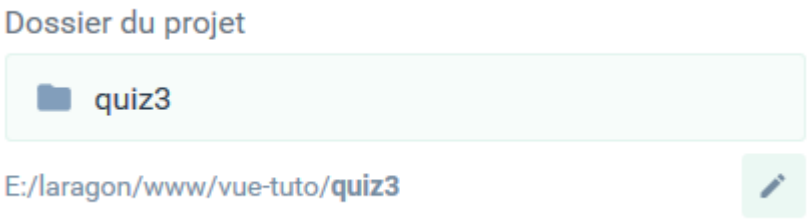
# Comprendre Vue.js : le routeur

Dans cet article je vous propose de prolonger l'exemple du quiz en l'enrichissant. Jusque là on a proposé un seul questionnaire, ce qui est limité. Dans une situation réaliste on aurait le choix entre plusieurs questionnaires. Du coup on va se  retrouver avec deux page : une pour le choix du questionnaire et l'autre pour répondre aux questions. Ça nous donne l'occasion de découvrir le routeur de Vue qui est bien pratique !

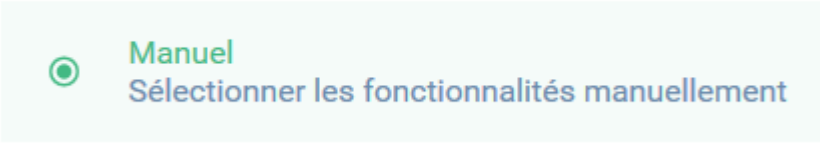
Le code final pour cet article est téléchargeable ici.

# On crée le projet

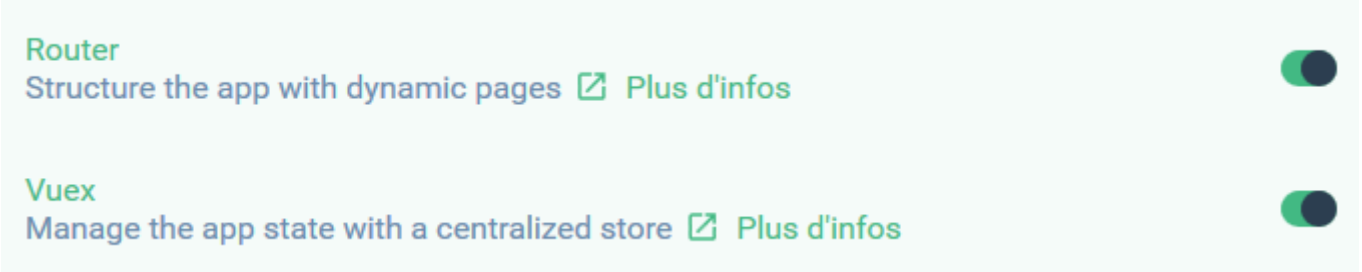
On va encore utiliser Vue Cli pour créer le projet. Je ne vais pas trop détailler à nouveau le processus parce que je l'ai fait dans les précédents articles. Par exemple avec l'interface graphique, on va choisir le nom **quiz3** :




Choisir le mode manuel :

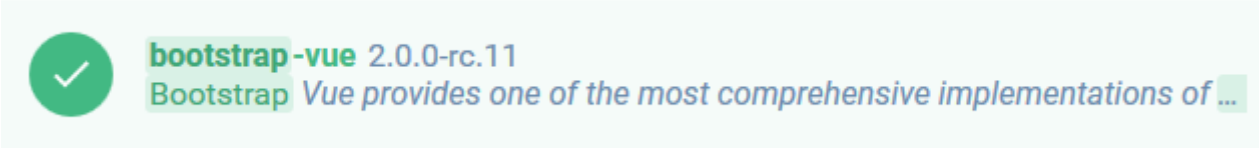



Là en plus de babel et du linter on prend Vuex et le routeur :



On crée le projet en prenant une configuration de base pour le linter dont je ne vous ai pas encore parlé.

