VUE.JS

Développer avec VueJS 3

LUCCIONI Jean-Baptiste

m2i Formation

OBJECTIFS

Objectifs de formation

A l'issue de cette formation, vous serez capable de :

- Mettre en oeuvre le framework Vue.js 3
- Utiliser Vue.js 3 dans le cadre d'une application SPA (Single Page Application) et d'applications clientes plus conventionnelles.

PREREQUIS

Avoir une très bonne connaissance de HTML 5, de CSS 3, et de JavaScript.

PUBLIC CONCERNE

Développeurs de sites Web désirant développer des applications actuelles Front End.

m2i Formation

PROGRAMME

1 – Introduction à Vue.js 3

- Présentation du framework
- Avantages et caractéristiques principales
- Comparaison avec d'autres frameworks

2 – Installation et Configuration

- Téléchargement et Installation
- Configuration de Base d'un Projet
- Utilisation de l'Interface en Ligne de Commande

3 – Composants

- Introduction aux Composants
- Création de Composants
- Propriétés (Props)
- Événements Personnalisés
- Slots
- Directives Avancées

4 – Routing avec Vue Router

- Présentation de Vue Router
- Configuration des routes
- Navigation entre les vues
- Passage de paramètres et utilisation des routes dynamiques

5 – Gestion d'état avec VueX

- Introduction à Vuex
- Structure de Base de Vuex
- Utilisation de Vuex dans les Composants

Introduction à Vue.js 3

Origines de Vue.js 3

Vue.js a été conçu par Evan You en 2014, initialement en tant que projet visant à simplifier le développement d'interfaces utilisateur interactives.
L'inspiration provient de la volonté de combiner les avantages de React et Angular tout en restant accessible aux développeurs de tous niveaux.

Popularité et Utilisation

La popularité de Vue.js 3 a explosé en raison de sa syntaxe claire, de sa documentation exhaustive et de sa courbe d'apprentissage progressive.
Il est utilisé par de grandes entreprises comme Alibaba, Xiaomi, Gitlab et d'autres, ainsi que dans des projets open source.

Caractéristiques Principales

Reactivité et liaison bidirectionnelle

- Vue.js 3 introduit un système de réactivité basé sur des dépendances pour un suivi précis des changements de données.
- La liaison bidirectionnelle permet une synchronisation fluide entre les composants et leur état interne.

Modularité des Composants

- Les composants Vue sont des unités autonomes, favorisant la modularité et la réutilisation.
- La composition de composants permet une structuration claire et une maintenance facilitée.

Avantages de Vue.js 3 par rapport aux autres Frameworks

Comparaison avec Angular et React

Vue.js 3 offre une approche plus légère et flexible par rapport à Angular, réduisant la complexité tout en conservant une architecture réactive.
Par rapport à React, Vue.js 3 propose une syntaxe plus concise et une intégration native des fonctionnalités clés telles que la gestion d'état.

Élégance et Simplicité

Vue.js 3 se distingue par son élégance syntaxique et sa simplicité de mise en œuvre.
Les développeurs apprécient sa facilité d'adoption et sa courbe d'apprentissage douce, ce qui en fait un choix intuitif.

Adaptabilité aux Besoins du Projet

Vue.js 3 peut être utilisé de manière modulaire, permettant aux développeurs de l'adopter partiellement dans un projet existant sans compromettre la stabilité.
Cette adaptabilité offre une flexibilité pour intégrer Vue.js 3 là où il apporte le plus de valeur, sans nécessiter une refonte majeure.

Environnement de Développement et Installation

Outils de Développement Vue.js

- Vue DevTools offre des fonctionnalités avancées telles que l'inspection des composants, la modification en temps réel des données, et le suivi des événements.
- Des extensions pour les navigateurs populaires simplifient le débogage.

Installation de Vue.js 3

- Les développeurs peuvent choisir entre CDN, npm, ou utiliser Vue CLI pour automatiser le processus.
- La création d'un nouveau projet avec Vue CLI simplifie la configuration initiale.

