Tutoriel Vue.js

Introduction

Vue est un framework javascript permettant de développer des interfaces utilisateurs. Comme ses principaux concurrents React et Angular, Vue se veut être un framework visant à simplifier et organiser le développement d'applications web en implémentant un bon nombre de fonctionnalités telles que les modèles, les composants, les transitions, le routage, etc…

La particularité de Vue.js est qu'il a été créé pour permettre des développements rapides tout en possédant une simplicité d'utilisation et de compréhension de la logique. Ainsi migrer depuis un projet sans framework est beaucoup plus facile. Un composant au sens Vue.js est représenté par un fichier qui a comme extension « .vue », dans ce fichier on peut retrouver le modèle du composant (son template html, son style css mais aussi sa logique métier grâce à un script javascript).

Les vues

Tout comme ses concurrents React Native et Angular, on peut réutiliser un même composant en définissant lors de sa création les données qu'il contiendra. Un fichier vue est découpée en 3 parties correspondant au code html, js et css.

```
<template>
  <div>
    <div v-for="(run, index) in runs" :key="index" v-on:click="clickRun(index)">
      <Run :title=run.title :description=run.description :date=run.date :img=run.img />
    </div>
  </div>
</template>
```

```
<script>
import RunsJSON from "../resources/runs.json";
export default {
  name: "Runs",
  components: {
    Run
  },
  data: function() {
    return {
      runs: RunsJSON.runs,
    };
  },
  methods: {
    clickRun: function(index) {
      this.clickedItem = index;
      this.$emit("clicked", index);
    }
  }
};
</script>
```

```
<style scoped>
.navbar-brand {
  background: #006b00;
  width: 100%;
  color: white;
  padding-right: 0px;
  margin-right: 0px;
  cursor: pointer;
}
</style>
```

Figure 1 : Exemple type d'un fichier Vue.js

Le framework vue dispose de plusieurs fonctions et directives permettant de simplifier le développement des composants. Il existe par exemple la directive v-if=’isValid()’ qui est une structure conditionnelle d’affichage, l’élément sera affiché seulement si la condition est vérifiée. J’ai pu utiliser également v-for=’(run, index) in runs’ qui permet le rendu d’une liste d’éléments avec la notion d’élément courant et son index. Ainsi comme le montre la figure 1, il est très simple de définir le composant fils qui sera réutilisé dans le boucle du v-for.

Les évènements

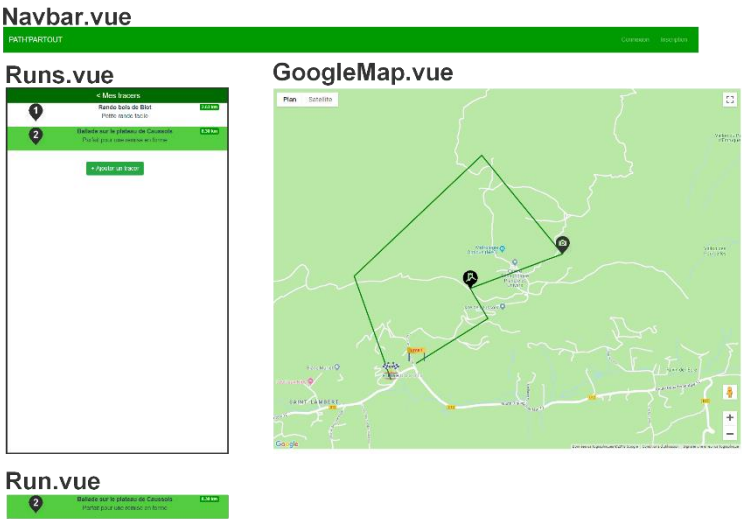
L’écriture d’évènement avec Vue.js est rendue très accessible et simple d’implémentation. En effet il est possible de définir sur n’importe quel élément une directive de type ’v-on :click=addMarker()’

