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

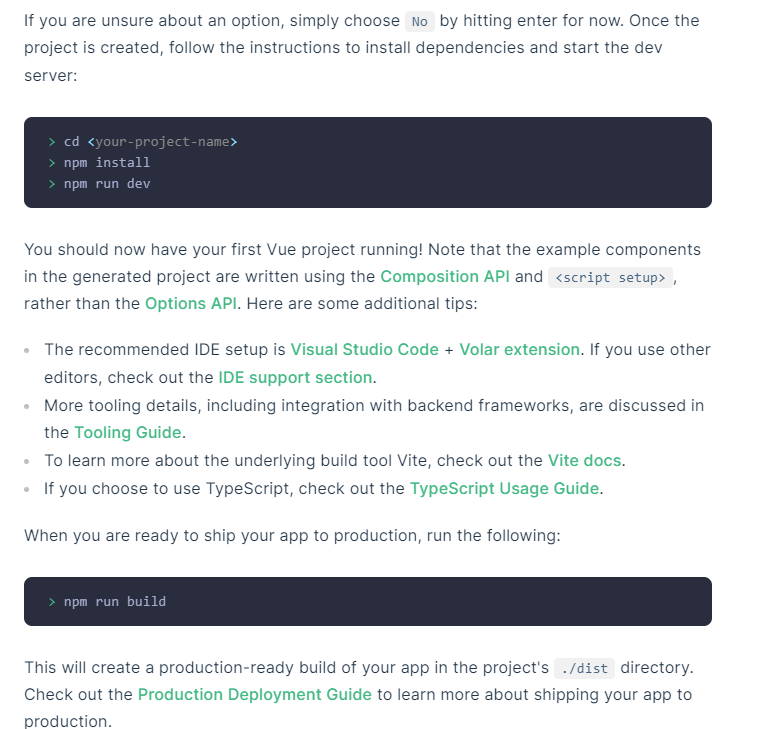
Description générée automatiquement

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.

3.3. computed properties

En Vue.js, les options **methods** et **computed** sont utilisées pour définir des fonctionnalités dans un composant, mais elles ont des différences importantes quant à leur utilisation et leur comportement.

1. **methods** : L'option **methods** est utilisée pour définir des méthodes dans un composant Vue.js. Les méthodes sont des fonctions qui peuvent être appelées depuis le template ou d'autres parties du composant. Voici un exemple :

export default {

    data() {

      return {

        count: 0

      };

    },

    methods: {

      increment() {

        this.count++;

      },

      decrement() {

        this.count--;

      }

    }

  };

Dans cet exemple, nous avons défini deux méthodes **increment** et **decrement** qui modifient la valeur de la propriété **count**.

Les méthodes sont généralement utilisées pour effectuer des actions ou des opérations qui nécessitent des interactions utilisateur, des événements, des appels à des API, etc. Elles sont appelées de manière explicite dans le code ou via des événements dans le template.

1. **computed** : L'option **computed** est utilisée pour définir des propriétés calculées dans un composant Vue.js. Les propriétés calculées sont des valeurs dérivées de manière réactive à partir des données existantes dans le composant. Voici un exemple :

export default {

    data() {

      return {

        width: 10,

        height: 5

      };

    },

    computed: {

      area() {

        return this.width \* this.height;

      }

    }

  };

Dans cet exemple, nous avons défini une propriété calculée **area** qui renvoie le produit des propriétés **width** et **height**.

Les propriétés calculées sont évaluées dynamiquement en fonction de leurs dépendances (dans cet exemple, **width** et **height**). Elles sont mises en cache et ne sont recalculées que lorsque leurs dépendances changent. Les propriétés calculées sont souvent utilisées pour effectuer des calculs complexes ou des transformations sur les données.

Comparaison :

* Les méthodes (**methods**) sont utilisées pour effectuer des opérations et des actions dans le composant, généralement en réponse à des événements ou des interactions utilisateur. Elles sont appelées de manière explicite et peuvent modifier les données du composant.
* Les propriétés calculées (**computed**) sont utilisées pour calculer des valeurs dérivées de manière réactive à partir des données existantes dans le composant. Elles sont évaluées automatiquement et fournissent une valeur mise à jour en fonction des dépendances. Les propriétés calculées ne modifient pas directement les données du composant, mais renvoient des valeurs calculées.