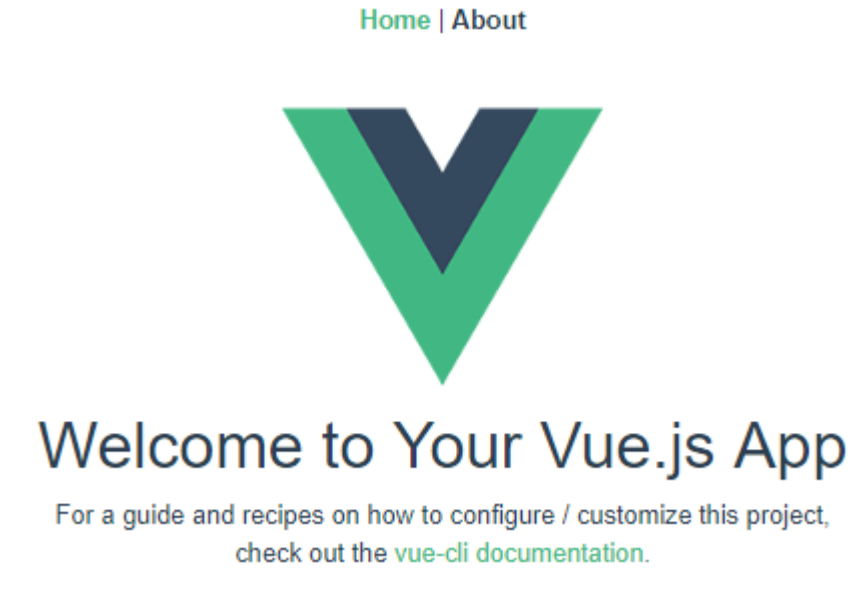
Ensuite on ajoute Bootstrap Vue comme dans les précédents articles :



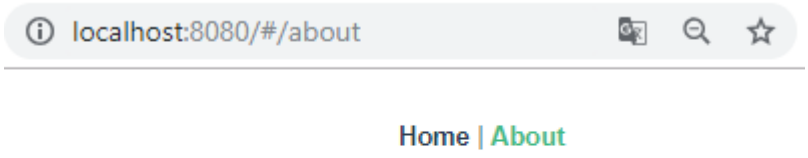

Vous devez donc avoir ces 4 dépendances principales :


En ce qui concerne le serveur pour l'API vous avez dû l'installer globalement dans le précédent article. Il doit donc être encore disponible.

On a la page d'accueil habituelle mais avec deux liens supplémentaires :



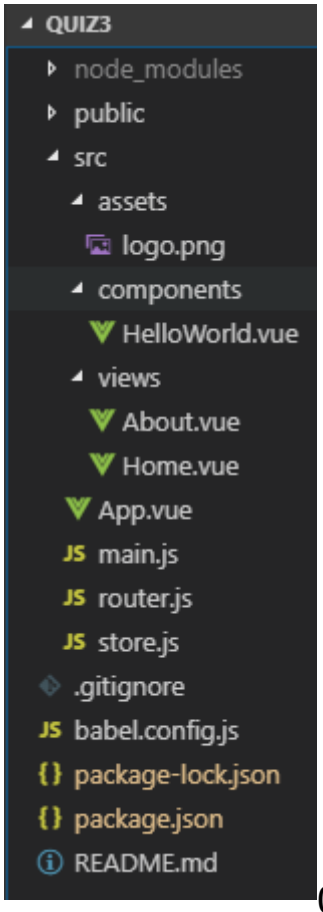

Si on clique sur **about** :




Notez la valeur de l'url. Vous avez peut-être noté que pour la page d'accueil l'url est : **http://localhost:8080/#/**.

Le routeur de Vue sait quel composant utiliser selon la valeur dans l'url qui suit **#/**.

Pour le code on a cette structure :



On a quelques nouveautés.

Un nouveau fichier apparaît, **router.js**, et comme vous devez vous en douter c'est là qu'on va avoir le code du routeur.

