VuesJs

1. Introduction

Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement

Vue.js est un framework JavaScript open-source utilisé pour construire des interfaces utilisateur réactives et dynamiques. Il est souvent utilisé pour créer des applications web à page unique (SPA - Single Page Applications).

Une SPA est une application web qui charge une seule page HTML au départ, puis met à jour son contenu dynamiquement à l'aide de JavaScript en communiquant avec une API pour récupérer et afficher les données. Au lieu de recharger toute la page lorsqu'un utilisateur interagit avec l'application, une SPA met à jour uniquement les parties nécessaires de l'interface, offrant ainsi une expérience utilisateur plus fluide et rapide.

Vue.js facilite la création de SPAs grâce à son approche basée sur les composants. Un composant Vue est une unité réutilisable qui encapsule à la fois le HTML, le CSS et le JavaScript nécessaires pour afficher une partie spécifique de l'interface utilisateur. Les composants peuvent être imbriqués les uns dans les autres pour créer des interfaces complexes.

Vue.js fournit un système de liaison de données bidirectionnelles, ce qui signifie que les modifications apportées aux données dans l'application sont automatiquement reflétées dans l'interface utilisateur, et vice versa. Cela permet de créer des applications réactives, où les changements de données sont immédiatement visibles à l'écran.

En utilisant Vue.js pour créer des SPAs, les développeurs peuvent organiser leur code de manière modulaire et maintenable. Vue.js offre également une excellente documentation, une communauté active et de nombreuses bibliothèques et plugins qui peuvent être utilisés pour étendre ses fonctionnalités.

En résumé, Vue.js est un framework JavaScript populaire qui facilite la création de SPAs réactives et dynamiques. Son approche basée sur les composants et son système de liaison de données en font un choix populaire pour le développement d'applications web modernes.

1.Pourquoi VueJs

Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, carte de visite

Description générée automatiquement

Vue.js est un framework JavaScript populaire qui a gagné en popularité en raison de ses nombreux avantages et fonctionnalités attrayantes. Voici quelques raisons pour lesquelles Vue.js est souvent choisi comme framework de développement :

1. Simplicité : Vue.js est réputé pour sa simplicité et son approche intuitive. Son API simple et cohérente facilite l'apprentissage et la prise en main rapide pour les développeurs.
2. Composabilité : Vue.js encourage le développement d'applications en utilisant des composants réutilisables. Les composants Vue.js permettent de découper l'interface utilisateur en petites parties autonomes, ce qui facilite la maintenance et la réutilisation du code.
3. Réactivité : Vue.js utilise un système de rendu réactif basé sur les dépendances. Cela signifie que les vues sont automatiquement mises à jour lorsque les données sous-jacentes changent. Cette approche permet de créer des interfaces utilisateur dynamiques et réactives avec peu d'effort.
4. Flexibilité : Vue.js est un framework flexible qui s'adapte aux besoins du développeur. Vous pouvez commencer par de petites intégrations Vue.js sur une page existante, ou construire une application complète en utilisant Vue.js comme framework principal.
5. Écosystème solide : Vue.js bénéficie d'une communauté active et d'un écosystème solide. Il dispose d'une grande variété de bibliothèques tierces, de plugins et d'outils qui facilitent le développement d'applications.
6. Performances : Vue.js est conçu pour être léger et performant. Il utilise un système de rendu virtuel efficace et optimise les mises à jour de l'interface utilisateur, ce qui se traduit par des performances rapides et réactives.
7. Documentations complètes : Vue.js dispose d'une documentation complète et bien organisée, ainsi que de nombreux tutoriels et exemples. Cela facilite l'apprentissage et la résolution des problèmes lors du développement.

En raison de ces caractéristiques, Vue.js est devenu un choix populaire pour le développement d'applications web modernes, qu'il s'agisse de petites intégrations sur une page existante ou de projets d'envergure.

2. Vue 3

Vue.js 3 apporte plusieurs avantages et améliorations par rapport aux versions précédentes. Voici quelques-uns des principaux avantages de Vue.js 3 :