En général, utilisez les méthodes lorsque vous avez besoin d'effectuer des actions ou des opérations avec des effets de côté, et utilisez les propriétés calculées lorsque vous avez besoin de calculer des valeurs dérivées de manière réactive.

1. The Vue Cli & Bigger Projects

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

Description générée automatiquement

* 1. Vue Cli Definition

Vue CLI (Command Line Interface) est un outil de développement en ligne de commande pour créer, gérer et déployer des applications Vue.js. Vue CLI facilite la création de nouveaux projets Vue.js en fournissant une structure de projet prédéfinie et des fonctionnalités prêtes à l'emploi.

Avec Vue CLI, vous pouvez générer un projet Vue.js à partir de zéro ou utiliser des modèles préconfigurés pour des projets spécifiques tels que des applications Web, des applications mobiles, des Progressive Web Apps (PWA) et des applications avec des configurations avancées.

L'un des principaux avantages de Vue CLI est sa simplicité d'utilisation et sa facilité de configuration. Il offre une interface en ligne de commande intuitive pour générer du code, gérer des dépendances, exécuter des tâches de développement, créer des versions de production optimisées et déployer votre application.

Vue CLI fournit également des fonctionnalités de développement avancées telles que le rechargement à chaud (hot-reloading), qui permet de voir instantanément les modifications apportées au code sans avoir à rafraîchir manuellement la page. Il intègre également des outils de test, de linting (analyse de la qualité du code) et de gestion des plugins.

En résumé, Vue CLI est un outil puissant pour développer des applications Vue.js de manière efficace en fournissant une infrastructure prête à l'emploi, des fonctionnalités avancées et une expérience de développement fluide.

* 1. Template Refs en VueJs

Les "template refs" en Vue.js sont une fonctionnalité qui permet de référencer des éléments spécifiques dans un template Vue et d'interagir avec eux directement dans le code JavaScript.

Pour utiliser les "template refs", vous pouvez utiliser la directive **ref** dans votre template pour donner un nom à un élément. Par exemple :

<template>

  <div>

    <input ref="myInput" type="text">

    <button @click="focusInput">Focus Input</button>

  </div>

</template>

Dans cet exemple, la directive **ref="myInput"** est utilisée pour référencer l'élément **<input>**. Cela crée une référence nommée "myInput" pour cet élément.

Ensuite, dans le code JavaScript correspondant, vous pouvez accéder à cet élément en utilisant la référence créée. Par exemple :

<script>

export default {

  methods: {

    focusInput() {

      this.$refs.myInput.focus();

    }

  }

}

</script>

Dans cette méthode **focusInput**, **this.$refs.myInput** fait référence à l'élément **<input>** grâce à la référence nommée "myInput". L'appel à la méthode **focus()** permet de mettre le focus sur l'élément.

Les "template refs" sont utiles lorsque vous avez besoin d'accéder directement à des éléments spécifiques dans votre template, par exemple pour effectuer des manipulations DOM ou pour accéder à des méthodes et propriétés de ces éléments.

Il convient de noter que les "template refs" ne fonctionnent que sur les éléments définis dans le template du composant courant. Si vous avez besoin d'accéder à des éléments dans un composant enfant, vous pouvez utiliser les "refs" conventionnels en leur attribuant une référence à l'aide de **ref** et en accédant à cette référence via **this.$refs** dans le composant parent.

* 1. Multiple components

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

Description générée automatiquement

En Vue.js, la création et l'utilisation de plusieurs composants sont courantes et encouragées pour organiser et réutiliser efficacement votre code. Voici comment vous pouvez créer et utiliser plusieurs composants dans Vue.js :

1. **Création de composants :** Vous pouvez créer des composants en utilisant la syntaxe **Vue.component** ou en définissant des composants locaux dans un fichier **.vue**.

