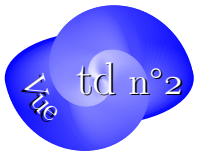

Vue JS

Débuts avec VueJS

Dans ce TD, on va se familiariser avec la librairie VueJS version 3 et son utilisation pour réaliser ensuite un service "Front" sous la forme d'une application monopage ou Single Page Application (SPA).

Des sites importants ont développé des SPA :

- gmail depuis 2004
- any.do
- feedly
- OneDrive

VueJS, comme beaucoup de librairies de composants déporte une partie de la logique métier sur le client et fonctionne donc bien avec un serveur REST. Les principes sont ceux de MVVM (ModelView View, Model) qui permettent d'avoir un binding entre les données représentées et celles du modèle sous-jacent de manière simple. Un système de templating est intégré. Vue a été développé par Evan You (cf `http://evanyou.me/`) lorsqu'il travaillait à Google.

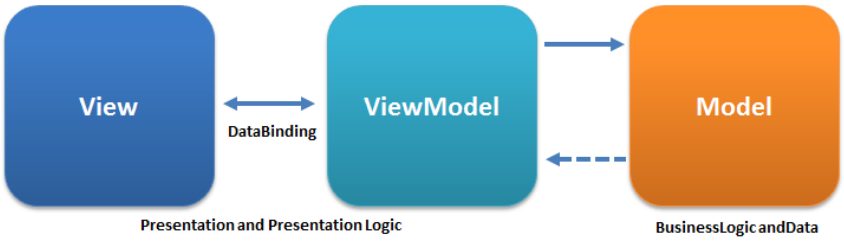

Figure 1 – Modèle-Vue Vue Modèle

Les principaux systèmes concurrents sont :

- React JS
- Svelte JS
- Angular JS

et beaucoup ont été également initiés par Google (ou Facebook pour React). Vue se caractérise par sa grande souplesse d'utilisation et est notamment plus léger qu'Angular qui est un Framework plus complet et Vue se révèle plus simple dans sa gestion des Etats que React. Polymer se contente de définir un modèle complet de composants Web sans être un

Framework. SvelteJS est un autre Framework qui se veut plus léger que VueJS et qui est basé sur le principe de compilation mais il n'est pas encore aussi mature que VueJS.

Vous trouverez la documentation sur Vue JS à l'adresse : `https://vuejs.org/guide/` et un tuto en français à l'adresse :`https://www.youtube.com/watch?v=TWE6NVVPNcc`

La manière moderne d'utiliser VueJS est soit de l'installer avec npm (Node Package Manager) et de l'importer dans un projet, soit comme nous allons le faire d'utiliser la version en ligne qui est disponible à l'adresse : `https://unpkg.com/vue@3/dist/vue.esm-browser.js`.

Cela se fait via l'utilisation de modules JS : `https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Modules`.

Exercice 1. Exemple de base

Prenons en main Vue en appelant le dépôt en ligne le plus récent.

```
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="utf-8">
      <script type="importmap">
        {
          "imports": {
            "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
          }
        }
      </script>
    </head>
  <body>
  <div id="app">{{ message }}</div>

  <script type="module">
    import { createApp } from 'vue'

    createApp({
      data() {
        return {
          message: 'Hello Vue!'
        }
      }
    }).mount('#app')
  </script>
  </body>
  </html>
```

La variable message a été remplacée par sa donnée définie dans le composant Vue. On dit aussi qu'elle a été interpolée. Le binding est ici unidirectionnel, c'est à dire en lecture seule.

Les modules ne fonctionnent pas nativement sur tous les navigateurs et il faut parfois utiliser un polyfill pour que cela fonctionne sur tous les navigateurs.

Exercice 2. Hello Vue

L'interpolation de variables n'est pas possible dans un attribut :

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script type="importmap">
      {
        "imports": {
          "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
        }
      }
    </script>
  </head>
  <body>
  <div id="app2">
  <span title="{{message}}">
    Survolez moi pour observer mon titre dynamiquement chargé !
  </span>
</div>

<script type="module">
  import { createApp } from 'vue'

  createApp({
    data() {
      return {
        message: 'Cette page a été chargée le ' + new Date()
      }
    }
  }).mount('#app2')
</script>
</body>
</html>
```

Essayez ce code. Que constatez-vous ?

Pour que cela fonctionne, Vue prévoit l'utilisation de l'attibut spécial v-bind :

```
<div id="app2">
<span v-bind:title="message">
Survolez moi pour observer mon titre dynamiquement chargé !
</span>
</div>

```