1. Performances améliorées : Vue.js 3 est plus rapide que les versions précédentes grâce à l'utilisation du système de rendu virtuel optimisé. Il utilise également un nouveau compilateur qui génère un code plus efficace, ce qui se traduit par des performances améliorées lors du rendu de l'interface utilisateur.
2. Taille réduite : Vue.js 3 a une empreinte plus légère en termes de taille du bundle. Il utilise un nouveau système de compilation appelé "Compiler 2.0" qui génère un code plus concis et élimine les fonctionnalités inutilisées, ce qui réduit la taille finale de l'application.
3. Composition API : La Composition API est une nouvelle façon de structurer et de réutiliser la logique des composants dans Vue.js 3. Elle permet de mieux organiser le code en séparant la logique en "composition functions", ce qui facilite la réutilisation et la compréhension du code.
4. Meilleure typage : Vue.js 3 est livré avec un support amélioré pour TypeScript, ce qui permet de bénéficier d'une vérification statique des types et d'une meilleure intégration avec les outils de développement. Cela améliore la robustesse et la maintenabilité des applications.
5. Portée des slots améliorée : Les slots sont une fonctionnalité clé de Vue.js permettant de passer du contenu à un composant parent. Dans Vue.js 3, la portée des slots a été améliorée pour offrir plus de flexibilité et de contrôle lors de l'utilisation des slots.
6. Optimisation du rendu : Vue.js 3 introduit de nouvelles fonctionnalités telles que le rendu conditionnel basé sur les fragments, l'évitement des re-renders inutiles et l'amélioration de la gestion des clés pour les rendus de liste. Ces optimisations contribuent à des performances accrues et à une expérience utilisateur plus fluide.
7. Écosystème actif : Vue.js 3 s'appuie sur la solide communauté et l'écosystème existant de Vue.js. De nombreuses bibliothèques et outils tiers sont déjà compatibles avec Vue.js 3, ce qui facilite l'adoption et l'intégration dans les projets existants.

Ces améliorations font de Vue.js 3 une version plus performante, plus légère et plus flexible, offrant une expérience de développement améliorée par rapport aux versions précédentes.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquementUne image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

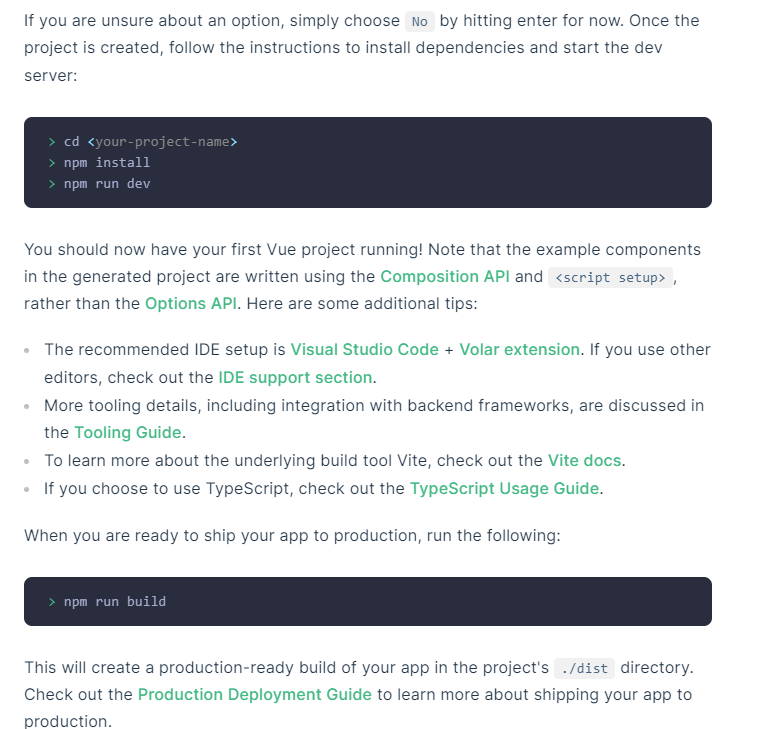
Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Basics
2. Installation

Une image contenant texte, Appareils électroniques, capture d’écran, Police

Description générée automatiquementUne image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

1. Data && Template && clickEvent && Conditional rendring
   1. Data and template

En Vue.js, les options **data** et **template** sont utilisées pour définir les données et le template d'un composant. Voici comment vous pouvez les utiliser :

1. L'option **data** : L'option **data** est utilisée pour définir les données du composant, c'est-à-dire les variables qui seront utilisées dans le template. Vous pouvez définir **data** comme une fonction qui renvoie un objet contenant les données. Voici un exemple :

export default {

    data() {

      return {

        message: 'Bonjour, Vue.js!'

      };

    }

  }

Dans cet exemple, nous avons défini une seule propriété **message** dans l'objet de données. Vous pouvez ensuite utiliser cette propriété dans le template du composant en utilisant la syntaxe **{{ message }}**.