// Création d'un composant global

Vue.component('mon-composant', {

  // Options du composant

});

// Création d'un composant local dans un fichier .vue

<template>

  <!-- Structure du composant -->

</template>

<script>

export default {

  // Options du composant

};

</script>

**Utilisation de composants :** Une fois que vous avez créé des composants, vous pouvez les utiliser dans d'autres composants ou dans votre application principale.

<template>

  <div>

    <mon-composant></mon-composant>

    <autre-composant></autre-composant>

  </div>

</template>

<script>

import MonComposant from './MonComposant.vue';

import AutreComposant from './AutreComposant.vue';

export default {

  components: {

    MonComposant,

    AutreComposant

  },

  // Autres options du composant

};

</script>

1. Dans cet exemple, les composants **MonComposant** et **AutreComposant** sont importés et enregistrés dans la propriété **components** du composant parent. Vous pouvez ensuite les utiliser dans le template en utilisant leurs noms comme des balises HTML.

Notez que si vous utilisez des composants locaux, vous devez importer les fichiers **.vue** correspondants et les enregistrer dans la propriété **components** du composant parent.

1. **Communication entre composants :** Les composants peuvent communiquer entre eux en utilisant des propriétés (props) pour passer des données du composant parent à l'enfant, et des événements pour émettre des actions de l'enfant vers le parent.

<template>

  <div>

    <mon-composant :message="texte" @click="handleClick"></mon-composant>

  </div>

</template>

<script>

import MonComposant from './MonComposant.vue';

export default {

  components: {

    MonComposant

  },

  data() {

    return {

      texte: 'Bonjour'

    };

  },

  methods: {

    handleClick() {

      // Gérer l'événement

    }

  }

};

</script>

1. Dans cet exemple, le composant parent passe la valeur de **texte** au composant enfant **MonComposant** via la propriété **message**. Le composant enfant peut accéder à cette valeur en tant que prop et l'utiliser dans son template. De plus, le composant enfant peut émettre un événement **click** qui sera capturé par le composant parent et géré dans la méthode **handleClick**.

C'est ainsi que vous pouvez créer et utiliser plusieurs composants dans Vue.js. Cette approche modulaire favorise la réutilisabilité, la maintenabilité et la lisibilité de votre code.

* 1. Scoped and global css

En Vue.js, vous avez la possibilité d'appliquer des styles CSS globaux et des styles CSS encapsulés (scoped) à vos composants. Cela vous permet de définir des styles qui s'appliquent soit globalement à toute l'application, soit uniquement à un composant spécifique. Voici comment fonctionnent les styles CSS globaux et les styles CSS encapsulés dans Vue.js :

**Styles CSS globaux :** Les styles CSS globaux sont des styles qui s'appliquent à l'ensemble de votre application.

**App.Vue**

<style >

#app {

  font-family: Avenir, Helvetica, Arial, sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  text-align: center;

  color: #2c3e50;

  margin-top: 60px;

}

h1 {

  color: red;

  border-bottom: 1px solid #ddd;

  display: inline block;

  padding-bottom: 10px;

}

</style>

**Ou cree un fichier global.css dans assets et l’impoerter dans main.js**

body {

  margin: 0;

}

p {

  font-style: italic;

}

**Styles CSS encapsulés (scoped) :** Les styles CSS encapsulés sont des styles qui sont spécifiques à un composant particulier et ne s'appliquent qu'à ce composant. Ils sont encapsulés dans le composant lui-même pour éviter les conflits de styles entre les différents composants.

Dans un composant Vue, vous pouvez utiliser l'attribut **scoped** dans la balise **<style>** pour activer le mode d'encapsulation des styles

<template>

  <div class="my-component">

    <!-- Contenu du composant -->

  </div>

</template>

<style scoped>

/\* Styles CSS encapsulés \*/

.my-component {

  background-color: #f5f5f5;

  /\* ... \*/

}

/\* ... \*/

</style>

Dans cet exemple, les styles définis dans la balise **<style scoped>** s'appliqueront uniquement au composant dans lequel ils sont déclarés. Les sélecteurs CSS définis dans les styles encapsulés seront automatiquement étendus pour être spécifiques au composant.

