Comprendre Vue.js : un quiz

Poursuivons notre découverte de Vue.js avec une nouvelle petite application. Cette fois il va être question d'un quiz avec donc question et possibilités de réponses, ce qui va nous permettre de voir comment Vue gère les listes de données.

Le code final pour cet article est téléchargeable ici.

On crée le projet

Toujours avec le Cli on crée un nouveau projet :

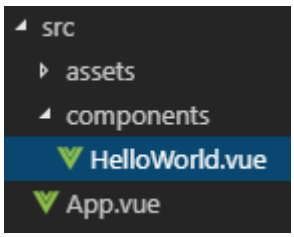
vue create quiz

Vérifiez que le projet est correctement créé en lançant le serveur :

npm run serve

Si vous avez la page d'accueil alors tout va bien on peut poursuivre…

Cette fois on va commencer par supprimer le composant **HelloWorld** :



Et pour que ça fonctionne encore on va modifier le code du composant **App** :

```
<template>
  <div id="app">
    <h1>Un quiz ?</h1>
  </div>
</template>

<script>
export default {
```

```
  name: 'app'
}
</script>

<style>
#app {
  Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

J’ai supprimé les références au composant **HelloWorld**, le logo, et ajouté un titre :

Un quiz ? Bootstrap

On va encore ajouter Bootstrap pour gérer l’apparence mais cette fois, au lieu de charger la librairie directement, on va faire appel à une librairie qui a transformé les composants de Bootstrap en composants Vue. Il existe des template pour avoir directement une application toute prête avec Vue Cli mais on va plutôt détailler l’installation.

D’abord on installe la librairie :

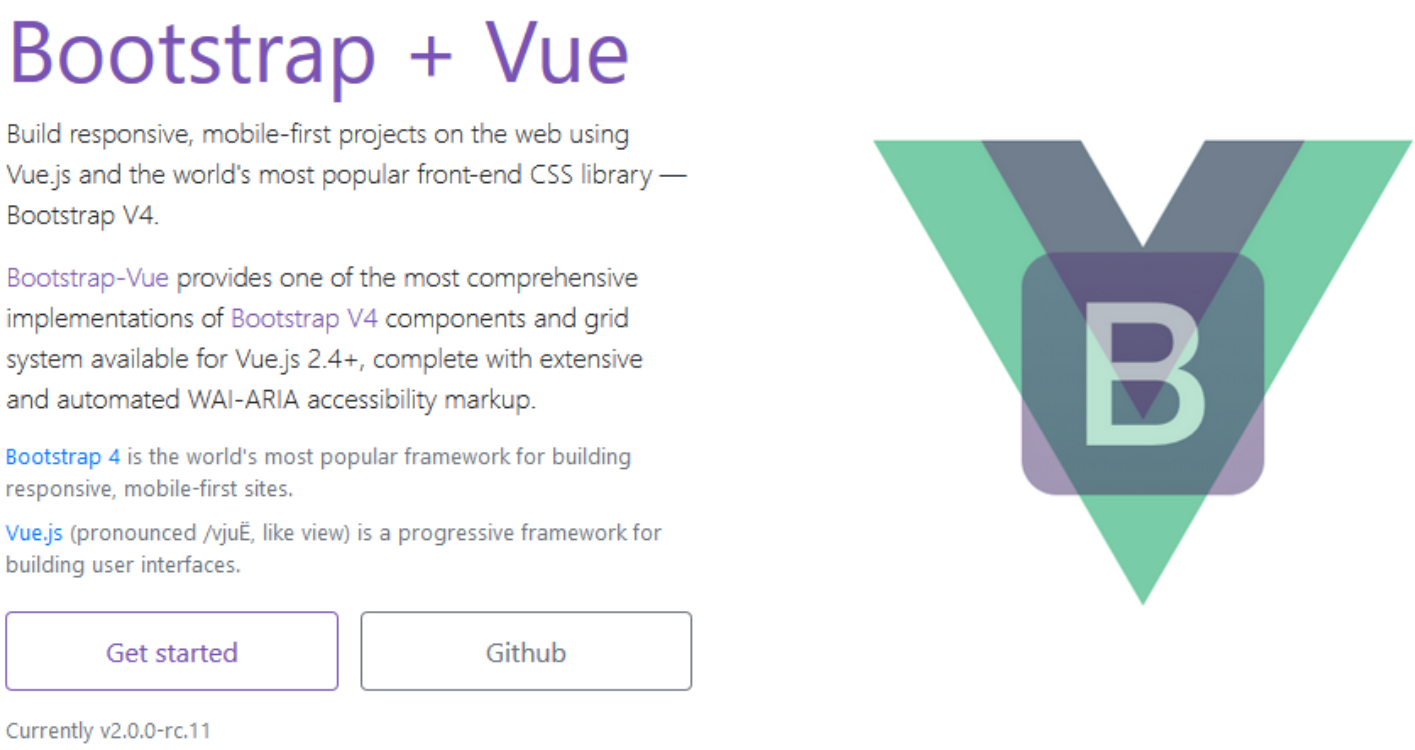
```
npm i bootstrap-vue
```

Et on charge **main.js** de charger tout ça :

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

import BootstrapVue from 'bootstrap-vue'

Vue.use(BootstrapVue);

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

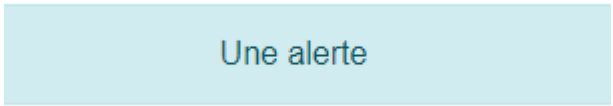
```
import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.config.productionTip = false