On voit aussi un nouveau dossier, **views**, avec deux fichiers. Alors ce sont des composants comme les autres, la seule différence réside dans une question d'organisation. On pourrait se contenter de tout mettre dans le dossier **components**. Mais on nous propose cette architecture pour avoir un composant pour chacune des urls du routeur dans le dossier **views**. Ensuite ces composants peuvent très bien utiliser d'autres composants qui eux sont dans le dossier **components**. On va pour cet article se plier à cette organisation même si c'est un peu lourd par rapport à la simplicité de l'application.

# La configuration

Regardons le code du fichier **main.js** :

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

```
import store from './store'

Vue.config.productionTip = false

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

On retrouve ce qu'on a déjà vu avec en plus le routeur importé et déclaré de la même manière que Vuex (store).

Comme on utilise Bootstrap Vue on va ajouter le code que vous connaissez bien maintenant :

```
...

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

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

...
```

Et on est bon pour la configuration !

# Le routeur

Vous pouvez lire la documentation complète ici.

Pour le routeur c'est la découverte, voyons le code de **router.js** :

```
import Vue from 'vue'
import Router from 'vue-router'
import Home from './views/Home.vue'

Vue.use(Router)

export default new Router({
  routes: [
    {
```

```
      path: '/',
      name: 'home',
      component: Home
    },
    {
      path: '/about',
      name: 'about',
      // route level code-splitting
       // this generates a separate chunk (about.[hash].js) for
this route
      // which is lazy-loaded when the route is visited.
       component: () => import(/* webpackChunkName: "about" */
'./views/About.vue')
    }
  ]
})
```

Pour chaque route on définit : le chemin (ce qui apparaît après le **#**), optionnellement un nom, et le composant à utiliser. Je n'évoquerai pas la particularité de la deuxième route pour rester simple (on diffère le chargement du composant). On pourrait écrire :

```
{
  path: '/about',
  name: 'about',
  component: About
}
```

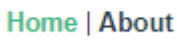
En important évidemment le composant…

Regardez le code du template dans le composant principal **App.vue** :

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
    <router-view/>
  </div>
</template>
```

On voit qu'on utilise deux composants :

- **router-link** : génère par défaut une balise **<a>,** la propriété **to** prend la valeur de la route, c'est ce qui génère ces deux liens :




- **router-view** : le composant sélectionné par la route est généré à cet emplacement, pour la page d'accueil (route '/') c'est le composant **Home** qui est généré (dans le dossier **views**) :

```
<template>
  <div class="home">
    <img alt="Vue logo" src="../assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
// @ is an alias to /src
import HelloWorld from '@/components/HelloWorld.vue'

export default {
  name: 'home',
  components: {
    HelloWorld
  }
}
</script>
```

On a affichage du logo et appel du composant **Helloworld** qui lui affiche tout le reste.

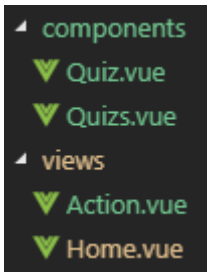
Pour la route **about** c'est le composant **About** qui est tout simple pour l'exemple :

```
<template>
  <div class="about">
    <h1>This is an about page</h1>
  </div>
</template>
```

Je pense que vous avez compris le principe de base de ce routeur.

Si vous parcourez la documentation, ce que je vous conseille vivement, vous verrez qu'il est très riche en possibilités !

Maintenant qu'on a un routeur et qu'on a compris comment il fonctionne il faut décider quelle architecture on va mettre en place pour les quizs. je vous propose deux vues qui feront chacune appel à un composant (mais rappelez-vous que les vues sotn en fait des composants) :




On a deux états :

- **choix du quiz** : c'est la page d'accueil **Home** qui utilise le composant **Quizs** avec la route **'/'**
- **exécution du quiz** : c'est la vue **Action** qui utilise le composant **Quiz** avec la route **'/action'**

Donc pour le routeur on va avoir ce code :

```
import Vue from 'vue'
import Router from 'vue-router'
import Home from './views/Home.vue'
import Action from './views/Action.vue'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home
    },
    {
      path: '/action',
      name: 'action',
      component: Action
```

```
    }
  ]
})
```

Là la compilation ne va plus fonctionner parce que vous n’avez pas créé le composant **Action**. Dans un premier temps renommez **About** en **Action** pour pouvoir compiler.

