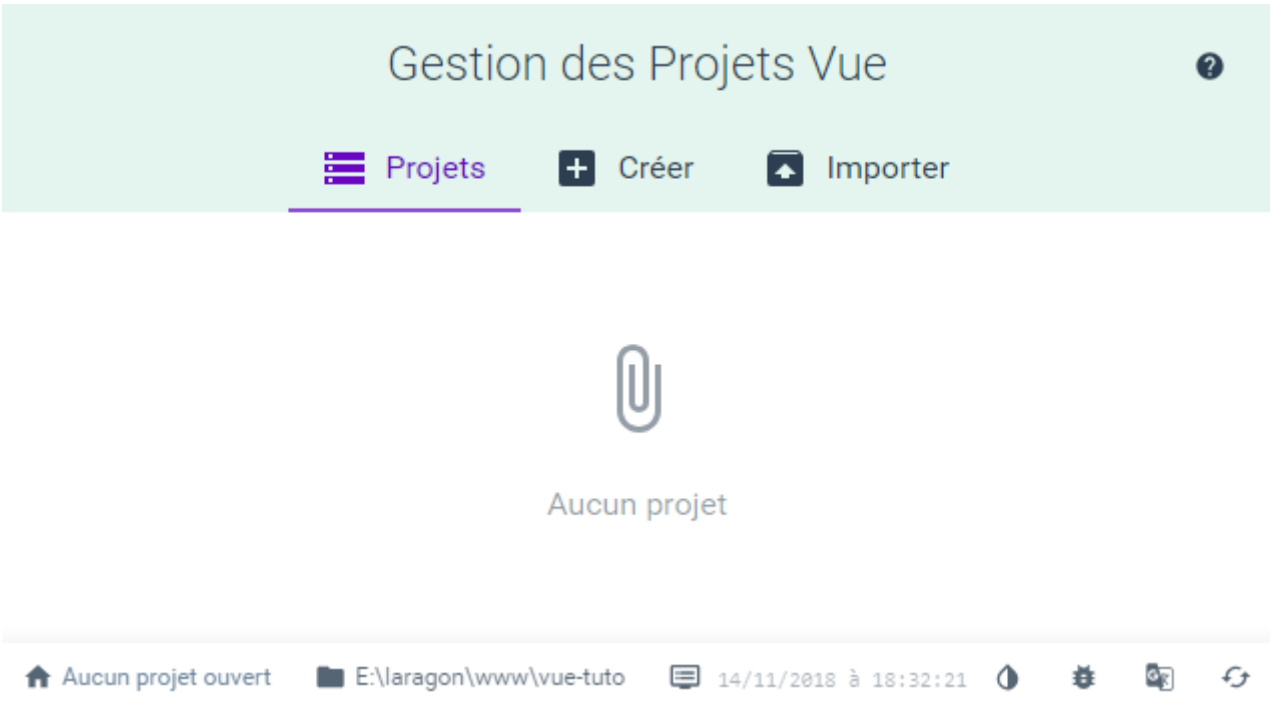
Le code final pour cet article est téléchargeable ici.

