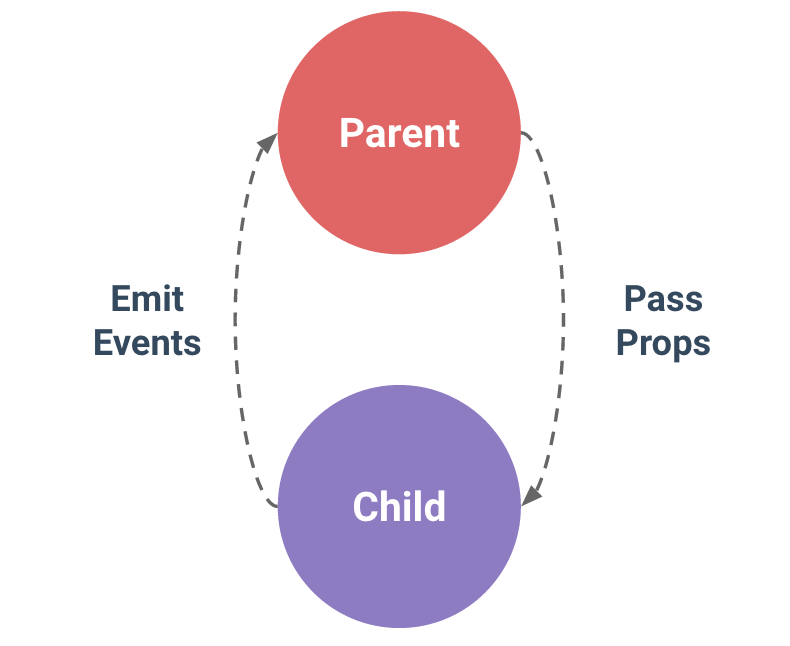
Atelier 4 : Les props

« props » est un mot-clé spécial qui signifie propriétés. Il peut être enregistré sur un composant pour transmettre des données d'un composant parent à l'un de ses composants enfants. Les « props » sont des propriétés pour lesquels le composant attend des valeurs.

Les « props » sont en lecture seule et ne peuvent pas être modifiés par le composant enfant car le composant parent "possède" cette valeur.

Les composants parents transmettent des accessoires au(x) composant(s) enfant tandis que le composant enfant émet des événements au(x) composant(s) parent(s).

Le composant enfant émet également des événements au(x) composant(s) parent(s) au cas où vous auriez besoin d'envoyer des données/événements de l'enfant au composant parent.



1. Créer le composant :

FiltreView.vue sous components/filtre (on créera au préalable ce dossier)

1. Ajouter le tableau intitulé « titles » dans « FiltreView.vue » puis le passer comme étant un props vers le composant « SearchForm.vue » qu’on créera sous /components/filtre par la suite. L'objectif principal de l'utilisation de props est de transmettre des données.



<template>

    <SearchForm :titles="titles" />

  </template>

  <script setup>

import { ref } from 'vue';

  import SearchForm from './SearchForm.vue';

  const titles = ref([

    { id: 1, name: "Pc Montage - Core 2 DUO - 4Go - 320Go" },

    { id: 2, name: "DELL PC Inspiron 5570" },

    { id: 3, name: "Hp PC Portable - 13.3 - Intel Core i5-7200U" },

    { id: 4, name: "Lenovo PC Portable IP330/4000 DualCore/4Go/1To" }

  ]);

  </script>

1. Créer le composant appelé « SearchForm.vue » sous le dossier appelé « components /filtre » avec la donnée « query », la portée de data biding avec le champ de formulaire de recherche.

Ce composant récupère le props  « titles » à partir du composant père « FiltreView.vue » puis l’envoie avec « query » vers le composant « SearchResults.vue » en tant que props.



<template>

    <div>

      <h2>Filter Articles</h2>

      <input

        type="text"

        placeholder="Filter Search"

        v-model="query"

      />

      <br>

      <search-results :query="query" :titles="titles" />

    </div>

  </template>

  <script setup>

  import { ref } from 'vue';

  import SearchResults from './SearchResults.vue';

  const query = ref("");

  const props = defineProps({

    titles:Array

})

  </script>

1. Créer le composant appelé « SearchResults.vue » sous le dossier appelé « components/filtre ». Ce composant va récupérer « titles » et « query » envoyés comme props par le parent « SearchForm.vue ». On va avoir besoin de computed pour :
   * Afficher le contenu du tableau « titles ».
   * Réaliser une méthode qui filtre l’affichage du tableau en fonction de la saisie faite dans le champ de recherche et envoyé dans le props « query ». Pour cette méthode, nous voulons filtrer tous les titres à l'aide du props. Tout est simplement converti en minuscules, nous n'avons donc pas à nous soucier de la casse.

Contrairement aux autres méthodes qui vont se ré-exécuter à chaque rendu de votre application, les méthodes « computed » vont être ré-exécutées seulement si les variables qu’elles contiennent sont modifiées. « computed » ne sera réévaluée que lorsque certaines de ses dépendances seront changées.