On reviendra plus loin sur la route **action** pour la compléter.

Comme on ne va pas utiliser les liens pour le routage dans le composant **App** supprimez-les dans le template :

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>
```

# Les données

Comme on l’a fait dans le précédent article on va ajouter un fichier **db.json** à la racine du projet avec ce code pour avoir 2 quizs à disposition :

```
{
  "quizs" : [
    {
      "nom" : "Culture générale",
      "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
        }
      ]
    },
    {
      "nom" : "Technique",
      "questions" : [
        {
           "question": "Quelle est la science de base des autres
sciences ?",
          "answers": [
            "La biologie",
            "Les mathématiques",
            "La géographie",
            "La chimie"
          ],
          "ok": 1
        },
        {
           "question": "Quelle est l'unité de la capacité d'un
condensateur ?",
          "answers": [
            "Le faraday",
            "Le volt",
```

```
            "Le pascal",
            "L'hertz"
          ],
          "ok": 0
        },
        {
          "question": "Quelle est la formule de la loi d'Ohm ?",
          "answers": [
            "E = mc²",
            "P = UI",
            "U = RI",
            "P = RI²"
          ],
          "ok": 2
        }
      ]
    }
  ]
}
```

Vous pouvez démarrer le serveur avec :

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

Vérifier à l'adresse http://localhost:3000/quizs que vous recevez bien les données.

On doit maintenant configuer Vuex (donc le fichier **store.js**) pour récupérer ces données et les rendre disponibles pour l'application.

Comme il nous faut **Axios** installez-le aussi (avec l'interface graphique ou avec la console **npm i axios**).

Le code va être pratiquement identique à celui du dernier article :

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

Vue.use(Vuex)
```

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

# Le choix du quiz

On va commencer par régler le code du composant principal **App** :

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>

<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>
```

On place juste le résultat du routeur et du style pour toute l’application.

Le routeur nous envoie dans le composant **Home** pour l'url de base. Par défaut on a l'appel du composant **Helloworld** qu'on supprime. A la place on va appeler un composant **Quizs** qu'on a pas encore créé :

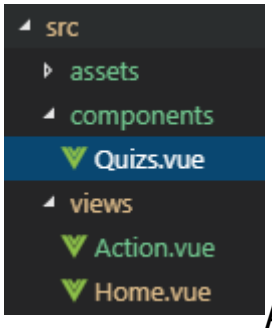
```
<template>
  <div class="home">
    <img alt="logo" class="mb-4" src="../assets/quiz.png">
    <quizs />
  </div>
</template>

<script>
import Quizs from '@/components/Quizs.vue'

export default {
  name: 'home',
  components: {
    Quizs
  }
}
</script>
```

J'ai aussi supprimé le titre et changé le nom du logo pour **quiz**, trouvez un joli logo pour l'application !

Donc on crée un fichier **Quizs.vue** :




Avec ce code :

```
<template>
  <div class="container">
    <b-card header="Choisissez un quiz" header-tag="header">
      <b-list-group>
        <b-list-group-item
          button
          v-for="(item, index) in quizs"
```

```
          :key="item.id"
          @click="action(index)">
          {{ item.nom }}
        </b-list-group-item>
      </b-list-group>
    </b-card>
  </div>
</template>