Vue CLI : Interface en Ligne de Commande

- Simplification du Développement : Vue CLI (Interface en Ligne de Commande) est un outil essentiel pour simplifier le processus de développement.
- Création de Projets : Vue CLI permet de créer facilement de nouveaux projets Vue.js avec une structure de fichiers bien définie.
- Gestion de Projets : Il facilite la gestion du projet en fournissant des commandes pour la création de composants, la gestion des dépendances, et d'autres tâches courantes.
- Automatisation des Tâches : Vue CLI automatise de nombreuses tâches de configuration, permettant aux développeurs de se concentrer sur le développement plutôt que sur la configuration.
- Déploiement Simplifié : La CLI facilite également le déploiement des applications Vue.js, simplifiant ainsi la mise en production.
- Extensions et Plugins : Vue CLI supporte des extensions et plugins, permettant d'ajouter des fonctionnalités spécifiques au projet, adaptant ainsi l'outil aux besoins particuliers de chaque équipe de développement.

Téléchargement et Installation

Méthodes d'Installation

CDN (Content Delivery Network) : Une méthode rapide et simple pour intégrer Vue.js dans des projets simples ou des prototypes, bien que la dépendance à un CDN puisse présenter des risques de disponibilité.
npm (Node Package Manager) : Offre un contrôle précis des versions et des dépendances, idéal pour des projets complexes, bien que son utilisation puisse être plus complexe et entraîner une taille de dossier plus importante.
Vue CLI (Command Line Interface) : Simplifie le processus avec des commandes automatisées, créant une structure de projet cohérente, bien que son utilisation initiale puisse demander une certaine familiarisation avec la CLI et nécessite Node.js.

Mise en place par VueCLI

```
npm install -g @vue/cli
```

```
vue create hello-world
```

*L'installation d'un projet par NPM ou VueCLI nécéssite d'avoir installer **NodeJs** sur sa machine, dans l'objectif de cette formation, nous utiliserons la version LTS de node dans sa version **20.X***

Création d'un Nouveau Projet

Pour initier un projet Vue.js, la commande essentielle est **vue create**.

Elle permet de créer un nouveau projet interactivement en posant des questions cruciales aux développeurs, comme le choix du gestionnaire de paquets (npm ou yarn), la configuration manuelle ou préconfigurée, et l'inclusion de fonctionnalités telles que *VueX* et *Vue Router*.

Exemple :

```
vue create mon-projet
```

Vue CLI génère une structure de projet cohérente avec un dossier src pour le code source, public pour les ressources statiques, et des fichiers tels que *App.vue* et *main.js* qui définissent l'entrée principale du projet.

Comprendre cette structure facilite la navigation et la localisation des éléments du projet.

Les Composants et leur Rôle

Un composant Vue.js est une unité réutilisable et autonome qui encapsule une partie spécifique de l'interface utilisateur. Il est défini avec une structure comprenant un template (pour la vue HTML), des données (pour le modèle), des méthodes (pour la logique), et d'autres options. Les composants favorisent la modularité et facilitent la gestion de l'application.

```vue
<!-- Définition d'un composant -->
<template>
    <div>
        <h1>{{ titre }}</h1>
        <p>{{ contenu }}</p>
    </div>
</template>

<script>
2 usages
export default {
    data() {
        return {
            titre: 'Mon Premier Composant',
            contenu: 'Contenu du composant ici.'
        };
    }
};
</script>
```

Avantages de la Création de Composants Réutilisables :

Les composants permettent de diviser l'interface utilisateur en petites pièces gérables, favorisant la réutilisation du code. Cela simplifie la maintenance, améliore la lisibilité, et offre une meilleure gestion de l'état. Les composants peuvent être utilisés à plusieurs endroits de l'application, garantissant une cohérence visuelle.