On crée le projet

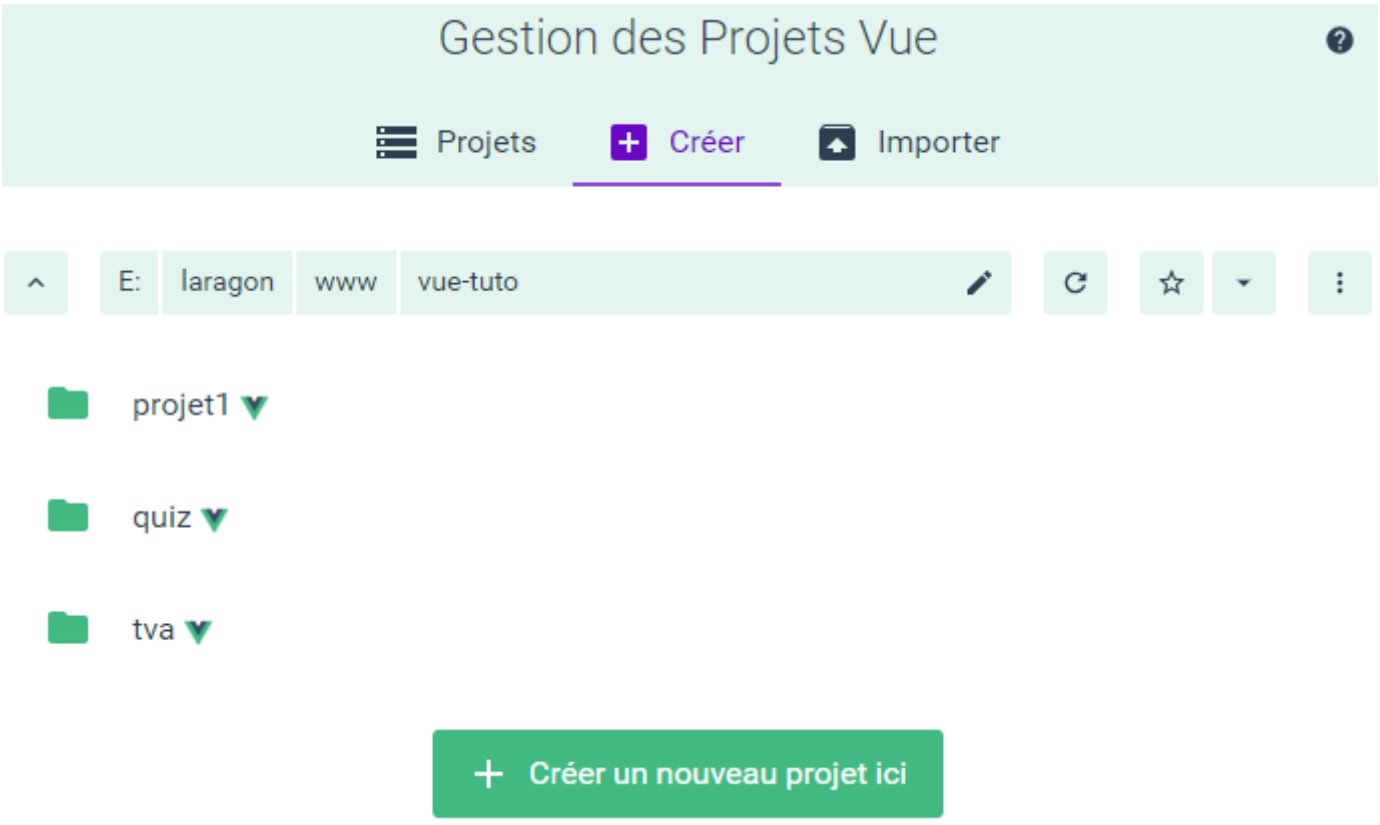
Vue Cli comporte la possibilité de passer par une interface graphique. Elle en est encore en version beta mais fonctionne bien alors nous allons l'utiliser. Placez-vous dans le dossier de travail et tapez :

```
vue ui
```

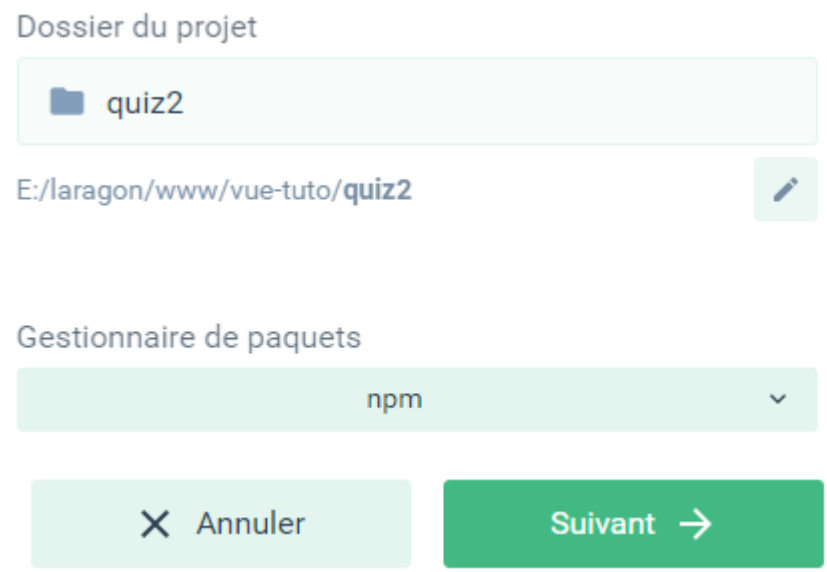
Vous devriez aboutir sur cette page dans votre navigateur :



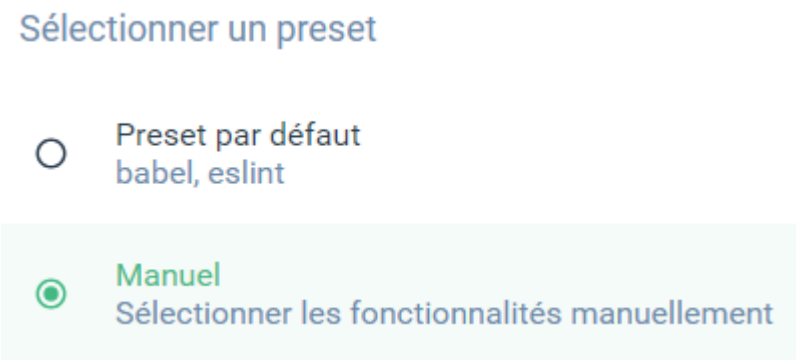
Cliquez sur Créer :