```
<script type="module">
import { createApp } from 'vue'

createApp({
data() {
  return {
    message: 'Cette page a été chargée le ' + new Date()
  }
}
}).mount('#app2')
</script>
```

Vous pouvez également remplacer **v-bind :title** par **:title**

Exercice 3. Avec une liste de données

Les données peuvent comporter une liste :

```

<div id="app4">
  <h2>{{ title }}</h2>
  <ul>
    <li>{{ items[0] }}</li>
    <li>{{ items[1] }}</li>
   </ul>

  <hr/>
  <em>Changez le titre de votre liste ici !</em>
  <input v-model="title" type="text"/>
</div>

<script type="module">
import { createApp } from 'vue'
createApp({
  data() {
    return {
      items: ['Courses', 'Apprendre REST'],
      title: 'Ma liste de taches'
    }
  }
}).mount('#app4')
</script>
```

Exercice 4. Une todolist avec Bootstrap

Réalisons maintenant une TodoList avec Boostrap. On utilise les CDN correspondants. Remarquez ici l'usage de la boucle for avec `v-for`. Il existe de même un `v-if` et `v-else` entre autres.

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <script type="importmap">
      {
        "imports": {
          "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
        }
      }
    </script>
    <link rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/
          bootstrap.min.css"
      integrity="
          sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65
          "
      crossorigin="anonymous">
</head>
<body>
<div id="app4" class="container">
    <h2>{{ title }}</h2>
    <ul>
      <li>{{ items[0] }}</li>
      <li>{{ items[1] }}</li>
     </ul>

    <hr/>
    <em>Changez le titre de votre liste ici !</em>
    <input v-model="title" type="text"/>
</div>

<script type="module">
  import { createApp } from 'vue'
  createApp({
    data() {
      return {
        items: ['Courses', 'Apprendre REST'],
        title: 'Ma liste de taches'
      }
    }
  }).mount('#app4')
</script>
// Tester en console: app4.todos.push({ text: 'New item' })
</script>
</body>
</html>
```

Exercice 5. Gestion d'un clic avec Vue

Mettons en place un bouton qui déclenche l'inversion d'une chaine de caractères. Ceci se fait via l'attribut `v-click` de Vue. (Toujours avec le même head contenant Vue et Bootstrap)

```
<div id="app5" class="container">
  <p>{{ message }}</p>
  <span class="input-group-btn">
  <button v-on:click="renverse"
    class="btn btn-default"
    type="button">Renverse</button>
  </span>
</div>

<script type="module">
  import { createApp } from 'vue'
  createApp({
    data() {
      return {
        message: 'Hello Vue.js!'
      }
    },
    methods: {
      renverse: function () {
        this.message = this.message.split('').reverse().join('')
      }
    }
  }).mount('#app5')
</script>
```

Exercice 6. Gestion des tâches effectuées

Reprenons notre TodoList et ajoutons au modèle un booléen pour indiquer si la tâche est achevée ou non. On va gérer l'affichage à l'aide d'une petite classe de style qui va barrer le texte des tâches effectuées.

```
<div id="app6" class="container">
<h2>{{ title }}</h2>
  <ol>
    <li v-for="todo in todos">
      <div class="checkbox">
        <label>
          <input type="checkbox">
          {{ todo.text }}
        </label>
        <div class="alert alert-success" v-if="todo.checked">
         Done
        </div>
      </div>
    </li>
  </ol>
</div>
```

```
17    <script type="module">
18      import { createApp } from 'vue'
19      createApp({
20        data() {
21          return {
22            todos: [
23          { text: 'Apprendre ES6', checked: true },
24          { text: 'Apprendre Vue', checked: false },
25          { text: 'Faire un projet magnifique', checked: false }
26            ],
27          title: 'Mes taches urgentes:'
28          }
29        }
30      }).mount('#app6')
31    </script>
```

Mais si on coche ou décoche les checkboxes, le lien avec le modèle n'est pas fait.