1. L'option **template** : L'option **template** est utilisée pour définir la structure et le contenu du template HTML du composant. Vous pouvez le définir en tant que chaîne de caractères ou en utilisant des balises **<template>**. Voici un exemple :

export default {

    template: `

      <div>

        <h1>{{ message }}</h1>

        <button @click="changeMessage">Changer le message</button>

      </div>

    `,

    data() {

      return {

        message: 'Bonjour, Vue.js!'

      };

    },

    methods: {

      changeMessage() {

        this.message = 'Nouveau message!';

      }

    }

  }

Dans cet exemple, nous avons défini le template en utilisant une chaîne de caractères multi-lignes entourée de backticks (). Le template contient une balise **<h1>** qui affiche la valeur de la propriété **message** grâce à la syntaxe **{{ message }}**, ainsi qu'un bouton qui appelle la méthode **changeMessage** lorsque cliqué.

Il existe également d'autres façons de définir le template, telles que l'utilisation d'un sélecteur CSS pour cibler un élément existant dans la page ou la séparation du template dans un fichier externe et son importation dans le composant. Ces méthodes permettent une meilleure séparation des préoccupations et une meilleure organisation du code.

En utilisant les options **data** et **template**, vous pouvez définir les données et le contenu du template d'un composant Vue.js, ce qui permet de créer des interfaces utilisateur dynamiques et réactives.

* 1. ClickEvent

En Vue.js, pour gérer les événements de clic, vous pouvez utiliser la directive **v-on** (ou son raccourci **@**) pour attacher un écouteur d'événement à un élément HTML. Voici comment vous pouvez utiliser **v-on** pour gérer un événement de clic :

1. Écouteur d'événement en ligne :

<button v-on:click="handleClick">Cliquez ici</button>

Dans cet exemple, nous utilisons **v-on:click** pour attacher un écouteur d'événement de clic au bouton. Lorsque le bouton est cliqué, la méthode **handleClick** sera appelée.

1. Écouteur d'événement avec une expression en ligne :

<button v-on:click="counter++">Incrémenter</button>

Dans cet exemple, nous utilisons **v-on:click** pour incrémenter une variable **counter** chaque fois que le bouton est cliqué.

1. Écouteur d'événement avec une méthode définie dans les options du composant :

<button v-on:click="incrementCounter">Incrémenter</button>

export default {

    methods: {

      incrementCounter() {

        this.counter++;

      }

    }

  }

Dans cet exemple, nous définissons une méthode **incrementCounter** dans les options du composant. Lorsque le bouton est cliqué, la méthode **incrementCounter** est appelée.

Vous pouvez également utiliser la syntaxe du raccourci **@** pour simplifier le code :

<button @click="handleClick">Cliquez ici</button>

En résumé, **v-on:click** ou **@click** permet d'attacher un écouteur d'événement de clic à un élément, et vous pouvez définir une méthode ou une expression en ligne pour gérer cet événement dans le contexte de votre composant Vue.js.

* 1. Conditional rendring

En Vue.js, vous pouvez utiliser le rendu conditionnel pour afficher ou masquer des éléments en fonction d'une condition spécifique. Il existe plusieurs façons de réaliser un rendu conditionnel dans Vue.js. Voici quelques exemples :

1. Directive **v-if** : La directive **v-if** est utilisée pour conditionner l'affichage d'un élément en fonction d'une expression booléenne. Si l'expression est évaluée à **true**, l'élément est rendu ; sinon, il est supprimé du DOM. Voici un exemple :
2. <div v-if="condition">
3. Contenu affiché si la condition est vraie.
4. </div>

Dans cet exemple, l'élément **<div>** sera rendu si la valeur de la propriété **condition** est **true**.

1. Directive **v-else** : La directive **v-else** est utilisée en conjonction avec **v-if** pour afficher un élément alternatif lorsque la condition de **v-if** est fausse. Voici un exemple :

<div v-if="condition">

    Contenu affiché si la condition est vraie.

  </div>

  <div v-else>

    Contenu affiché si la condition est fausse.

  </div>

Dans cet exemple, si la valeur de **condition** est **true**, le premier **<div>** sera rendu, sinon le deuxième **<div>** sera rendu.

1. Directive **v-else-if** : La directive **v-else-if** est utilisée pour ajouter une condition supplémentaire à un rendu conditionnel en chaîne. Voici un exemple :

<div v-if="condition1">

    Contenu affiché si condition1 est vraie.

  </div>

  <div v-else-if="condition2">

    Contenu affiché si condition1 est fausse et condition2 est vraie.

  </div>

  <div v-else>

    Contenu affiché si toutes les conditions précédentes sont fausses.