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

Vous trouvez une documentation détaillée de la librairie ici :



Pour vous assurez que ça fonctionne ajoutez une alerte dans le template :

```
<b-alert show>Une alerte</b-alert>
```



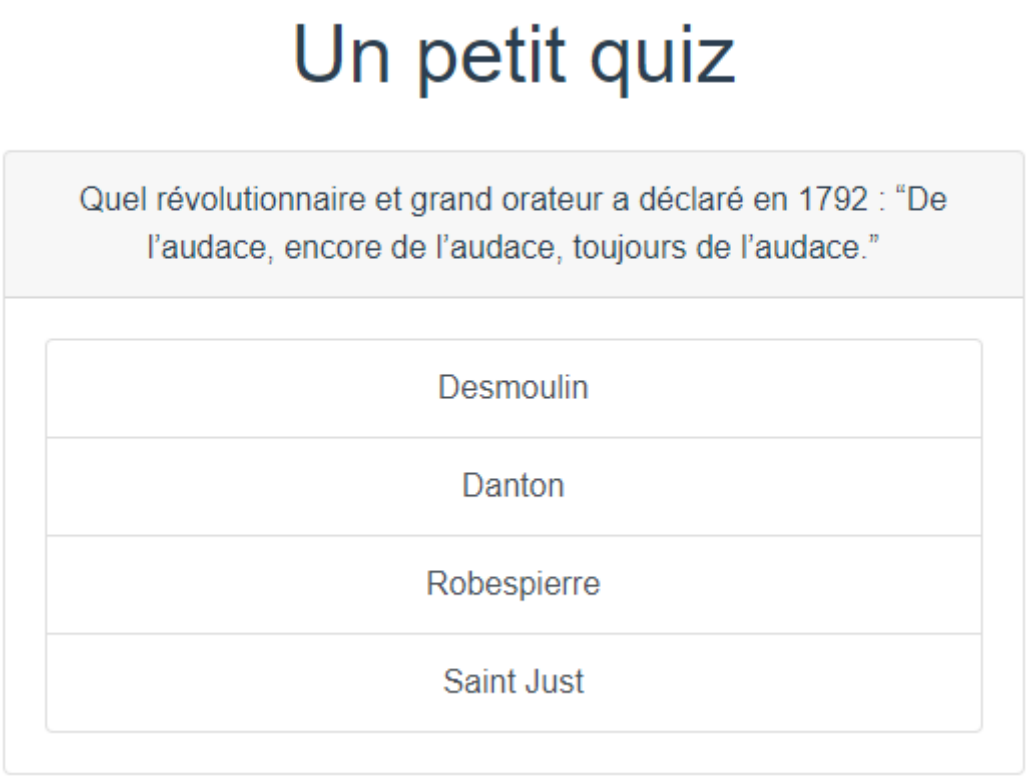
Vous l’avez bien ? Alors c’est parfait on peut poursuivre…