Création de Composants

- La création d'un composant Vue.js se fait en définissant un nouvel objet Vue, souvent appelé composant, à l'aide de la méthode *Vue.component*. La syntaxe de base implique de déclarer un template, des propriétés de données, des méthodes, et d'autres options essentielles.

- Chaque composant Vue.js suit une structure définie. Le fichier peut inclure une section <template> pour le rendu HTML, <script> pour la logique JavaScript (composant Vue), et <style> pour les styles CSS spécifiques au composant.

```vue
<template>
    <div>
        <h2>{{ titre }}</h2>
        <p>{{ contenu }}</p>
    </div>
</template>

<script>
2 usages
export default {
    data() {
        return {
            titre: 'Mon Composant',
            contenu: 'Contenu du composant ici.'
        };
    }
};
</script>

<style scoped>
/* Styles spécifiques au composant */
</style>
```

Propriétés (Props)

Les propriétés pour transmettre des Données aux composants Enfants

Les props sont des propriétés personnalisées que vous pouvez passer à un composant enfant. Ils permettent la communication entre composants en transmettant des données du composant parent au composant enfant. Les props sont déclarées dans la balise du composant parent et sont accessibles dans le composant enfant.

```
<!-- Composant Parent -->
<template>
  <mon-composant :message="donneeDuParent"></mon-composant>
</template>

<script>
export default {
  data() {
    return {
      donneeDuParent: 'Donnée du parent'
    };
  }
};
</script>
```

```
<!-- Composant Enfant -->
<template>
  <div>
    <p>{{ message }}</p>
  </div>
</template>

<script>
export default {
  props: ['message']
};
</script>
```

Validation des Props

Garantir la Cohérence des Données

La validation des props permet de spécifier le type attendu pour une prop donnée. Cela garantit la cohérence des données et aide à détecter rapidement les erreurs de développement.

```
export default {
    data() {
        return {
            titre: 'Mon Composant',
            contenu: 'Contenu du composant ici.'
        };
    },
    props: {
        message: String, // La prop message doit être une chaîne de caractères
        nombre: Number,  // La prop nombre doit être un nombre
        estActif: Boolean // La prop estActif doit être un booléen
    }
};
</script>
```

Événements Personnalisés

Émission d'événements depuis les composants enfants

Les composants enfants peuvent émettre des événements personnalisés pour notifier le composant parent d'une action ou d'un changement. Cela permet une communication ascendante entre composants.

```
<!-- Composant Enfant -->
<template>
    <button @click="emettreEvenement">Cliquez-moi</button>
</template>

<script>
2 usages
export default {
    methods: {
        emettreEvenement() {
            this.$emit( event: 'evenement-personnalise', /* données optionnelles */);
        }
    }
};
</script>
```

Communication bidirectionnelle entre Composants

La communication bidirectionnelle peut être établie en utilisant les props et les événements. Un composant parent peut passer une *prop* à un composant enfant, et l'enfant peut émettre un événement pour notifier le parent d'un changement. Cela permet une mise à jour dynamique des données entre les composants.

```vue
<!-- Composant Parent -->
<template>
    <mon-composant :valeur="donneeDuParent"
                   @changement="mettreAJourDonnee"></mon-composant>
</template>

<script>
2 usages
export default {
    data() {
        return {
            donneeDuParent: 'Donnée du parent'
        };
    },
    methods: {
        mettreAJourDonnee(nouvelleValeur) {
            this.donneeDuParent = nouvelleValeur;
        }
    }
};
</script>
```

```vue
<!-- Composant Enfant -->
<template>
    <input :value="valeur" @input="emettreChangement">
</template>

<script>
2 usages
export default {
    props: ['valeur'],
    methods: {
        emettreChangement(event) {
            this.$emit( event: 'changement', event.target.value);
        }
    }
};
</script>
```

Cycle de Vie des Composants

Phases du Cycle de Vie

Chaque composant Vue.js traverse différentes phases tout au long de son existence. Les phases clés comprennent :