  </div>

Dans cet exemple, si **condition1** est **true**, le premier **<div>** sera rendu. Sinon, si **condition2** est **true**, le deuxième **<div>** sera rendu. Sinon, le troisième **<div>** sera rendu.

1. Utilisation de l'opérateur ternaire : Vous pouvez également utiliser l'opérateur ternaire dans les interpolations (**{{ }}**) pour réaliser un rendu conditionnel dans une expression. Voici un exemple :

<p>{{ condition ? 'Condition vraie' : 'Condition fausse' }}</p>

Dans cet exemple, si **condition** est **true**, le texte "Condition vraie" sera affiché, sinon le texte "Condition fausse" sera affiché.

Ces techniques de rendu conditionnel vous permettent de contrôler dynamiquement l'affichage des éléments dans votre template en fonction de certaines conditions.

* 1. Differnce entre v-if et v-show

1. **v-show** est plus adapté lorsque vous avez besoin de masquer ou afficher rapidement un élément sans affecter son état. Il est plus performant pour les éléments qui changent souvent entre visible et invisible.
2. **v-if** est plus adapté lorsque vous avez besoin de conditionner l'existence d'un élément et que son état peut être complètement supprimé du DOM. Il est utile lorsque vous avez des éléments qui ne sont pas souvent affichés et qui ont un état qui doit être réinitialisé lorsqu'ils sont retirés du DOM.

En général, si vous avez besoin de masquer et afficher fréquemment un élément, **v-show** est plus approprié. Si vous avez besoin de conditionner l'existence d'un élément avec des changements d'état, **v-if** est plus approprié.

const app = Vue.createApp({

  // data,functions

  //   template: "<h2>I am the template </h2>",

  data() {

    return {

      showBooks: true,

      title: "The final Empire",

      author: "Brandon Sanderson",

      age: 45,

    };

  },

  methods: {

    toggleShowBooks() {

      this.showBooks = !this.showBooks;

    },

  },

});

app.mount("#app");

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Learning Vue</title>

    <script src="https://unpkg.com/vue@3.0.2"></script>

  </head>

  <body>

    <h1>Hello Vue</h1>

    <div id="app">

      <!-- Id app Relation avec app.mount("#app")  -->

      <div v-if="showBooks">

        <p>{{title}} -{{author}} -{{age}}</p>

      </div>

      <button @click="toggleShowBooks">

        <span v-if="showBooks">Hide Books</span>

        <span v-else>Show Books</span>

      </button>

      <div v-show="showBooks">currently showing books</div>

    </div>

    <script src="app.js"></script>

  </body>

</html>

1. v-for && Attribute binding && computed properties

3.1. v-for

La directive **v-for** est utilisée en Vue.js pour itérer sur une liste ou un tableau d'éléments et générer du contenu répété dans le template. Elle permet de créer des boucles dans le template en utilisant les données existantes dans le composant.

Voici comment utiliser la directive **v-for** en Vue.js :

1. Itération sur un tableau : Vous pouvez utiliser **v-for** pour itérer sur un tableau et générer du contenu répété. Voici un exemple :

<ul>

  <li v-for="item in items" :key="item.id">{{ item.name }}</li>

</ul>

Dans cet exemple, nous itérons sur le tableau **items** du composant et générons une balise **<li>** pour chaque élément du tableau. Chaque élément est accessible dans la boucle sous le nom **item**. L'attribut **:key** est utilisé pour donner une clé unique à chaque élément itéré afin d'améliorer les performances lors de la mise à jour de la liste.

1. Itération sur un objet : **v-for** peut également être utilisé pour itérer sur les propriétés d'un objet. Voici un exemple :

<ul>

  <li v-for="(value, key) in object">{{ key }}: {{ value }}</li>

</ul>

Dans cet exemple, nous itérons sur les propriétés de l'objet **object** et générons une balise **<li>** pour chaque propriété. Les variables **key** et **value** représentent respectivement la clé et la valeur de chaque propriété.

1. Obtention de l'index de l'itération : Si vous avez besoin d'accéder à l'index de chaque élément itéré, vous pouvez utiliser la syntaxe suivante :

<ul>

  <li v-for="(item, index) in items" :key="item.id">{{ index + 1 }}. {{ item.name }}</li>

</ul>

Dans cet exemple, nous utilisons **index** pour obtenir l'index de l'itération et l'afficher avec chaque élément.