<script>
export default {
  name: 'quizs',
  methods: {
    action: function(index) {
      this.$router.push({ name: 'action', params: { id: index }})
    }
  },
  computed: {
    quizs () {
      return this.$store.state.quizs;
    }
  },
  created() {
    this.$store.dispatch('getData');
  }
}
</script>
```

Lorsque le composant est créé (**created**) on demande les données à Vuex (**store**) :

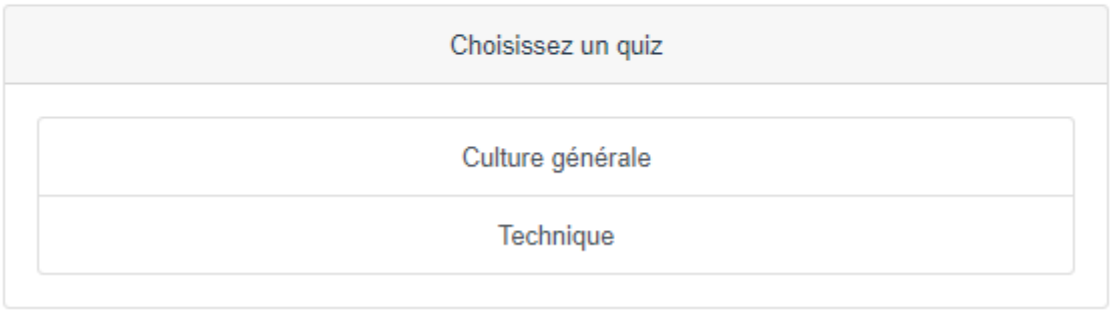
```
created() {
  this.$store.dispatch('getData');
}
```

Les données sont récupérées ici :

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

C’est ce qu’on a déjà vu dans le précédent article. Et on affiche

les noms des quizs :


Quand on clique sur une des deux possibilités on appelle la méthode **action** :

```
action: function(index) {
  this.$router.push({ name: 'action', params: { id: index }})
}
```

On accède au routeur avec **this.$router**. La méthode **push** permet d'appeler une route, ici **action**. Mais pour identifier le quiz on ajoute un paramètre **id** qui est l'index du quiz.

On va modifier le code du routeur (**router.js**) pour en tenir compte :

```
path: '/action/:id',
name: 'action',
component: Action
```

Le paramètre arrive après un double point.

Pour le moment on aboutit juste à cette page :


Parce qu'on a juste changé le com du composant, on va donc maintenant s'occuper de cette partie.

# Action !

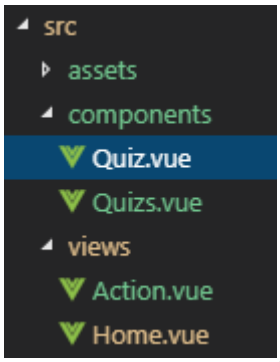
Pour le composant **Action** on va se contenter de ce code :

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

<script>
import Quiz from '@/components/Quiz.vue'

export default {
  name: 'action',
  components: {
    Quiz
  }
}
</script>
```

En fait on se contente d'appeler un composant **Quiz**. On crée ce composant :




Avec ce code :

```
<template>
  <div class="container">
    <h1 class="mb-4">{{ nom }}</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="fin" to="/" class="mt-4 ml-2">Choisir un
autre quiz !</b-button>
       <b-button v-if="voirReponse && !fin" @click="continuer"
class="mt-4">Continuer !</b-button>
    </b-card>
  </div>
</template>

<script>
export default {
  name: 'quiz',
  data: function () {
    return {
        id: 0,
        fin: false,
        index: 0,
        score: 0,
        variants: [...Array(4)],
        voirReponse: false,
    }
  },
  methods: {
    action: function(index) {
      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: {
    nom () {
        return this.$store.state.quizs[this.id].nom;
    },
    questions () {
        return this.$store.state.quizs[this.id].questions;
    }
  },
  created() {
    this.id = this.$route.params.id;
  }
}
</script>
```

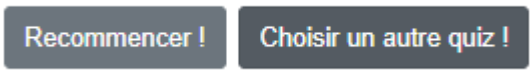
On récupère l'**id** dans la méthode **created** :

```
created() {
  this.id = this.$route.params.id;
}
```

On sait ainsi quel quiz est actif. Le reste du code n'apporte aucune nouveauté.

On peut afficher le nom du Quiz et les questions :


A la fin du quiz on a maintenant les deux possibilités :




Pour le bouton de choix d'un autre quiz voici le code :

```
<b-button v-if="fin" to="/" class="mt-4 ml-2">Choisir un autre quiz !</b-button>
```

La propriété **to** permet créer facilement un lien pour le routeur.

# Conclusion

On a vue dans cet article le principe du routeur de Vue. Vous pouvez consulter la documentation officielle pour plus de précision.