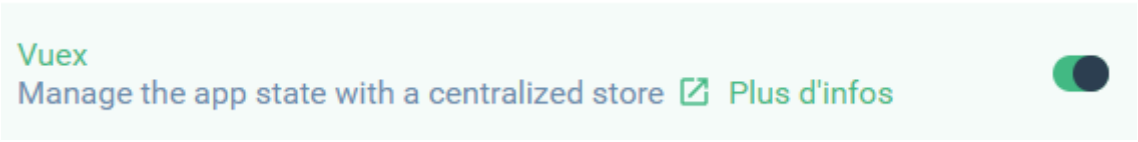
Puis sur « Créer un nouveau projet ici » si vous êtes bien dans le dossier désiré, sinon vous pouvez vous déplacer facilement :



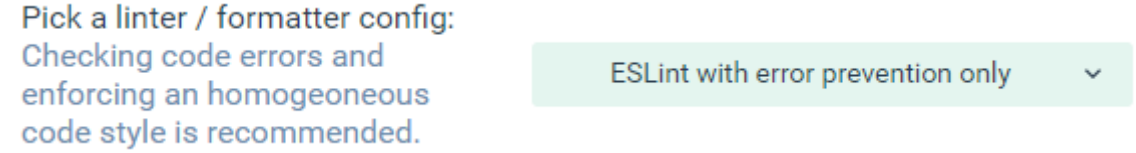
J’ai choisi le nom **quiz2** et **npm**. On clique sur « Suivant » :



Là choisissez le mode manuel parce qu’on va ajouter des choses. Cliquez sur « Suivant » :

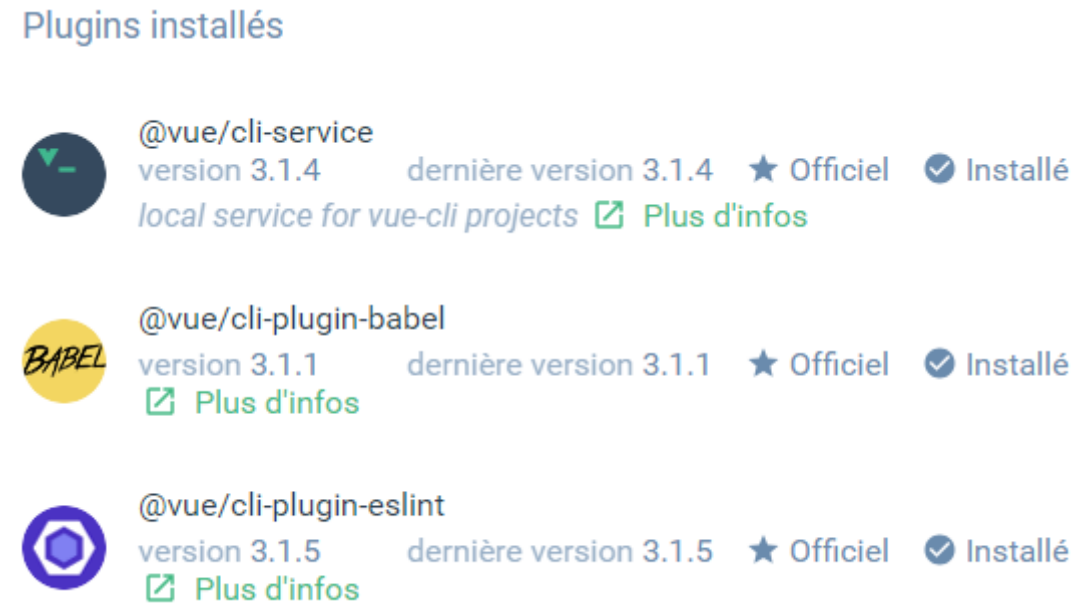


Là vous activez Vuex et vous cliquez encore sur « Suivant » :



Vous réglez le linter par défaut, puis cliquez « Créer le projet » puis « Continuer sans sauvegarder ». Et là plus qu’à attendre :

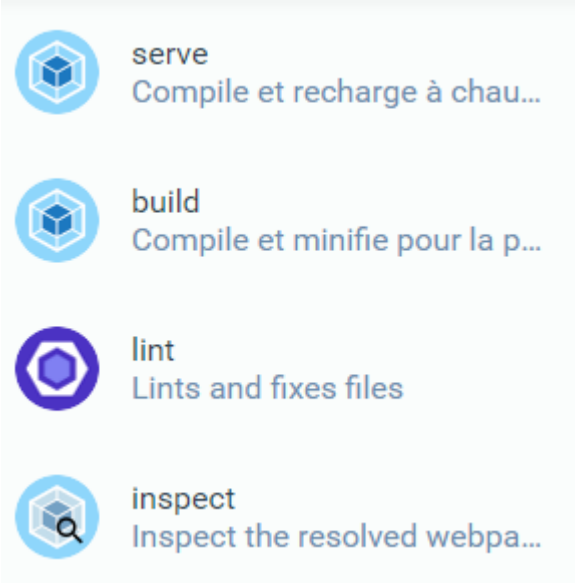
Quand c’est terminé vous pouvez voir les plugins installés :