Il convient de noter que les styles encapsulés ne sont pas entièrement isolés. Ils sont transformés par Vue.js à la compilation en utilisant des attributs de données uniques pour chaque élément du composant, ce qui permet de restreindre la portée des styles. Cependant, les styles peuvent encore être affectés par des styles parent ou par des sélecteurs CSS plus spécifiques.

Utiliser des styles globaux ou des styles encapsulés dépend de vos besoins spécifiques. Les styles globaux sont utiles pour les styles qui doivent s'appliquer de manière cohérente à toute l'application, tandis que les styles encapsulés offrent une encapsulation plus stricte et évitent les conflits de styles entre les composants.

* 1. Props

En Vue.js, les props (propriétés) sont utilisées pour transmettre des données d'un composant parent à un composant enfant. Les props permettent de rendre les composants plus modulaires, réutilisables et flexibles en leur fournissant des données externes.

Voici comment vous pouvez définir et utiliser des props dans Vue.js :

**Passage des props :** Dans le composant parent, vous pouvez passer des valeurs aux props lors de l'utilisation du composant enfant. Cela se fait en ajoutant des attributs à la balise du composant enfant.

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

Description générée automatiquement

1. Dans cet exemple, le composant parent utilise le composant **mon-composant** et lui passe les valeurs des props **message** et **count**. Le **:** avant **count** indique que la valeur de **count** est une expression JavaScript.
2. **Utilisation des props :** Dans le composant enfant, vous pouvez utiliser les props passées dans le template, dans les méthodes ou n'importe où dans le composant.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Dans cet exemple, le composant enfant utilise les props **message** et **count** dans le template en utilisant les doubles accolades **{{ }}** pour l'interpolation des données.

Les props sont une façon efficace de transmettre des données entre les composants et de favoriser la réutilisabilité. Ils permettent de rendre les composants plus flexibles et modulaires en leur fournissant des données externes.

Example

<template>

  <div class="backdrop" @click.self="toogleModal">

    <div class="modal" :class="{ sale: theme === 'sale' }">

      <h1>{{ header }}</h1>

      <p>{{ text }}</p>

    </div>

  </div>

</template>

<script>

export default {

  props: ["header", "text", "theme","toogleModal"]

};

</script>

* 1. Slots

Les slots sont une fonctionnalité puissante de Vue.js qui permet de définir des zones de contenu personnalisées dans un composant, où le contenu peut être passé depuis le composant parent. Les slots permettent une meilleure flexibilité et réutilisabilité des composants en leur permettant d'accepter du contenu dynamique.

Voici comment utiliser les slots en Vue.js :

1. **Définition d'un slot dans le composant enfant :** Dans le composant enfant, vous pouvez définir des zones de contenu en utilisant la balise **<slot>**. Vous pouvez nommer les slots en utilisant l'attribut **name** si vous souhaitez avoir plusieurs slots dans un même composant.

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

Description générée automatiquement

1. Dans cet exemple, le composant enfant définit un slot sans nom à l'aide de la balise **<slot>**. Tout contenu passé depuis le composant parent sera inséré à cet emplacement.
2. **Utilisation du composant enfant avec du contenu personnalisé :** Dans le composant parent, vous pouvez utiliser le composant enfant et fournir du contenu personnalisé à insérer dans le slot. Cela se fait en plaçant le contenu entre les balises d'ouverture et de fermeture du composant enfant.

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

Description générée automatiquement

1. Dans cet exemple, le composant parent utilise le composant enfant **mon-composant** et insère le contenu **<p>Contenu personnalisé inséré dans le slot</p>** à l'endroit où le slot est défini dans le composant enfant.

Vous pouvez également fournir du contenu dynamique en utilisant des expressions et des données du composant parent.

1. **Utilisation de slots nommés :** Si vous souhaitez avoir plusieurs slots dans un même composant enfant, vous pouvez les nommer en utilisant l'attribut **name**. Vous pouvez ensuite les référencer en utilisant la balise **<slot>** avec l'attribut **name** correspondant.

