

Vue Gadget Shop - step7-challenge


Objectif

Dans cette étape, vous allez apprendre à **organiser votre code avec des composants Vue** en utilisant `app.component()`. L'objectif est de rendre le code plus modulaire et réutilisable.

Vous devez :

- ☒ Créer un **composant GadgetDisplay** pour afficher un gadget.
- ☒ Modifier `index.html` pour utiliser ce composant.
- ☒ Mettre à jour `main.js` pour enregistrer et gérer le composant.

Comprendre un Composant Vue.js

 Qu'est-ce qu'un Composant Vue.js ?

Un **composant** est un bloc réutilisable de code Vue.js qui encapsule **le HTML, le CSS et la logique JavaScript** associée. Il permet de **modulariser** une application et d'éviter la répétition du code.

Structure d'un Composant Vue.js

Un composant Vue.js est défini avec `app.component("nom-du-composant", {...})`. Il contient généralement :

- **Un template** : la structure HTML du composant.
- **Des props** : les données passées au composant.
- **Des méthodes** : les fonctions propres au composant.

Exemple d'un composant `gadget-display` :

```
app.component("gadget-display", {
  props: ["gadget"], // Le gadget est passé en tant que prop depuis le parent
  template: /*html*/ `
    <div class="gadget">
      
      <h2>{{ gadget.name }}</h2>
      <p>Prix : {{ gadget.price }}€</p>
      <p :class="{ 'in-stock': gadget.inStock, 'out-of-stock':
!gadget.inStock }">
        {{ gadget.inStock ? '✅ En stock' : '❌ En rupture de stock' }}
      </p>
      <button @click="addToCart" :disabled="!gadget.inStock">Ajouter au
panier</button>
    </div>
  `,
  methods: {
```

```

    addToCart() {
      console.log(
        "Nous allons modifier le corps de cette méthode un peu plus tard dans
        cette étape !"
      );
    },
  },
});

```

📝 Décomposition du Code

1 Déclaration des **props**

```

props: ["gadget"],

```

- **props** permet de **recevoir des données** depuis un parent.
- Ici, **gadget** est un **objet** contenant des informations comme **name**, **price**, **image**, et **inStock**.

2 Le **template** : Structure HTML du Composant

Nous avons simplement reporter le code Vuejs correspondant au template du gadget.

```


<h2>{{ gadget.name }}</h2>
<p>Prix : {{ gadget.price }}€</p>
<p :class="{ 'in-stock': gadget.inStock, 'out-of-stock': !gadget.inStock}">
  {{ gadget.inStock ? '✅ En stock' : '❌ En rupture de stock' }}
</p>

```

3 Ajout d'une Méthode

```

methods: {
  addToCart() {
    console.log("Nous allons modifier le corps de cette méthode un peu plus tard
    dans cette étape !");
  }
}

```

methods permet de définir des **fonctions internes** au composant.

Nous reviendrons un peu plus tard sur le corps de cette méthode.

Maintenant que vous en savez un peu plus sur les composants, nous allons utiliser ce composant **gadget-display**.