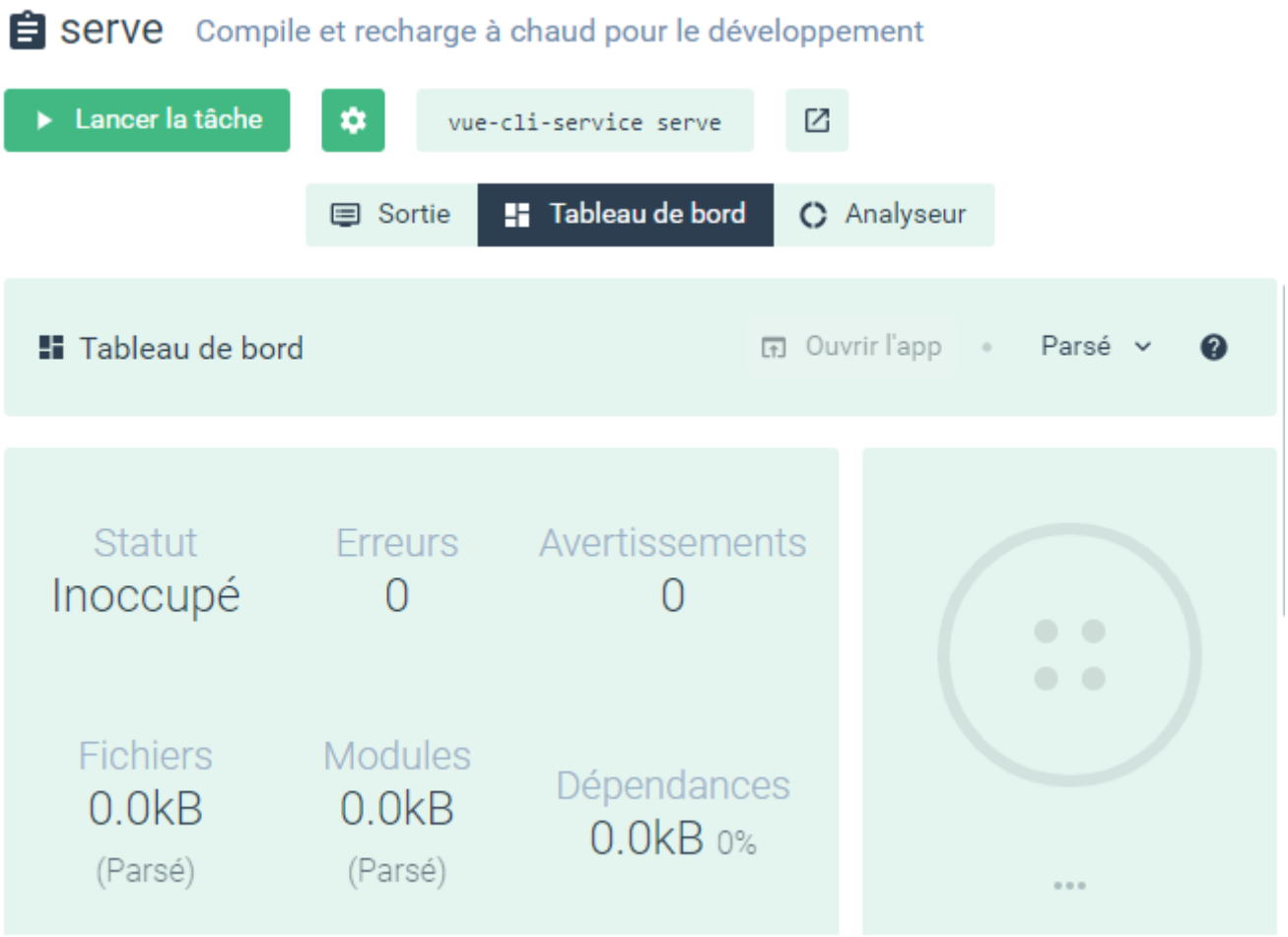
Les dépendances :

Dépendances de développement

Vous avez accès aux actions de base :



Vous pouvez utiliser le bouton **serve** :