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

Description générée automatiquement

Dans cet exemple, le composant enfant définit trois slots : **header**, un slot sans nom et **footer**. Vous pouvez ensuite utiliser ces slots nommés dans le composant parent en utilisant la balise **<template>** pour regrouper le contenu correspondant à chaque slot.

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

Description générée automatiquement

1. Dans cet exemple, le composant parent utilise le composant enfant **mon-composant** et fournit du contenu pour chaque slot nommé en utilisant la syntaxe **v-slot:** suivie du nom du slot.

Les slots sont extrêmement utiles pour créer des composants flexibles et réutilisables, permettant aux composants enfants d'accepter et d'afficher du contenu dynamique provenant de leur composant parent. Ils offrent une grande souplesse dans la composition des interfaces utilisateur en Vue.js.

1. Reaction Timer Project

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

Description générée automatiquement

* 1. LifeCycle Diagram

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

Description générée automatiquement

En Vue.js, les "lifecycle hooks" (ou "hooks de cycle de vie") sont des méthodes prédéfinies que vous pouvez ajouter à vos composants pour exécuter des actions à des moments spécifiques du cycle de vie d'un composant. Ces hooks vous permettent de contrôler le comportement du composant lors de sa création, de sa mise à jour et de sa destruction. Voici les principaux hooks de cycle de vie disponibles :

1. **beforeCreate() :** Ce hook est appelé immédiatement après la création du composant, avant que les données et les événements ne soient initialisés. Les propriétés et méthodes du composant ne sont pas encore accessibles à ce stade.
2. **created() :** Ce hook est appelé lorsque le composant est créé et initialisé, les données, les méthodes, les propriétés calculées et les observateurs ont été configurés. C'est un bon endroit pour effectuer des appels réseau initiaux ou des initialisations asynchrones.
3. **beforeMount() :** Ce hook est appelé avant que le composant ne soit inséré dans le DOM. À ce stade, le template du composant a été compilé et est prêt à être rendu.
4. **mounted() :** Ce hook est appelé une fois que le composant a été inséré dans le DOM. C'est le moment idéal pour accéder au DOM du composant et interagir avec des bibliothèques tierces via des plugins.
5. **beforeUpdate() :** Ce hook est appelé avant que les données du composant ne soient mises à jour et que le re-rendering soit déclenché. Vous pouvez utiliser ce hook pour effectuer des opérations spécifiques avant que les changements ne soient appliqués.
6. **updated() :** Ce hook est appelé après que les données du composant ont été mises à jour et que le re-rendering a été effectué. C'est un bon endroit pour effectuer des opérations supplémentaires après une mise à jour, mais soyez prudent pour éviter les boucles infinies en modifiant les données du composant ici.
7. **beforeUnmount() :** Ce hook est appelé avant que le composant ne soit retiré du DOM. Vous pouvez utiliser ce hook pour nettoyer les ressources, annuler les abonnements aux événements ou arrêter les minuteries.
8. **unmounted() :** Ce hook est appelé une fois que le composant a été retiré du DOM. À ce stade, le composant n'est plus actif et toutes les références à ses données et méthodes doivent être supprimées.

Il est important de noter que les hooks de cycle de vie ne sont disponibles que pour les composants Vue.js en instance (composants déclarés avec **new Vue()**). Pour les composants Vue.js en simple fichier (Single File Components), vous pouvez utiliser les hooks fournis par le système de compilation, tels que **beforeCreate**, **created**, **beforeMount**, etc.

Les hooks de cycle de vie sont un outil puissant pour gérer le comportement des composants à différents moments de leur cycle de vie. Ils vous permettent de réaliser des actions spécifiques à chaque étape et de contrôler le flux de votre application Vue.js.