<template>

    <div>

      <p>Showing {{ filteredTitles.length }} results - {{ query }}</p>

      <ul>

        <li

          v-for="title in filteredTitles ? filteredTitles : titlesTab"

          :key="title.id">

          {{ title.name }}

        </li>

      </ul>

    </div>

  </template>

  <script setup>

  import { computed } from 'vue';

  const props = defineProps({

    query: String,

    titles:Array

})

  const filteredTitles = computed(() => {

    return props.titles.filter((s) =>

      s.name.toLowerCase().includes(props.query.toLowerCase())

    );

  });

  const titlesTab = computed(() => props.titles);

  </script>

1. Ajouter la route dans le fichier routes.js



import FiltreView from './components/filtre/filtreView.vue';

….

  {

        name:'filtreView',

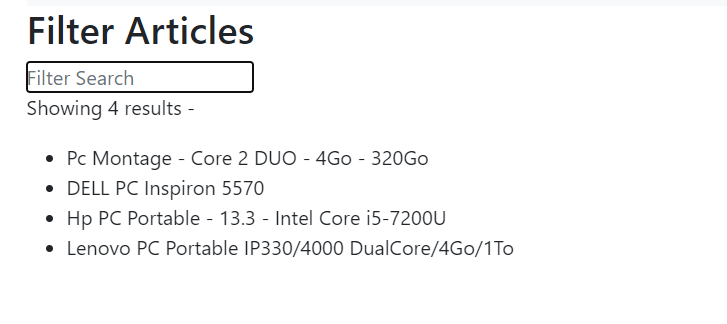
        path:'/filtreView',

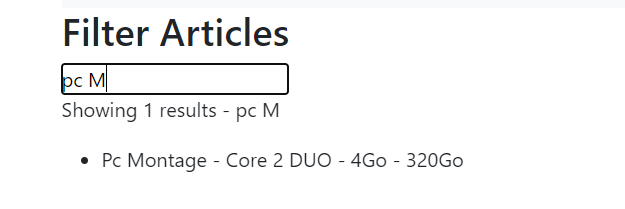
        component:FiltreView

    },

1. Ajouter le lien dans le menu dans App.vue

<router-link to="/filtreView" class="nav-item nav-link">Recherche</router-link>





1. Créer un composant appelé « AddTabForm.vue » sous le dossier appelé « components/filtre ». Ce composant va créer un champ texte qui permet de saisir un nouvel article. Dans le but d’alimenter le tableau « titles » dans « FiltreView.vue ».

Nous avons besoin de passer des paramètres du fils (AddTabForm.vue ) vers le parent (FiltreView.vue).

Lorsque ces actions se produisent, des événements sont émis afin que nous puissions y répondre en utilisant des techniques comme les gestionnaires d'événements. Un objet événement (event-object) est alors automatiquement passé aux événements. Il contiendra des fonctionnalités supplémentaires ou des informations sur l'événement. Et ça sera réalisé avec la fonctionnalité intégrée « emit » de Vue.

La fonction « emit » prend deux paramètres :

* Le nom de l'événement.
* Un payload optionnel que vous pouvez utiliser pour passer des données à l'écouteur de l'événement.



<template>

    <div>

      <h2>Add Articles</h2>

      <input

        type="text"

        placeholder="Article name"

        v-model="nom"

      />

      <button @click.prevent="addName" class="btn btn-primary">Add</button>

    </div>

  </template>

  <script setup>

  import { defineProps, defineEmits } from 'vue';

  const props = defineProps({

    nom: String

})

  const emit = defineEmits(['custom-event-name']);

  const addName = () => {

    emit('custom-event-name', { message: props.nom});

  };

  </script>

1. Modifier « FiltreView.vue » pour récupérer la valeur du champ saisi dans « AddTabForm.vue » et l’ajouter au tableau « titles ».

Une fois qu'un événement est émis, le composant parent qui utilise le composant enfant est capable d'écouter l'événement. Pour cela, vous devez :

* Ajouter un écouteur d'événement avec le nom de l'événement personnalisé sur le composant émettant l'événement.
* Lui assigner une méthode recevant le payload comme premier argument s'il existe.

La méthode Array.from(myArray).pop() permet d’éviter de supprimer le dernier élément du tableau original lorsqu’on veut récupérer la valeur du dernier « id » de « titles ».



<template>

   <AddTabFormVue @custom-event-name="setMessage" />

    <SearchForm :titles="titles" />

  </template>

  <script setup>

  import { ref } from 'vue';

  import SearchForm from './SearchForm.vue';

  import AddTabFormVue from './AddTabForm.vue'

  const titles = ref([

    { id: 1, name: "Pc Montage - Core 2 DUO - 4Go - 320Go" },

    { id: 2, name: "DELL PC Inspiron 5570" },

    { id: 3, name: "Hp PC Portable - 13.3 - Intel Core i5-7200U" },

    { id: 4, name: "Lenovo PC Portable IP330/4000 DualCore/4Go/1To" }

  ]);

  const message=ref('');

          // Définit la méthode utilisée par le payload pour mettre à jour titles

    const setMessage=(payload)=> {

            message.value = payload.message;

            const idnum=Array.from(titles.value).pop().id+1;

            titles.value.push({id:idnum,name:message.value});

          }

  </script>