On va faire un petit essai de look pour notre quiz avec ce template :

```
<template>
  <div id="app" class="container">
    <h1 class="mb-4">Un petit quiz</h1>
     <b-card header="Quel révolutionnaire et grand orateur a
```

```
déclaré en 1792 : “De l’audace, encore de l’audace, toujours de l’audace.”"
            header-tag="header">
      <b-list-group>
        <b-list-group-item button>Desmoulin</b-list-group-item>
        <b-list-group-item button>Danton</b-list-group-item>
        <b-list-group-item button>Robespierre</b-list-group-item>
        <b-list-group-item button>Saint Just</b-list-group-item>
      </b-list-group>
    </b-card>
  </div>
</template>
```

On obtient cet aspect :



Vous voyez que cette librairie nous permet de mettre en œuvre Bootstrap avec une syntaxe simple et lisible.

Maintenant il faut nous occuper de la gestion de ce quiz. On va avoir évidemment plusieurs questions qui vont se succéder et à l’arrivée le bilan des réponses.

Les données

Pour ne pas me compliquer la vie je vais prendre 3 questions d'un site de culture générale :

```
<script>
export default {
  name: 'app',
  data: function () {
    return {
      index: 0,
      score: 0,
      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
        }
      ]
    }
  },
}
</script>
```

Donc on va avoir :

- **index** : pour savoir où on en est des questions
- **score** : pour savoir où en est le score
- **questions** : les questions à poser avec pour chacune :
  - la question
  - les réponses possibles
  - l'index de la bonne réponse

Gestion d'une liste d'éléments

Vue propose la directive **v-for** pour gérer les listes de données. c'est une directive puissante et simple qu'on va utiliser pour afficher les réponses. En ce qui concerne le texte de la question c'est une simple liaison. Voici le template modifier pour afficher la question et ses réponses :

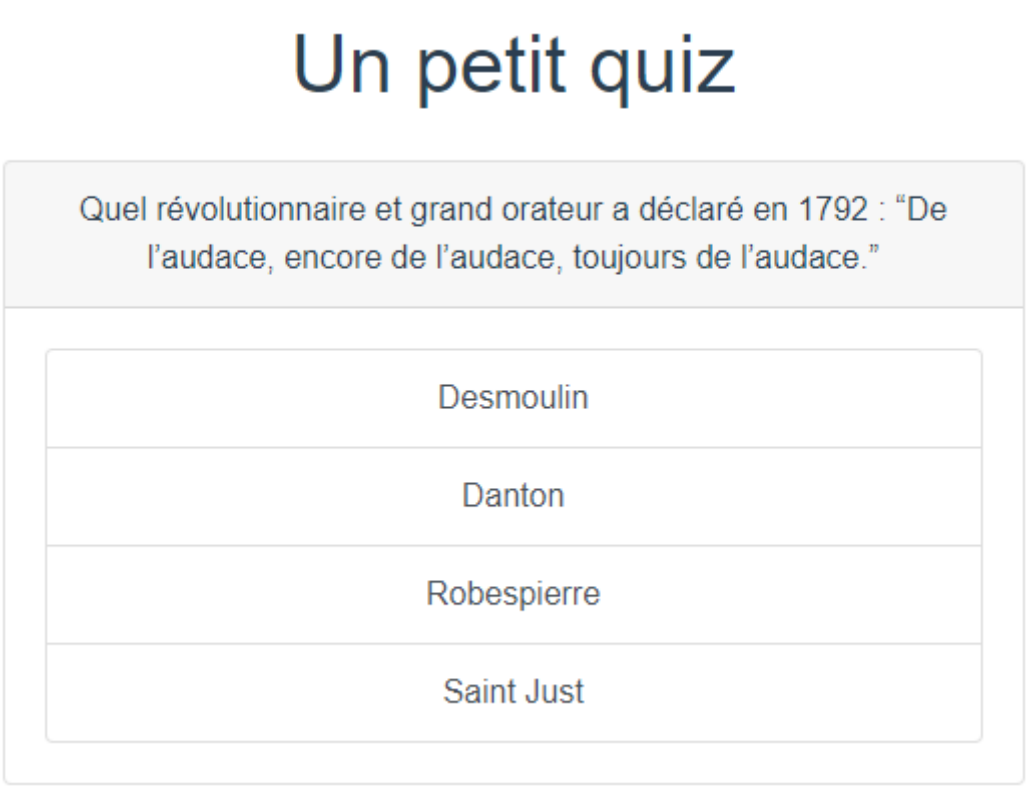
```
<template>
  <div id="app" class="container">
    <h1 class="mb-4">Un petit quiz</h1>
    <b-card :header="questions[index].question"
            header-tag="header">
      <b-list-group>
        <b-list-group-item
          button
          v-for="item in questions[index].answers"
          :key="item">
          {{ item }}
        </b-list-group-item>
      </b-list-group>
    </b-card>
```

```
  </div>
</template>
```