Exercice 7. Tâches effectuées avec double binding

Pour assurer le lien bi-directionnel entre les données de l'objet Vue et notre code html, changeons ce code comme suit grâce à la directive v-model

```
1     <div id="app6" class="container">
2     <h2>{{ title }}</h2>
3       <ol>
4         <li v-for="todo in todos"
5             v-bind:class="{ 'alert alert-success': todo.checked }">
6           <div class="checkbox">
7             <label>
8               <input type="checkbox"
9                      v-model="todo.checked">
10              {{ todo.text }}
11            </label>
12          </div>
13        </li>
14      </ol>
15    </div>
16
17    <script type="module">
18      import { createApp } from 'vue'
19      createApp({
20        data() {
21          return {
22            todos: [
23          { text: 'Apprendre ES6', checked: true },
24          { text: 'Apprendre Vue', checked: false },
25          { text: 'Faire un projet magnifique', checked: false }
26            ],
```

```
27          title: 'Mes taches urgentes:'
28          }
29        }
30      }).mount('#app6')
31    </script>
```

Exercice 8. Ajoutons des tâches

Développons maintenant un moyen pour ajouter des tâches ...

```
1   <div id="app6" class="container">
2   <h2>{{ title }}</h2>
3     <ol>
4       <li v-for="todo in todos"
5           v-bind:class="{ 'alert alert-success': todo.checked }">
6         <div class="checkbox">
7           <label>
8             <input type="checkbox"
9                    v-model="todo.checked">
10            {{ todo.text }}
11          </label>
12        </div>
13      </li>
14    </ol>
15  </div>
16
17  <script type="module">
18    import { createApp } from 'vue'
19    createApp({
20      data() {
21        return {
22          todos: [
23        { text: 'Apprendre ES6', checked: true },
24        { text: 'Apprendre Vue', checked: false },
25        { text: 'Faire un projet magnifique', checked: false }
26          ],
27        title: 'Mes taches urgentes:'
28        }
29      }
30    }).mount('#app6')
31  </script>
```

Exercice 9. Cycle de vie - montage et destruction

Vous pouvez consulter la documentation de Vue sur le cycle de vie d'un composant qui est assez riche à l'adresse `https://vuejs.org/guide/essentials/lifecycle.html`.

Le cycle de vie y est notamment décrit par le schema suivant dans lequel les hooks (points d'entrée ou d'accroche pour le programmeur) sont décrits en rouge horizontalement.