<template>

  <div class="block" v-if="showBlock">Click me</div>

</template>

<script>

export default {

  props: ["delay"],

  data() {

    return {

      showBlock: false,

    };

  },

  mounted() {

    console.log("component mounted");

    setTimeout(() => {

      this.showBlock = true;

    }, this.delay);

  },

  updated() {

    console.log("Component Updated");

  },

  unmounted() {

    console.log("Unmounted");

  },

};

</script>

* 1. Custom events

En Vue.js, la directive **v-on** avec l'argument **@** est utilisée pour écouter et répondre aux événements déclenchés par un composant enfant. Pour communiquer de l'enfant vers le parent, vous pouvez utiliser la fonction **this.$emit()** dans le composant enfant pour déclencher un événement personnalisé. Voici comment utiliser **emit** en Vue.js :

1. **Définition de l'événement personnalisé dans le composant enfant :** Dans le composant enfant, vous pouvez définir un événement personnalisé en utilisant la fonction **this.$emit()** au moment approprié. Cette fonction prend le nom de l'événement en premier argument, suivi des données que vous souhaitez passer au parent.

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

Description générée automatiquement

1. Dans cet exemple, lorsque le bouton est cliqué, la méthode **sendData()** est appelée. Elle utilise **this.$emit()** pour déclencher l'événement personnalisé nommé **'custom-event'** et passe les données **'Données à envoyer'** au parent.
2. **Écoute de l'événement dans le composant parent :** Dans le composant parent, vous pouvez utiliser la directive **v-on** (ou son raccourci **@**) pour écouter l'événement personnalisé déclenché par le composant enfant. Lorsque l'événement est déclenché, vous pouvez exécuter une méthode du composant parent pour y répondre.

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

Description générée automatiquement

1. Dans cet exemple, le composant parent utilise le composant enfant **child-component** et utilise la directive **@custom-event** pour écouter l'événement personnalisé. Lorsque l'événement est déclenché, la méthode **handleEvent()** du composant parent est appelée et reçoit les données transmises par le composant enfant.

En utilisant **this.$emit()** dans le composant enfant et en écoutant l'événement avec **v-on** dans le composant parent, vous pouvez établir une communication entre les composants et transmettre des données du composant enfant vers le composant parent. Cela vous permet de mettre à jour l'état du composant parent en fonction des actions effectuées dans le composant enfant.

1. Forms & Input
   1. V modal

En Vue.js, la directive **v-model** est une fonctionnalité pratique qui permet de lier automatiquement les données d'un élément de formulaire à une propriété dans le modèle de données du composant. Elle facilite la synchronisation bidirectionnelle des données entre l'interface utilisateur et le modèle de données. Voici comment utiliser **v-model** en Vue.js :

1. **Liaison de données avec v-model :** Dans le composant, vous pouvez utiliser la directive **v-model** pour lier une propriété du modèle de données à un élément de formulaire, tel qu'un champ de texte (**<input>**) ou une case à cocher (**<checkbox>**).

<!-- Composant -->

<template>

  <div>

    <input type="text" v-model="message">

    <p>{{ message }}</p>

  </div>

</template>

<script>

export default {

  data() {

    return {

      message: ''

    };

  }

};

</script>

1. Dans cet exemple, la propriété **message** est liée à l'élément **<input>** à l'aide de **v-model**. Lorsque l'utilisateur saisit du texte dans le champ de texte, la valeur est automatiquement mise à jour dans la propriété **message** du modèle de données. De même, lorsque la valeur de **message** est mise à jour dans le modèle de données, elle est automatiquement reflétée dans le champ de texte.
2. **Utilisation de v-model avec des composants personnalisés :** Vous pouvez également utiliser **v-model** avec des composants personnalisés en définissant les propriétés **value** et **emit** dans le composant. Cela permet d'utiliser **v-model** avec des composants réutilisables.

<!-- Composant enfant -->

<template>

  <div>

    <input :value="value" @input="$emit('input', $event.target.value)">

  </div>

</template>

<script>

export default {

  props: ['value']

};

</script>

Dans cet exemple, le composant enfant accepte une propriété **value** et émet un événement **input** avec la nouvelle valeur lorsqu'il y a un changement dans l'élément **<input>**. Ainsi, vous pouvez utiliser **v-model** avec ce composant personnalisé, tout comme avec les éléments de formulaire natifs.

<!-- Composant parent -->

<template>

  <div>

    <child-component v-model="message"></child-component>

    <p>{{ message }}</p>

  </div>

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

  components: {

    ChildComponent

  },

  data() {

    return {

      message: ''

    };

  }

};

</script>

1. Dans ce cas, le composant parent utilise **v-model** avec le composant enfant **child-component**. La propriété **message** du composant parent est automatiquement liée à la propriété **value** du composant enfant, et les mises à jour sont synchronisées dans les deux sens.