Là vous pouvez lancer la compilation du projet…

Vous avez tous les détails sur le tableau de sortie :

```
Sortie
 23% building 115/117 modules 2 active ...node_modules\css-loader\lib
 23% building 116/117 modules 1 active ...dex=0&id=469af010&scoped=tr
 84% chunk reviving RecordIdsPlugin DONE  Compiled successfully in 41
84ms18:57:14


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.43:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

Vous arrivez évidemment sur la page d'accueil classique :

Welcome to Your Vue.js App

For guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.

Installed CLI Plugins

babel eslint

Essential Links

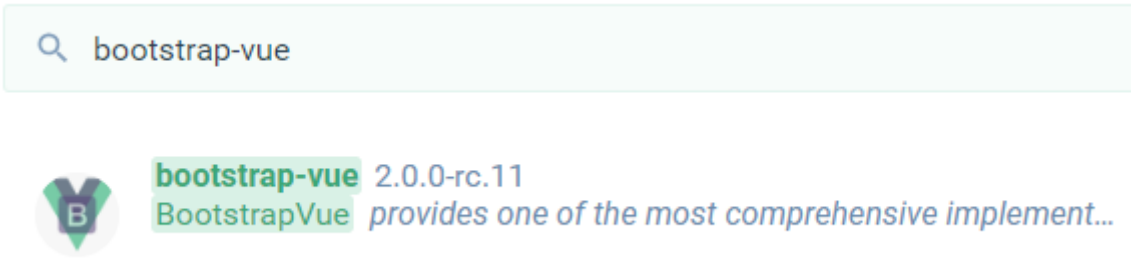
Core Docs Forum Community Chat Twitter News

Ecosystem

vue-router vuex vue-devtools vue-loader awesome-vue

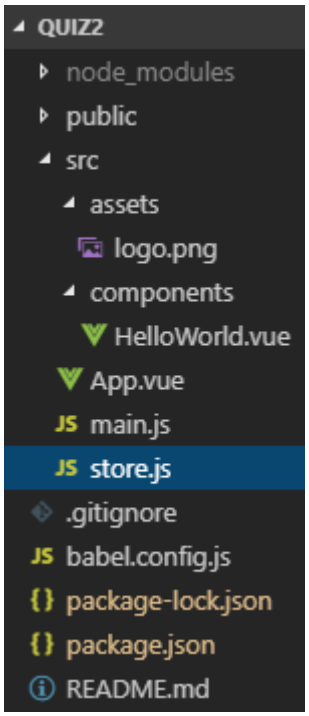
C'est une alternative à l'utilisation de la console !

On va en profiter pour ajouter aussi **Bootstrap-vue** dans les dépendances :



Le projet

Voyons un peu ce qu'on a sous le capot :



Par rapport au projet par défaut on a le fichier **store.js** en plus. C'est justement celui de Vuex :

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
```

```
  },
  mutations: {

  },
  actions: {

  }
})
```

On importe Vue et Vuex. La méthode **use** de Vue est celle qui est utilisée pour utiliser un plugin. On crée une nouvelle instance de Vuex avec **new**. Ensuite le module exporte ses données avec **state**. Il est prévu aussi des propriétés pour les mutateurs et les actions (de la même manière on peut ajouter **getters**). On a donc le squelette de notre module de gestion des données.

Maintenant si vous ouvrez **main.js** (je rappelle que c'est le fichier de configuration) :

```
import Vue from 'vue'
import App from './App.vue'
import store from './store'

Vue.config.productionTip = false

new Vue({
  store,
  render: h => h(App)
}).$mount('#app')
```

On voit qu'on importe **store** et qu'on le déclare. Il sera donc disponible dans toute l'application.

Tant qu'on est dans ce fichier on va ajouter ce qu'il faut pour **Bootstrap-vue** comme on l'a fait dans le précédent article :

```
import Vue from 'vue'
import App from './App.vue'
import store from './store'

import BootstrapVue from 'bootstrap-vue'
Vue.use(BootstrapVue);

import 'bootstrap/dist/css/bootstrap.css'
```

```
import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.config.productionTip = false

new Vue({
  store,
  render: h => h(App)
}).$mount('#app')
```

Les données

J’ai dit que les données seront gérées par Vuex alors on met le questionnaire dans **store.js** :

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    questions: [
      {
        question: "Quel révolutionnaire et grand orateur a déclaré en 1792 : “De l’audace, encore de l’audace, toujours de l’audace.”",
        answers: [
          'Desmoulin',
          'Danton',
          'Robespierre',
          'Saint Just'
        ],
        ok: 1
      },
      {
        question: "Dans quel pays peut-on trouver le mont Elbrouz ?",
        answers: [
          'Russie',
          'Azerbaïdjan',
          'Géorgie',
          'Iran'
```

```
        ],
        ok: 0
      },
      {
        question: "Qui a dit “Ich bin ein Berliner” ?",
        answers: [
          'Bismarck',
          'Reagan',
          'De Gaulle',
          'Kennedy'
        ],
        ok: 3
      }
    ]
  },
  mutations: {

  },
  actions: {

  }
})
```

Le composant App

On supprime le composant **HelloWorld** qui ne nous servira pas.

Dans le composant **App** on recopie à l’identique le template du dernier article :

