Résumé et projet de fin de cours de Vue.js

Cours de Vue.js 2

Résumé

1. Niveau : Débutant
2. Objectif : Renforcer les bases de Vue.js et approfondir certaines notions plus complexes
3. Temps imparti : 15 heures (3 x 5h)

Mise en œuvre

Contexte d'apprentissage

Les élèves avait déjà eu un cours de 10-15h sur le framework Vue.js ainsi que des cours sur le Js natif mais par manque d'expérience et d'entraînement ces bases étaient faibles. Selon les élèves, leur premier cours de Vue.js était sur l'utilisation d'un appelle à une API pour dynamiser un l'affichage d'une page Web. Le rendu déclaratif, les conditions et les boucles en Vue.js ont été abordés mais de façon trop succincte ce qui, selon mon observation, ne leur a pas permis d'avoir des bases assez solide en Vue.js.

Procédure

Le déroulement du cours se divise en 3 parties pour chaque notions :

- Présentation et explication du fonctionnement de la notion en live coding au tableau.
- Entrainement sur cette notion avec des exercices proche de l'exemple présenté au tableau, auquel ils peuvent se référer. Encadrement et explication approfondis pour les élèves en difficulté. Exercices en plus pour les élèves les plus efficaces qui aurai terminé en avance.
- Correction des exercices en live coding au tableau

Les exercices données sur la notions suivantes nécessite l'utilisation des notions précédentes pour construire petit à petit une base de connaissances plus solide. Les élèves recopient les exemples au tableau, ainsi que les corrections, pour garder un code de référence qui fonctionne et qui fera office de notes pour les prochains cours et qui sert aussi pour des révisions.

Darius Kohen - cours du 24/02/2020 au 27/02/2020 pour l'IIM

Contenu des cours

Notions abordés au premier cours

- Le rendu déclaratif, interpolation

```
{{ message }}
```

- L'instance de Vue

```
let vm = new Vue ({
        el: "#app",
        data: {
                message: "bonjour"
        }
})
```

- Les filtres

```
{{ message | capitalize }}
```

- Les définitions de filtres en global et local

```
Vue.filter('capitalize', (value) => {
        return value.charAt(0).toUpperCase() + value.slice(1)
})
```

- Les directives de conditions

```
<p v-if="success">{{ message | capitalize }}</p>
<p v-else>{{ errormsg }}</p>
<p v-show="debug">{{ debugmsg }}</p>
```

- Les liaisons d'attributs, de class et de style

```
<a v-bind:href="link">{{ linkName }}</a>
<a :href="link" v-bind:style="colorLink">{{ linkName }}</a>
<i class="fa" :class="success ? 'fa-angle-down' : 'fa-angle-up'"
        :style="success ? 'color:green' : 'color:red'"></i>
```

- La gestions des entrées utilisateur, directive d'événement

```
<a :href="link" v-on:click="clickLink">{{ linkName }}</a>
<button @click="launchTime">click</a>
```

- Les modificateurs de directives

```
<a :href=”link” @click.prevent>click</a>
<input type=”text” v-model.lazy=”message”>
```

- Les méthodes

```
let vm = new Vue ({
        el: “#app”,
        data: {
                seconds: 0
        },
        methods:
        clickLink: function (e) {
                e.preventDefault()
        }
})
```

- Le cycle de vie d’une instance Vue

```
let vm = new Vue ({
        el: “#app”,
        data: {
                seconds: 0
        },
        mounted () {
                this.$interval = setInterval(() => {
                        this.seconds++
                }, 1000)
        },
        destroyed () {
                clearInterval(this.$interval)
        }
})
```

Notions abordés au deuxième cours

- La directive de boucles

```
<div v-for="(el, index) in table">
{{ index}} {{ el.quantity }}
</div>
```

- Les propriétés calculées

```
let vm = new Vue ({
        el: “#app”,
        data: {
                products : [1, 0, 5, 2]
        },
        computed: {
                totalProducts: function () {
                        this.$total = 0
                        this.products.forEach((el, i) => {
                                this.$total += el
                        });
                        return this.$total
                }
        }
})
```

- Les composants
  - Html :
```
<rowproduct
        :quantity=”product.quantity”
        :name=”product.name”
        @add=”product.quantity+=1”
        @sub=”product.quantity-=1”
    >
        <div slot="addbutton">Add</div>
        <div slot="subbutton">Sub</div>
    </rowproduct>
```
  - Js :
```
Vue.component(‘rowproduct’, {
        props: {
            quantity: {type: Number, default:0},
            name: {type: String, default:’’}
        },
        template: `<div>
            <input type="number" v-model.number="quantity">
            {{ name | capitalize }}
            <button @click="plus">
                <slot name="addbutton"></slot>
            </button>
            <button @click="minus">
                <slot name="subbutton"></slot>
            </button>
        </div>`,
        methods: {
            plus () {
                this.$emit(‘add’)
            },
            minus () {
                this.$emit(‘sub’)
            }
        }
    })
```

Notions abordés au troisième cours

- Les transitions CSS

```
<transition name="fade" appear>
    <p v-if=”show”>Coucou</p>
</transition>
```

Projet de fin de cours, la todo list

La todo list est un exercice courant pour s'entraîner sur un framework Js, et reste utile en production sur certains site web de part son côté générique.
Pour ce projet il y aura les fichiers todo.html et todo.css de fournis pour que les élèves puisse ne pas perdre de temps avec l'affichage. L'élève rendra les fichiers todo.html et todo.js. En partant de l'exemple fournis todo.html, qui contient toute les balises html nécessaire à la réalisation de la todo list, il devra la dynamiser avec ses connaissances en Vue.hs et créer le fichier todo.js.

Évaluation

La notation des l'élèves se fera sur le travail fournis pendant les exercices sur les notions abordées pendant le cours, la participation pendant les cours et sur le rendu du projet de fin de cours. Les fonctionnalités suivantes sont attendu sur le projet todo list pour considéré le projet terminé à 100% :

- Appuyer sur enter après avoir remplis l'input principale ajoute une tâche.
- Appuyer sur la checkbox à gauche dans la tâche met la tâche en gris et barré (accomplie)
- Appuyer sur la croix rouge à droite dans la tâche la supprime
- Le nombre de "tâche à faire" est dynamique, une tâche accomplie n'est pas compté
- Les filtres (Toute, à faire et faite) sont dynamiques et affiche respectivement toute les tâches, les tâches non accomplies et les tâches accomplies
- Le bouton "Clear accomplished" supprime toute les tâches accomplies
- Le bouton "Clear accomplished" n'apparaît que si il y a des tâches accomplies
- Le footer n'apparaît pas si il n'y a aucune tâche
- Cocher la flèche à gauche dans l'input principal met toute les tâches en accomplies
- Si toute les tâches sont accomplies, la flèche à gauche dans l'input principal est coché
- Décocher la flèche à gauche dans l'input principal, met toute les tâches en non accomplies
- Faire un double clic sur une tâche permet d'accéder à un input pour changer le texte de la tâche
- Appuyer sur enter après avoir modifié la tâche fait disparaître l'input
- Appuyer sur échap après avoir modifié la tâche lui rend sa valeur avec la modification

Des points bonus peuvent être donné si des fonctionnalités pertinentes sont ajouté.