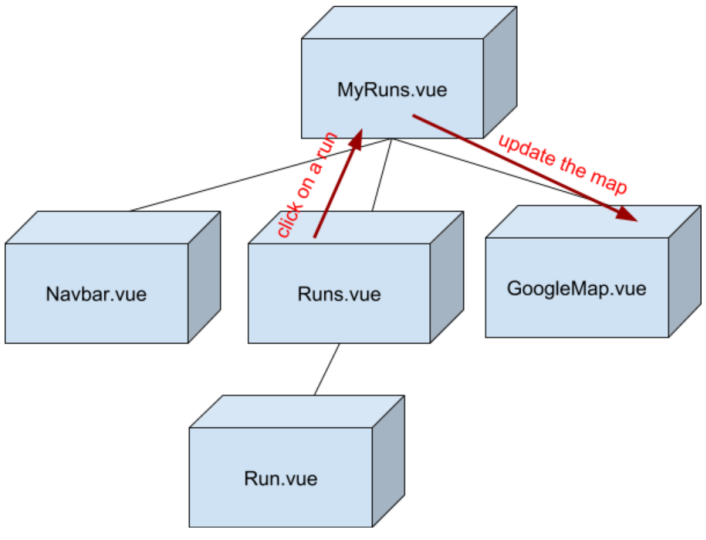
```
<div v-for="(run, index) in runs" :key="index" v-on:click="clickRun(index)">
  <Run :title=run.title :description=run.description :date=run.date :img=run.img />
</div>
```

```
methods: {
  clickRun: function(index) {
    this.clickedItem = index;
  }
}
```

Communication entre les composants

Ma principale difficulté dans la prise en main de ce framework a été de bien gérer la communication entre les composants. Par exemple dans l’application, sur la vue de mes tracers, dès lors que l’on clique sur un tracer en particulier, la carte est mise à jour pour afficher le tracer. Ainsi je dois faire remonter l’information au parent comme quoi un des tracers a été cliqué et ensuite le parent doit informer le composant GoogleMap.vue pour que la carte se mette à jour. La documentation officielle de Vue.js préconise d’utiliser les $emits et les évènements.





J'ai donc procédé comme suit :

Runs.vue

```
<div v-for="(run, index) in runs" :key="index" v-on:click="clickRun(index)">
  <Run :title=run.title :description=run.description :date=run.date />
</div>
```

```
clickRun: function(index) {
  this.clickedItem = index;
  this.$emit("clicked", index);
},
```

MyRuns.vue

```
<div class="col-md-4">
    <Runs @clicked="onClickChild"/>
</div>
<div class="col-md-8">
  <div>
    <GoogleMap ref="runIndex"/>
  </div>
</div>
```

```
onClickChild(value) {
  this.$refs.runIndex.setMarkers(value);
},
```

GoogleMap.vue

```
setMarkers(index) {
  this.path = this.runs[index].path;
  this.markers = this.runs[index].pois;
},
```

Adaptation au dispositif

Comme le but de notre projet était d'adapter l'affichage en fonction du dispositif, il était nécessaire d'avoir un moyen de détecter la taille du dispositif de l'utilisateur. Pour cela plusieurs possibilités s'offraient à moi. Je pouvais soit utilisé les Media Queries en CSS, qui va charger un certain style en fonction de la taille de l'écran. Ou alors je pouvais aussi choisir de n'afficher un élément que si l'utilisateur était sur un certain dispositif avec du code javascript. J'ai décidé de procéder ainsi. J'ai donc créer un fichier DeviceHelper.js qui contient la fonction isMobileDevice(). Donc cette fonction, on vient détecter les meta name contenus dans la variable navigator. Et en fonction de ce qu'elle contient on peut estimer si l'utilisateur est sur mobile ou non.

Ainsi, dans chaque vue où j'affiche un élément de façon conditionnelle, je charge le fichier helper et j'utilise la directive v-if pour afficher ou non un composant. Ce moyen de faire est assez lourd et pas forcément le plus efficace, puisque sur le navigateur par exemple, si on ouvre la console et que l'on décide de montrer l'application comment elle serait sous mobile, la fonction isMobileDevice() ne détectera pas que l'on est sous mobile puisque le navigateur reste un navigateur desktop et donc les affichages conditionnels ne fonctionneront pas correctement. Mais c'est un choix que j'ai pris pour utiliser le plus possible les directives offertes par Vue.js.

```
<div class="col-md-4 myCol">
    <Runs v-if="this.isAddingRun == false" @clicked="onClickChild" @reset="reset" @addRun="addRun"/>
    <AddRun v-if="this.isAddingRun == true" @back="back" @newMarker="newMarker" />
</div>
<div v-if="!helper.isMobileDevice()" class="col-md-8 myCol">
    <div>
        <GoogleMap ref="runIndex"/>
    </div>
</div>
```

Référencement des avantages et des inconvénients

Avantages

- Architecture modulable
- Facilité de migration d’un projet sans framework vers Vue.js
- Framework très flexible
- Contrairement à React, Vue.js est lié au DOM et utilise des attributs HTML spéciaux pour rendre le DOM réactif
- Poids du framework comparé à React et Angular

Inconvénients

- Framework récent et peu documenté
- Fichiers « .vue » qui peuvent être rapidement énormes puisqu’ils embarquent le code HTML, CSS et JS
- Manque de ressources
- Communication entre les composants assez lourde (voir la partie correspondante ci-dessus)