- created : Le composant est créé, mais le DOM n'est pas encore accessible. C'est un moment idéal pour initialiser des données ou mettre en place des observateurs.
- mounted : Le composant est inséré dans le DOM. À ce stade, le composant est rendu et visible. C'est le moment propice pour effectuer des opérations nécessitant l'accès au DOM.
- updated : Le composant est mis à jour en réponse à des changements. Cela peut se produire suite à une modification de données. C'est un bon endroit pour effectuer des opérations après une mise à jour.
- destroyed : Le composant est détruit et nettoyé. Il s'agit de la phase finale, où les ressources peuvent être libérées.

Hooks de cycle de Vie

Les hooks de cycle de vie sont des fonctions spéciales que vous pouvez définir dans votre composant pour effectuer des actions à des moments précis du cycle de vie. Voici quelques hooks couramment utilisés en plus des précédents étudiés :

- beforeCreate : Exécuté avant la création du composant. À ce stade, les options du composant ne sont pas encore fusionnées.
- beforeMount : Exécuté avant que le composant ne soit inséré dans le DOM.
- beforeUpdate : Exécuté avant que le composant ne soit mis à jour, souvent en réponse à des modifications de données.
- beforeDestroy : Exécuté avant que le composant ne soit détruit. C'est l'occasion de nettoyer les ressources, les abonnements, etc.

Utilisation des Slots

Les slots sont des points d'insertion dans le template d'un composant où le contenu peut être injecté à partir du composant parent.
Ils offrent une flexibilité accrue en permettant au parent de définir le contenu du composant enfant.

Slots Nommés et Slots par Défaut

Les slots peuvent être nommés pour permettre au parent d'insérer du contenu à des emplacements spécifiques. De plus, un slot par défaut est utilisé lorsque le contenu n'est pas spécifié pour un slot nommé.

```
<!-- Composant Enfant avec Slot -->
<template>
    <div>
        <h2>Titre du Composant</h2>
        <slot></slot>
    </div>
</template>

<!-- Utilisation du Composant Enfant avec Slot -->
<template>
    <mon-composant>
        <p>Contenu du composant défini par le parent.</p>
    </mon-composant>
</template>
```

Utilisation des Slots

Exemple de slot nommé et de slot par défault :

```vue
<!-- Composant avec Slots Nommés et Slot par Défaut -->
<template>
  <div>
    <header>
      <slot name="entete"></slot>
    </header>
    <main>
      <slot></slot> <!-- Slot par défaut -->
    </main>
    <footer>
      <slot name="pied-de-page"></slot>
    </footer>
  </div>
</template>
```

```vue
<!-- Utilisation du Composant avec Slots Nommés -->
<template>
  <mon-composant>
    <template v-slot:entete>
      <h1>Titre de la Page</h1>
    </template>
    <p>Contenu principal de la page.</p>
    <template v-slot:pied-de-page>
      <p>Pied de page personnalisé.</p>
    </template>
  </mon-composant>
</template>
```

Directives Avancées

Directives v-if, v-for, v-bind, et v-on Avancées :

Les directives sont des instructions spéciales dans le template Vue.js qui appliquent des comportements réactifs aux éléments du DOM. Les directives avancées incluent :

- v-if : Contrôle conditionnel de l'affichage d'un élément.
- v-for : Itération à travers une liste pour le rendu répété d'éléments.
- v-bind : Liaison de propriétés d'attribut ou de style.
- v-on : Gestion des événements.
- v-model : Permet de faire du « Two-way data binding » en lui associant une donnée

```
<!-- Directive v-if pour le contrôle conditionnel -->
<p v-if="afficherMessage">Afficher ce message</p>

<!-- Directive v-for pour l'itération -->
<ul>
    <li v-for="item in listeItems">{{ item }}</li>
</ul>

<!-- Directive v-bind pour la liaison de propriétés -->
<a v-bind:href="lienDynamique">Lien Dynamique</a>

<!-- Directive v-on pour la gestion des événements -->
<button v-on:click="executerAction">Cliquer pour exécuter</button>
```