1. Utilisation de la directive **v-for** avec des composants : Vous pouvez également utiliser **v-for** avec des composants personnalisés pour générer des instances répétées de composants. Voici un exemple :

<template>

  <div>

    <my-component v-for="item in items" :key="item.id" :item="item"></my-component>

  </div>

</template>

Dans cet exemple, nous itérons sur le tableau **items** et générons une instance du composant personnalisé **my-component** pour chaque élément.

La directive **v-for** est un puissant outil de Vue.js pour générer du contenu répété dans le template en fonction des données du composant. Elle offre une grande flexibilité pour itérer sur des tableaux, des objets et même des composants personnalisés.

3.2.

3.2.1. Attribute binding

L'« attribute binding » (liaison d'attribut) en Vue.js permet de lier dynamiquement des valeurs aux attributs des éléments HTML dans le template. Cela permet de modifier les attributs en fonction des données du composant ou de l'état de l'application.

Pour effectuer une liaison d'attribut en Vue.js, vous pouvez utiliser la syntaxe **v-bind** ou le raccourci **:** suivi du nom de l'attribut. Voici un exemple :

<template>

  <div>

    <img :src="imageUrl" :alt="imageAlt">

    <a :href="linkUrl">{{ linkText }}</a>

    <input type="text" :value="inputValue">

  </div>

</template>

Dans cet exemple, nous utilisons **:src**, **:alt**, **:href** et **:value** pour effectuer des liaisons d'attribut avec les valeurs des propriétés correspondantes du composant.

Lorsque les valeurs des propriétés sont mises à jour dans le composant, les attributs correspondants sont également mis à jour dans le DOM.

Il est également possible de combiner des valeurs statiques et dynamiques dans une liaison d'attribut. Par exemple :

<template>

  <div>

    <a :class="['button', buttonType]">Cliquez ici</a>

  </div>

</template>

Dans cet exemple, nous utilisons un tableau pour lier dynamiquement la classe d'un élément **<a>**. La classe "button" est une valeur statique tandis que **buttonType** est une propriété du composant qui fournit une valeur dynamique.

En utilisant la liaison d'attribut, vous pouvez rendre votre code plus flexible et réactif, en permettant aux attributs HTML d'être mis à jour dynamiquement en fonction de l'état de votre application ou des données du composant.

3.2.2. Attribute binding and condition en vue Js

La liaison d'attribut et les conditions sont deux concepts distincts en Vue.js, mais ils peuvent être utilisés ensemble pour rendre les attributs conditionnels dans le template en fonction de certaines conditions.

Voici comment combiner la liaison d'attribut et les conditions en Vue.js :

1. Utilisation de **v-bind** avec une condition : La directive **v-bind** (ou son raccourci **:**) peut être utilisée en conjonction avec une condition pour lier un attribut uniquement lorsque la condition est vraie. Voici un exemple :

<template>

    <div>

      <button :class="{ active: isActive }">Cliquez ici</button>

    </div>

  </template>

Dans cet exemple, nous utilisons **:class** pour lier la classe "active" à l'attribut **class** du bouton uniquement lorsque la propriété **isActive** est évaluée à **true**.

1. Utilisation de **v-if** pour conditionner l'existence d'un attribut : La directive **v-if** peut être utilisée pour conditionner l'existence d'un attribut dans le template en fonction d'une condition. Voici un exemple :

<template>

    <div>

      <button v-if="isEnabled" disabled>Cliquez ici</button>

    </div>

  </template>

Dans cet exemple, l'attribut **disabled** est ajouté à l'élément **<button>** uniquement lorsque la propriété **isEnabled** est évaluée à **true**. Si la condition est fausse, l'attribut **disabled** n'est pas inclus dans l'élément.

1. Combinaison de **v-bind** et **v-if** : Il est également possible de combiner **v-bind** et **v-if** pour lier un attribut conditionnellement et conditionner son existence en même temps. Voici un exemple :

<template>

    <div>

      <button v-if="isButtonVisible" :class="{ active: isActive }" :disabled="isDisabled">Cliquez ici</button>

    </div>

  </template>

Dans cet exemple, l'attribut **disabled** est lié à la propriété **isDisabled** en utilisant **:disabled**, et l'attribut **class** est lié conditionnellement à la propriété **isActive** en utilisant **:class**. De plus, l'existence de l'élément **<button>** est conditionnée par la propriété **isButtonVisible** en utilisant **v-if**.

En combinant la liaison d'attribut et les conditions, vous pouvez rendre les attributs conditionnels dans votre template, en les liant à des valeurs dynamiques en fonction de certaines conditions définies dans le composant. Cela vous permet de rendre votre code plus flexible et réactif.