Remarquez comment on renseigne un attribut avec une liaison sur le nom de l'attribut, ici **header** pour la question.

Quant aux réponses possibles la directive **v-for** fait tranquillement le travail. Il est obligatoire ici d'ajouter une clé (**key**) pour avoir un identifiant unique pour chaque item de la boucle.

On obtient le même affichage que précédemment mais maintenant créé dynamiquement :



Maintenant on aura une mise à jour automatique de la question et de ses réponses en changeant seulement la valeur de **index**.

Action

Quand on clique sur une réponse il faut :

- évaluer cette réponse pour savoir si c'est la bonne et dans ce cas incrémenter le score

- passer à la question suivant et s'il n'y en a plus afficher les résultats

On va équiper les réponses d'un événement sur le clic :

```
<b-list-group-item
  button
  v-for="(item, index) in questions[index].answers"
  :key="item.id"
  @click="action(index)">
  {{ item }}
</b-list-group-item>
```

Remarquez que dans la directive **v-for** j'ai ajouté un **index** pour identifier les réponses. Comme ça on peut transmettre cet index comme argument à la méthode **action**.

On crée donc la propriété pour les méthodes dans le composant avec la méthode **action** :

```
<script>
export default {
  name: 'app',
  data: function () {
    return {
      fin: false,
      ...

  methods: {
    action: function(index) {
      // Test bonne réponse
      if(index == this.questions[this.index].ok) {
        this.score++;
      }
      // Test fin de quiz
      if(this.index == this.questions.length - 1) {
        this.fin = true;
      } else {
        this.index++;
      }
    }
  }
}
</script>
```

J’ai aussi ajouté une variable **fin** qui est **false** au départ et qui devient **true** à la fin du quiz.

On finit en ajoutant une barre d’alerte dans le template :

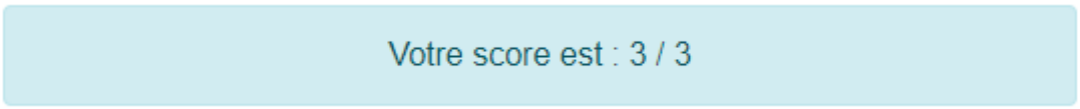
```
<h1 class="mb-4">Un petit quiz</h1>
<b-alert v-if="fin" show>Votre score est : {{ score }} / {{ questions.length }}</b-alert>
```

On la contrôle avec une directive **v-if** et la variable fin qu’on a **créée** plus haut.

On lance pour voir si ça fonctionne…

Après ma première réponse je passe bien à la réponse suivante :

Et à la fin j’ai bien l’alerte avec mon score :



Comme je connaissais les réponses j’ai une bonne note

Gestion du quiz

Le quiz fonctionne mais arrivé à la fin on ne peut plus rien faire si ce n’est relancer l’application pour le refaire. ce n’est pas très élégant alors on va ajouter un bouton pour proposer de

recommencer :

```
        <b-button    v-if="fin"    @click="recommencer"
class="mt-4">Recommencer !</b-button>
</b-card>
```

On utilise la même directive **v-if** que pour l'alerte parce que la condition est la même.

On intercepte l'événement clic en appelant une méthode **recommencer**. On ajoute cette méthode :

```
methods: {
  ...
  recommencer: function() {
    this.fin = this.index = this.score = 0;
  },
```

Là on se contente de réinitialiser les variables et c'est reparti !