Directives Avancées

Utilisation de Directives Personnalisées :

Vue.js permet également la création de directives personnalisées pour étendre le comportement du framework. Cela offre un moyen puissant d'ajouter des fonctionnalités spécifiques à l'application.

```
// Définition d'une directive personnalisée
app.directive( name: 'ma-directive', directive: {
    created(el, binding : DirectiveBinding<any> ) : void  {
        // Logique à exécuter lors de la liaison de la directive
        console.log('Directive liée!', binding.value);
    },
    // Autres hooks de cycle de vie de la directive
});
```

```
<div v-ma-directive="70"></div>
```

Introduction à Vue Router

Qu'est-ce que Vue Router ?

- Vue Router est le routeur officiel de *Vue.js*, spécialement conçu pour la construction d'applications à page unique (SPA).
- Il permet la navigation dans différentes vues (pages) de manière fluide sans rechargement de la page.

Pourquoi utiliser Vue Router ?

- Facilite la gestion de la navigation dans les applications Vue.js.
- Permet de définir des routes pour différentes parties de l'application.
- Facilite la gestion de l'historique de navigation.

Installation de Vue Router :

- Utiliser npm pour installer Vue Router dans un projet Vue.js.
- Configuration de base dans le fichier principal de l'application (généralement main.js).

```
npm install vue-router
```

Configuration des Routes

Vue Router est le routeur officiel de *Vue.js*, spécialement conçu pour la construction d'applications à page unique (SPA).
Il permet la navigation dans différentes vues (pages) de manière fluide sans rechargement de la page.

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import { createRouter, createWebHistory } from 'vue-router';

const router = createRouter({
  history: createWebHistory(),
  routes: [
    // Définir les routes ici
  ],
});

const app = createApp(App);
app.use(router);
app.mount('#app');
```

```
// Exemple de configuration des routes
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
  { path: '/contact', component: Contact },
];
```

Navigation entre les Vues

```
<!-- Dans le template -->
<router-link to="/">Accueil</router-link>
<router-link to="/about">À propos</router-link>
<router-link to="/contact">Contact</router-link>

<router-view></router-view>
```

- Utilisation de <router-link> :

<router-link> est une balise spéciale fournie par Vue Router pour créer des liens de navigation.
Elle accepte une propriété to indiquant la route vers laquelle naviguer.

- Utilisation de <router-view> :

<router-view> est une balise utilisée pour afficher le composant correspondant à la route active.
Elle est généralement placée à l'endroit où vous souhaitez que le composant de la route soit rendu.

Navigation Programmée

Utilisation de this.$router.push :
La méthode push est utilisée pour naviguer vers une nouvelle route.
Elle prend en argument le chemin vers la nouvelle route.

```
// Programmation de la navigation
methods: {
  allerVersAbout() {
    this.$router.push('/about');
  },
}
```

Utilisation de this.$router.replace :
La méthode replace est similaire à push, mais elle remplace l'entrée d'historique actuelle plutôt que d'ajouter une nouvelle entrée.

```
// Remplacement de la route actuelle
this.$router.replace('/nouvelle-route');
```

Navigation avec paramètres

- Définition d'une Route avec Paramètre :

Les paramètres sont définis dans le chemin de la route précédés de deux-points (:)

```
{ path: '/utilisateur/:id', component: UserProfile }
```

- Récupération du Paramètre dans le Composant :

Les paramètres sont accessibles via *this.$route.params*.