La directive **v-model** facilite grandement la gestion des données d'entrée des utilisateurs et la synchronisation entre les composants et le modèle de données. Elle vous permet de créer des formulaires interactifs et réactifs sans nécessiter beaucoup de code supplémentaire.

1. Example Input

SignUpForm.vue

<template>

  <div>

    <form @submit="handleSubmit">

      <label>Email :</label>

      <input type="email" required v-model="email" />

      <label>Password:</label>

      <input type="password" required v-model="password" />

      <div v-if="passwordError" class="error">{{ passwordError }}</div>

      <label>Role:</label>

      <select v-model="role">

        <option value="developer">Web Developer</option>

        <option value="designer">Web Designer</option>

      </select>

      <label>Skills:</label>

      <input type="text" v-model="tempSkill" @keyup.alt="addSkill" />

      <div v-for="skill in skills" :key="skill" class="pill">

        <span @click="deleteSkill(skill)"> {{ skill }} </span>

      </div>

      <div class="terms">

        <input type="checkbox" required v-model="terms" />

        <label> Accept terms and conditions </label>

      </div>

      <div>

        <input type="checkbox" value="shaun" v-model="names" />

        <label> Shaun </label>

      </div>

      <div>

        <input type="checkbox" value="Yoshi" v-model="names" />

        <label> Yoshi </label>

      </div>

      <div>

        <input type="checkbox" value="Mario" v-model="names" />

        <label> Mario </label>

      </div>

      <div class="submit">

        <button>Create an Account</button>

      </div>

    </form>

    <p>Email :{{ email }}</p>

    <p>Password :{{ password }}</p>

    <p>Role: {{ role }}</p>

    <p>Terms accepted : {{ terms }}</p>

    <p>Names: {{ names }}</p>

  </div>

</template>

<script>

export default {

  data() {

    return {

      email: "mario",

      password: "",

      role: "designer",

      terms: false,

      names: [],

      tempSkill: "",

      skills: [],

      passwordError: "",

    };

  },

  methods: {

    addSkill(e) {

      if (e.key === "," && this.tempSkill) {

        if (!this.skills.includes(this.tempSkill)) {

          this.skills.push(this.tempSkill);

        }

        this.tempSkill = "";

      }

    },

    deleteSkill(skill) {

      this.skills = this.skills.filter((item) => {

        return skill !== item;

      });

    },

    handleSubmit() {

      console.log("HandleSubmit");

      // Validate password

      this.passwordError =

        this.password.length > 5

          ? ""

          : "Password must be at least 6 chars long";

    },

  },

};

</script>

<style>

form {

  max-width: 420px;

  margin: 30px auto;

  background: white;

  text-align: left;

  padding: 40px;

  border-radius: 1epx;

}

label {

  color: #aaa;

  display: inline-block;

  margin: 25px 0 15px;

  font-size: 0.6em;

  text-transform: uppercase;

  letter-spacing: 1px;

  font-weight: bold;

}

input,

select {

  display: block;

  padding: 10px 6px;

  width: 100%;

  box-sizing: border-box;

  border: none;

  border-bottom: 1px solid #ddd;

  color: #555;

}

input[type="checkbox"] {

  display: inline-block;

  width: 16px;

  margin: 0 10px 0 0;

  position: relative;

  top: 2px;

}

.pill {

  display: inline-block;

  margin: 26px 18px 0 0;

  padding: 6px 12px;

  background: #eee;

  border-radius: 20px;

  font-size: 12px;

  letter-spacing: 1px;

  font-weight: bold;

  color: #777;

  cursor: pointer;

}

button {

  background: #0b6dff;

  border: 0;

  padding: 10px 20px;

  margin-top: 20px;

  color: white;

  border-radius: 20px;

}

.submit {

  text-align: center;

}

.error {

  color: #ff0062;

  margin-top: 10px;

  font-size: 0.8em;

  font-weight: bold;

}

</style>