Les bonnes réponses

On peut améliorer notre quiz en indiquant à chaque fois la bonne réponse. Mais du coup il nous faut aussi un bouton pour passer à la question suivante…

On commence par ajouter deux variables dans le composant :

```
data: function () {
  return {
    ...
    variants: [...Array(4)],
    voirReponse: false,
```

On initialise un tableau avec le nombre de questions. La variable **voirReponse** nous permettra de savoir si on en est à choisir une réponse ou à l'affichage de la réponse.

On réorganise les méthodes et on ajoute une méthode **continuer** :

```
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++;
  }
```

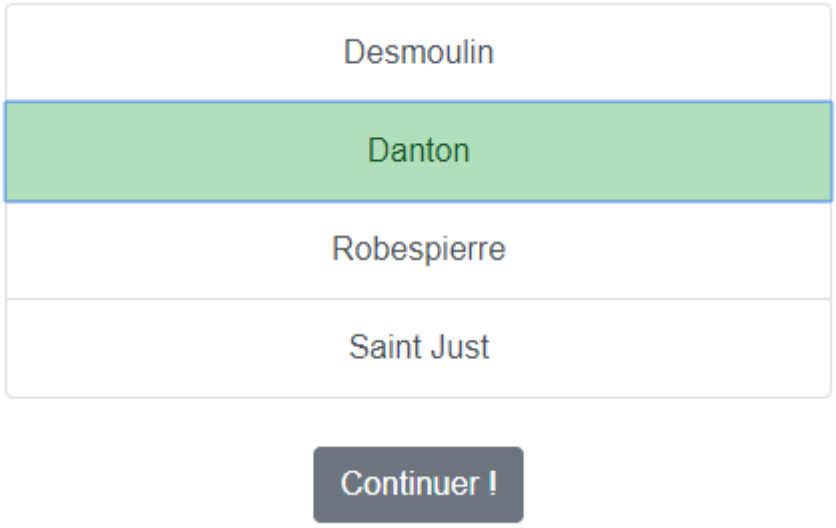
```
}
```

Il ne nous reste plus qu’à prévoir un bouton pour continuer :

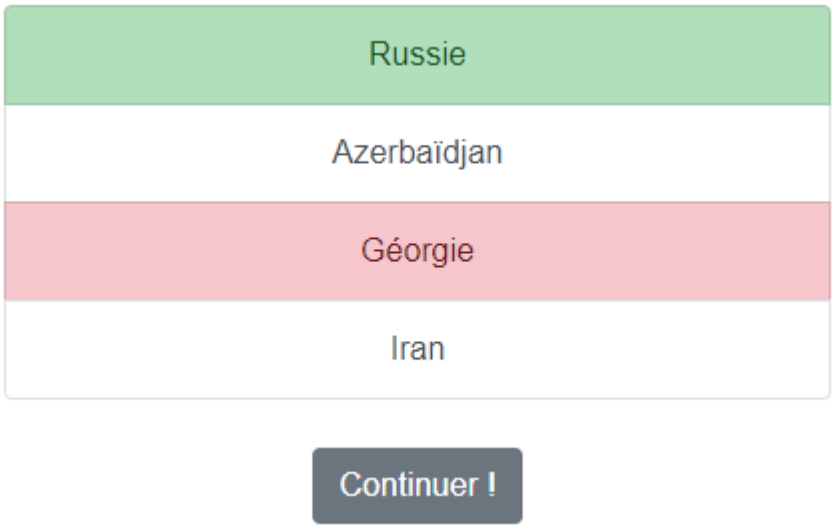
```
<b-button v-if="voirReponse && !fin" @click="continuer" class="mt-4">Continuer !</b-button>
```

Et ça devrait rouler !

Si on choisit la bonne réponse elle apparaît en vert et on a le bouton pour passer à la question suivante :



Si on a une mauvaise réponse elle apparaît en rouge et la bonne apparaît en vert :



Ça commence à avoir de l’allure !

Conclusion

On a vu pas mal de choses dans cet article : l’utilisation d’un librairie pour utiliser facilement Bootstrap, comment gérer une liste de données, comment rendre conditionnel l’affichage d’éléments, comment jouer avec les liaisons et les événements…