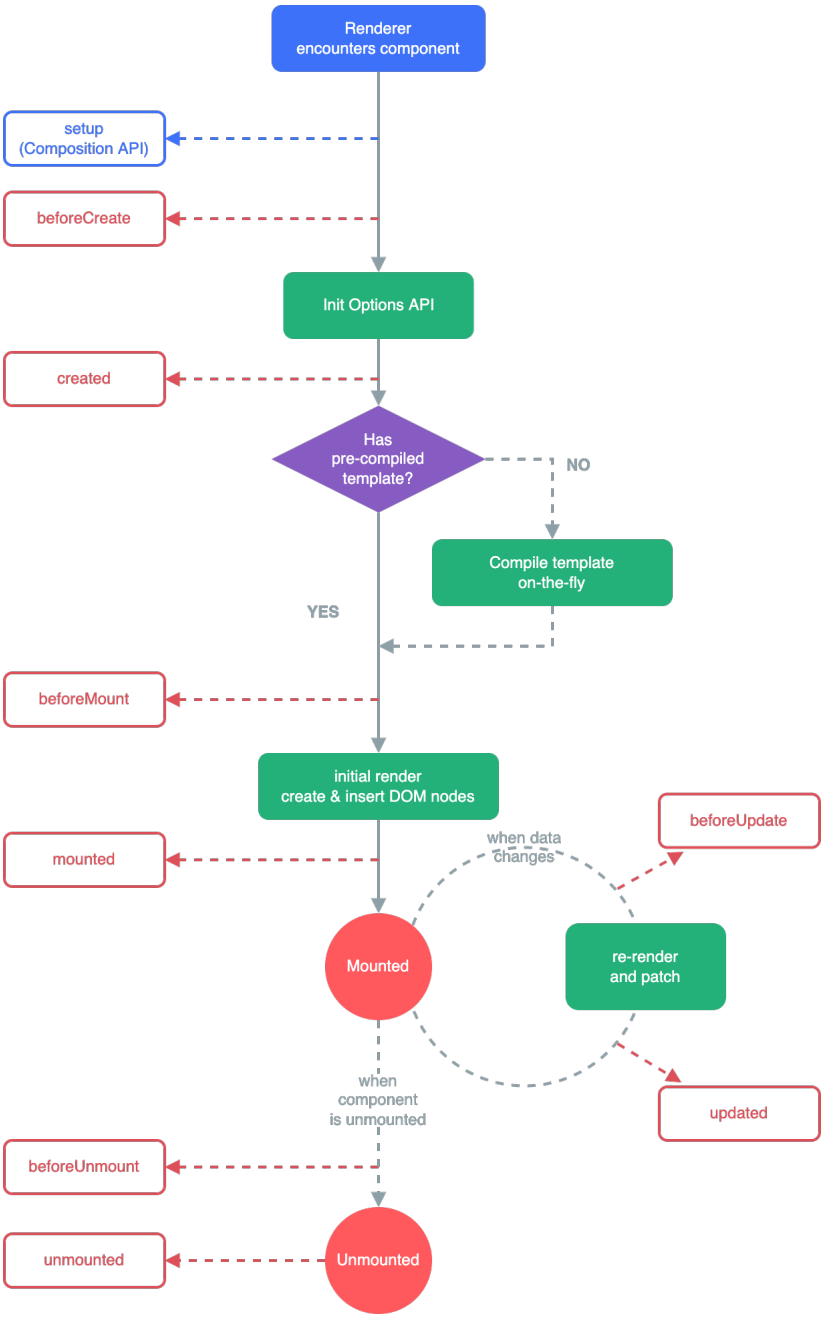

Figure 2 – Cycle de Vie Vue

Sur cette base, mettons en place un composant avec affichage d'un Timer :

```
  <div id="app8" class="container">
  <p>{{ seconds }}</p>
</div>

<script type="module">
  import { createApp } from 'vue'
  createApp({
    data() {
      return {
        seconds: 0
      }
    },
    mounted() {
      this.$interval = setInterval( () => {
        this.seconds++;
      }, 1000)
    },
    beforeUnmount(){
      clearInterval(this.$interval);
    }
  }).mount('#app8')
</script>
```

Remarquez l'usage de this.$interval pour notifier une variable propre au composant JS qui ne sera pas exportée

Exercice 10. Arrêtons le temps

Pour arrêter ce Timer, ajoutons un bouton qui appelle la méthode destroy() :

```
  <div id="app8" class="container">
    <p>{{ seconds }}</p>
    <span class="input-group-btn">
    <button v-on:click="stoppe"
      class="btn btn-default"
      type="button">Stop</button>
    </span>
  </div>

  <script type="module">
    import { createApp } from 'vue'
    createApp({
      data() {
        return {
          seconds: 0
        }
      },
      methods: {
        stoppe: function() {
```

```
20            clearInterval(this.$interval)
21              }
22        },
23        mounted() {
24          this.$interval = setInterval( () => {
25            this.seconds++;
26          }, 1000)
27        },
28        beforeUnmount(){
29          clearInterval(this.$interval);
30        }
31      }).mount('#app8');//montage et mise en place $interval
32    </script>
```

Exercice 11. Un composant basique

Mettons à présent en place un composant élémentaire doté de son propre tag.

```
1   <!doctype html>
2   <html>
3     <head>
4       <meta charset="utf-8">
5       <script type="importmap">
6       {
7         "imports": {
8           "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
9         }
10      }
11      </script>
12    </head>
13  <body>
14  <div id="app9">
15    <mon-composant></mon-composant>
16  </div>
17  <script type="module">
18    import { createApp } from 'vue'
19    createApp({
20      data() {
21        return {
22          message: 'Hello Vue!'
23        }},
24        components: {
25          'mon-composant': {
26            template: `<section> Hello Vue.js !</section>`
27          }
28        }
29      }
30    ).mount('#app9')
31  </script>
32  </body>
```

```
33    </html>
```

Exercice 12. Composant complété

Complétons à présent un peu ce composant pour rendre son texte et son style paramétrables :

```
1     <!doctype html>
2     <html>
3       <head>
4         <meta charset="utf-8">
5         <script type="importmap">
6           {
7             "imports": {
8               "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
9             }
10          }
11        </script>
12        <link rel="stylesheet"
13        href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/
              bootstrap.min.css"
14        integrity="
              sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65
              "
15        crossorigin="anonymous">
16      </head>
17    <body>
18    <div id="app">
19      <mon-composant :texte="texte"
20                     classe="alert alert-info">
21      </mon-composant>
22    </div>
23
24    <script type="module">
25      import { createApp } from 'vue'
26      createApp({
27        components: {
28          'mon-composant': {
29            props: ['classe', 'texte'],
30            template: '<section :class="classe"> {{ texte }}</section>',
31          }
32        },
33        data(){
34          return {
35            texte: 'Salut Vue.js',
36          }
37        }
38      }).mount('#app')
39    </script>
40    </body>
```

```
41    </html>
```

Exercice 13. TodoList en Vue

Reprendre la ToDoList pour la refaire en Vue avec 2 ou 3 composants bien choisis.

Exercice 14. Connexion à une API

Ajoutez un composant qui permet de se connecter à une API de votre choix (par exemple `https://jsonplaceholder.typicode.com/todos/`) et d'y récupérer les données des todos.