```
<template>
  <div id="app" class="container">
    <h1 class="mb-4">Un petit quiz</h1>
    <b-alert v-if="fin" show>Votre score est : {{ score }} / {{
questions.length }}</b-alert>
    <b-card :header="questions[index].question"
            header-tag="header">
      <b-list-group>
        <b-list-group-item
          button
          v-for="(item, index) in questions[index].answers"
          :key="item.id"
```

```
          @click="action(index)"
          :variant="variants[index]">
          {{ item }}
        </b-list-group-item>
      </b-list-group>
            <b-button  v-if="fin"  @click="recommencer"
class="mt-4">Recommencer !</b-button>
       <b-button v-if="voirReponse && !fin" @click="continuer"
class="mt-4">Continuer !</b-button>
    </b-card>
  </div>
</template>
```

On conserve le style par défaut.

Dans le script on va utiliser ce code :

```
<script>
export default {
  name: 'app',
  data: function () {
    return {
      fin: false,
      index: 0,
      score: 0,
      variants: [...Array(4)],
      voirReponse: false,
    }
  },
  methods: {
    action: function(index) {
      // Test bonne réponse
      if(index == this.questions[this.index].ok) {
        this.score++;
      } else {
        this.variants[index] = 'danger';
      }
      this.voirReponse = true;
      this.variants[this.questions[this.index].ok] = 'success';
      if(this.index == this.questions.length - 1) {
        this.fin = true;
      }
    },
    recommencer: function() {
```

```
      this.voirReponse = this.fin = this.index = this.score = 0;
      this.variants = [...Array(4)];
    },
    continuer: function() {
      this.voirReponse = false;
      this.variants = [...Array(4)];
      this.index++;
    }
  },
  computed: {
    questions () {
      return this.$store.state.questions;
    }
  }
}
</script>
```

On a comme différences :

- la suppression des questions dans **data**
- l’ajout de la propriété calculée (computed) **questions** :

```
computed: {
  questions () {
    return this.$store.state.questions;
  }
}
```

On récupère les données de Vuex avec **this.$store.state**. C’est très simple !

Et maintenant on se retrouve avec le même fonctionnement que pour le précédent article :

On s’est donné bien du mal pour avoir le même résultat

Un web service RESTful

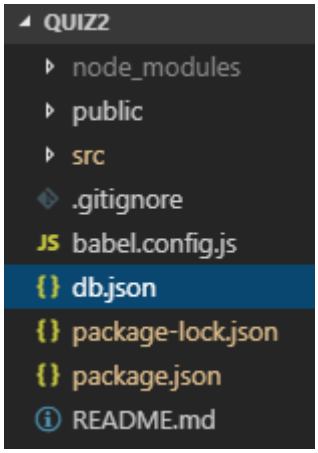
On va maintenant créer un web service pour nous fournir les questions, ce qui sera déjà plus réaliste. On pourrait le créer en PHP mais puisqu’on est dans Javascript autant continuer dans le style.

On va installer globalement json-server :

```
npm install -g json-server
```

On va ainsi pouvoir ajouter facilement un web service RESTful.

Il suffit de créer un fichier **db.json** (ou un autre nom quelconque) à la racine du projet :



Avec les données bien formées en JSON :

```
{
  "questions" : [
    {
      "question": "Quel révolutionnaire et grand orateur a déclaré en 1792 : “De l’audace, encore de l’audace, toujours de l’audace.”",
      "answers": [
        "Desmoulin",
        "Danton",
        "Robespierre",
        "Saint Just"
      ],
      "ok": 1
    },
    {
      "question": "Dans quel pays peut-on trouver le mont Elbrouz ?",
      "answers": [
        "Russie",
        "Azerbaïdjan",
        "Géorgie",
        "Iran"
      ],
      "ok": 0
    },
    {
      "question": "Qui a dit “Ich bin ein Berliner” ?",
      "answers": [
        "Bismarck",
        "Reagan",
        "De Gaulle",
```

```
        "Kennedy"
      ],
      "ok": 3
    }
  ]
}
```

Ensuite on lance le serveur :

```
json-server db.json
```

```
  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3000/questions

  Home
  http://localhost:3000
```

On peut maintenant aller chercher les questions à l'adresse http://localhost:3000/questions.

Axios

Vue ne propose rien pour faire des requêtes HTTP, il faut donc utiliser une autre librairie. Classiquement c'est Axios qui est utilisé. On commence par l'installer (vous pouvez aussi passer par l'interface graphique) :

```
npm i axios
```

Et on va l'importer dans **store.js** :

```
import Vue from 'vue'
import Vuex from 'vuex'
import Axios from 'axios'
```

On va modifier le code pour envoyer la requête HTTP et récupérer la réponse :