```
// Récupération du paramètre dans le composant
this.$route.params.id
```

Exemple de Navigation avec Paramètres :

```
<!-- Dans le template -->
<router-link :to="{ path: '/utilisateur/1' }">Voir Utilisateur 1</router-link>
```

Introduction à Vuex

Qu'est-ce que VueX ?

Vuex est un gestionnaire d'état d'application pour les applications Vue.js.
Il offre un moyen centralisé de gérer l'état partagé entre les composants de l'application.

Pourquoi utiliser VueX ?

Facilite la gestion de l'état global de l'application.
Élimine la nécessité de transmettre des données entre de nombreux composants.
Permet une gestion plus transparente des mutations d'état.

Installation de Vuex :

Utiliser npm pour installer Vuex dans un projet Vue.js.
Configuration de base dans le fichier principal de l'application (généralement main.js).

```
npm install vuex
```

Configuration de Vuex

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import { createStore } from 'vuex';

const store = createStore({
  // Configuration du store Vuex
  state() {
    return {
      // État global de l'application
    };
  },
  mutations: {
    // Mutations pour modifier l'état
  },
  actions: {
    // Actions pour effectuer des opérations asynchrones
  },
  getters: {
    // Getters pour obtenir des données calculées
  },
});

const app = createApp(App);
app.use(store);
app.mount('#app');
```

Structure de Base de Vuex

State :
Représente l'état global de l'application.
Accessible dans tous les composants de l'application.

Mutations :
Fonctions responsables de la modification de l'état.
Mutations sont synchrones et doivent être utilisées pour des modifications directes de l'état.

Actions :
Fonctions responsables de la gestion des opérations asynchrones ou complexes.
Les actions appellent des mutations pour modifier l'état.

Getters :
Fonctions utilisées pour obtenir des données calculées à partir de l'état.
Utile pour la logique de transformation des données.

Utilisation de Vuex dans les Composants

Accéder à l'État :

- Représente l'état global de l'application.
- Accessible dans tous les composants de l'application.

```
// Dans un composant
import { mapState } from 'vuex';

export default {
  // ...
  computed: {
    // Utilisation de mapState pour accéder à 'utilisateur'
    ...mapState(['utilisateur']),
  },
};
```

Mutation, actions et accesseur

Appel de Mutations

Utilisation de commit pour appeler une mutation.

```
// Dans une méthode d'un composant
this.$store.commit('modifierUtilisateur', nouveauUtilisateur);
```

Dispatch d'Actions

Utilisation de dispatch pour appeler une action.

```
// Dans une méthode d'un composant
this.$store.dispatch('chargerDonneesUtilisateur');
```

Utilisation de Getters

Utilisation de mapGetters pour accéder à des getters spécifiques.

```
export default {
  // ...
  computed: {
    // Utilisation de mapGetters pour accéder à 'utilisateurActif'
    ...mapGetters(['utilisateurActif']),
  },
};
```

Exemple d'utilisation du store

```
// Exemple de configuration du store
const store = createStore({
  state() {
    return {
      utilisateurs: [],
      utilisateurActif: null,
    };
  },
  mutations: {
    // Mutations pour modifier l'état
    modifierUtilisateurs(state, nouveauxUtilisateurs) {
      state.utilisateurs = nouveauxUtilisateurs;
    },
    selectionnerUtilisateur(state, utilisateur) {
      state.utilisateurActif = utilisateur;
    },
  },
```

Exemple d'utilisation du store

```
  actions: {
    // Actions pour effectuer des opérations asynchrones
    async chargerUtilisateurs({ commit }) {
      const response = await api.get('/utilisateurs');
      commit('modifierUtilisateurs', response.data);
    },
    // ...
  },
  getters: {
    // Getters pour obtenir des données calculées
    utilisateurActif: (state) => state.utilisateurActif,
    // ...
  },
});
```

Exemple d'utilisation du store

```vue
// Dans un composant
import { mapState, mapGetters, mapActions } from 'vuex';

export default {
  // ...
  computed: {
    ...mapState(['utilisateurs']),
    ...mapGetters(['utilisateurActif']),
  },
  methods: {
    ...mapActions(['chargerUtilisateurs']),
    selectionnerUtilisateur(utilisateur) {
      this.$store.commit('selectionnerUtilisateur', utilisateur);
    },
  },
};
```