```
export default new Vuex.Store({
  state: {
    questions: []
  },
  mutations: {
    setData(state, data) {
      state.questions = data;
    }
  },
  actions: {
    async getData(context) {
                                let     data     =     (await
Axios.get('http://localhost:3000/questions')).data;
      context.commit("setData", data);
    }
  }
})
```

On commence par supprimer les données qu'on avait mises dans **questions**.

On crée un mutateur **setData** pour renseigner les questions. C'est en effet le seul moyen pour modifier des données avec Vuex. On reçoit **state** comme premier argument et on peut passer d'autres arguments, ici on passe les données.

On crée une action asynchrone (async) et on attend (await) la réponse. Lorsqu'on la reçoit on invoque le mutateur avec **contect.commit**.

Maintenant que c'est en place il ne nous reste plus qu'à appeler cette fonction **getData**.

Les étapes d'une application

Une application Vue passe par 3 étapes :

- **created** (initialisations)
- **mounted** (création du DOM)
- **destroyed** (suppression de l'application)

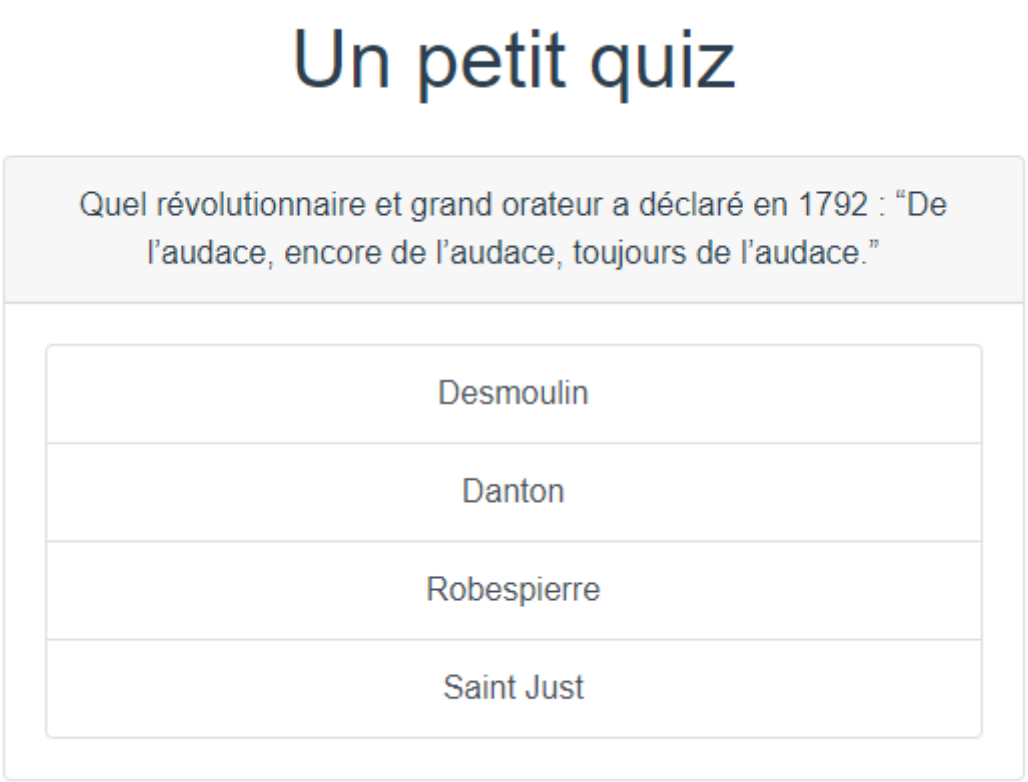
On dispose de 6 événements :

- beforeCreated
- created
- beforeMounted
- mounted
- beforeDestroyed
- destroyed

On va utiliser l'événement **created** dans le composant App :

```
<script>
export default {
  ...
  created() {
    this.$store.dispatch('getData');
  }
}
</script>
```

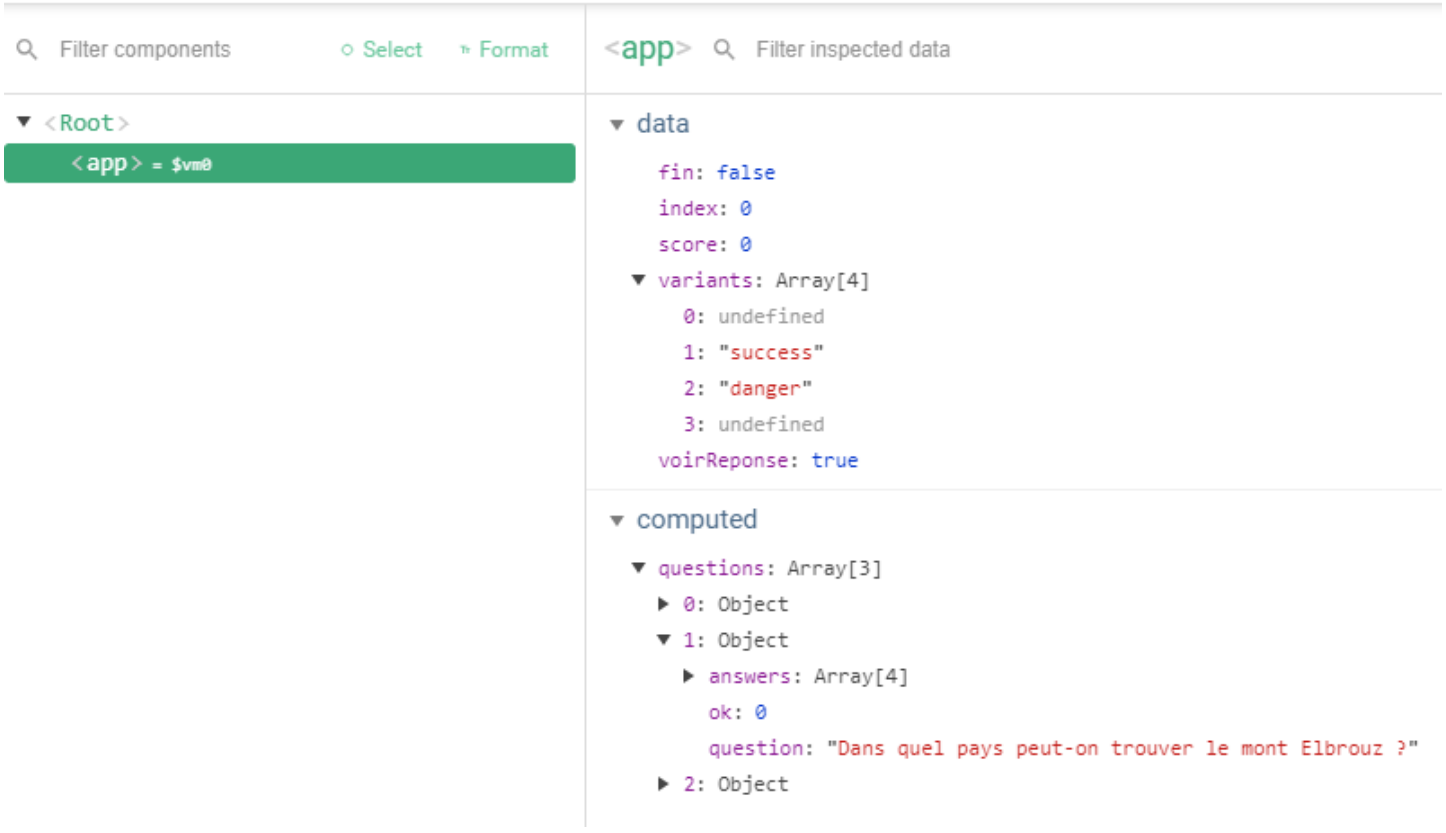
Là on appelle la méthode **getData** avec **dispatch** et normalement ça devrait fonctionner



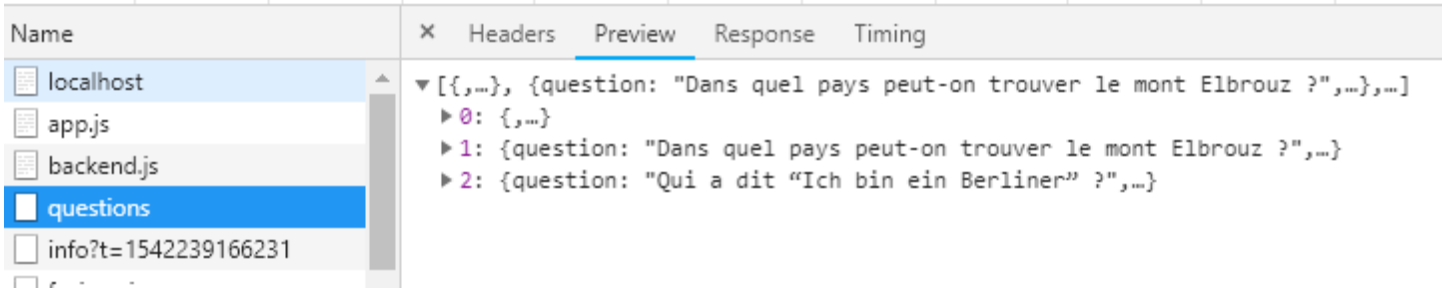
Les outils

Vous pouvez observer ce qui se passe dans Vuex avec les outils de développement :

Et évidemment voir tout ce qui se passe aussi dans le composant :



Et suivre les requêtes HTTP :



Conclusion

Dans cet article on a vu comment mettre en œuvre Vuex pour centraliser les données dans une application Vue et ainsi les rendre accessibles à partir de tous les composants. On a également vu l’utilisation d’Axios pour récupérer des données sur un serveur. On a également utilisé une interface graphique de Vue Cli comme alternative à la console.