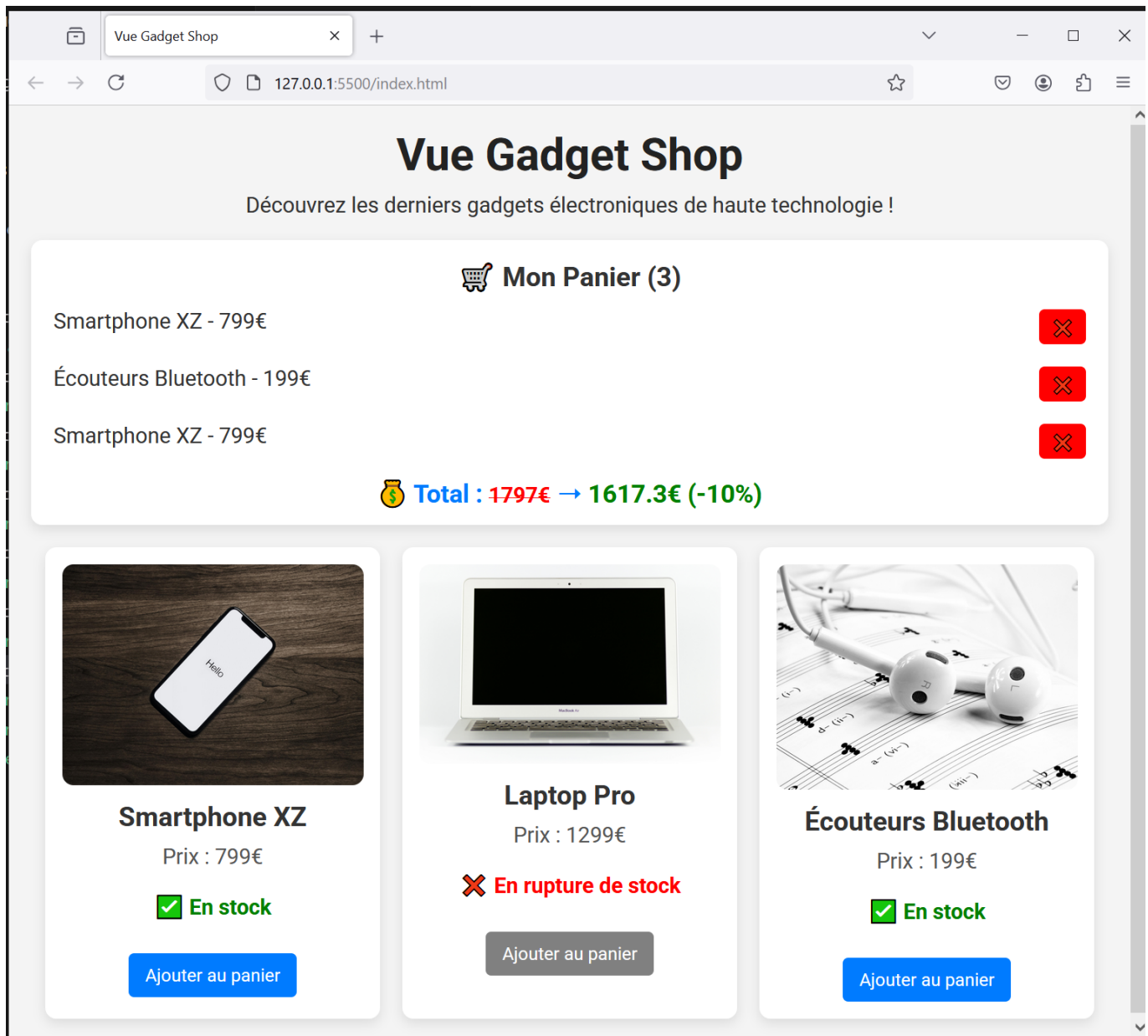
Structure du projet

Votre projet doit contenir la structure suivante :

```
/vue-gadget-shop
|— index.html
|— main.js
|— style.css
|— /components
|   |— GadgetDisplay.js
|— /assets
|   |— phone.jpg
|   |— laptop.jpg
|   |— earbuds.jpg
|
```

Challenge

Vous devez arriver au résultat suivant :



Le résultat n'a pas changé mais après la création et l'utilisation de ce nouveau composant, l'application devra fonctionner comme avant.

🔗 À faire :

- ✓ Créer le fichier `components/GadgetDisplay.js`
- ✓ Mettre à jour `index.html` pour utiliser le composant

- Remplacer le rendu des gadgets par `<gadget-display>`.
- Passer les données avec `:gadget="gadget"`.

Voilà le code modifié :

```
<div class="gadget-container">
  <gadget-display :gadget="gadget" v-for="gadget in gadgets"></gadget-display>
</div>
```

Ce qui est très important à comprendre ici est la communication entre le composant parent à savoir le composant `app` créé grâce à `createApp()` et le composant enfant `gadget-display`.

Grâce au `props` défini dans l'enfant `gadget-display`, il nous suffit de définir `:gadget="gadget"` pour transmettre une information du parent à l'enfant.

En revanche, nous verrons un peu plus bas, que la communication du composant enfant vers le composant parent est un peu plus complexe à mettre en place.

Pour terminer avec la mise à jour de `index.html` vous devez ajouter

```
<script src="./main.js"></script>

<script src="./components/GadgetDisplay.js"></script>

<!-- Mount App -->
<script>
  const mountedApp = app.mount("#app");
</script>
```

et dans `main.js` surppimer `app.mount("#app");`

A ce stade, les gadgets de votre application devraient s'afficher normalement mais le bouton `Ajouter au panier` ne fonctionne pas.

✓ Communication du composant enfant vers le composant parent

Notre objectif est d'offrir à notre composant la possibilité de transmettre à ses parents un événement qui s'est produit en son sein.

Quand nous avons refactorisé le code, nous avons déplacé le code lié au gadget dans le nouveau composant `gadget-display`. En faisant cela, nous avons cassé le fonctionnement du bouton `Ajouter au panier`.

Pourquoi ? Car `cart` se trouve à l'intérieur de l'application racine Vue dans `main.js` et qu'il est donc situé en dehors du « scope » du composant `gadget-display`.

Nous devons donner au composant `gadget-display` un moyen d'annoncer que son bouton est cliqué. Comment faire en sorte que cela se produise ?

La réponse est d'émettre un événement en prévenant le composant parent que cela s'est produit. Ajoutons cela dans notre composant `gadget-display` en modifiant la méthode `addToCart()`.

```
methods: {
  addToCart() {
    this.$emit("add-to-cart", this.gadget); // Émet un événement pour ajouter au panier
  }
}
```

Nous allons grâce à `this.$emit()` émettre un événement appelé `add-to-cart`. Donc, quand le bouton est cliqué, nous allons émettre cet événement.

Ensuite, nous pouvons écouter cet événement à partir du « scope » parent, où nous utilisons `gadget-display`, en ajoutant un listener : `@add-to-cart`.

```
<div class="gadget-container">
  <gadget-display
    v-for="gadget in gadgets"
    :key="gadget.id"
    :gadget="gadget"
    @add-to-cart="addToCart"
  >
</gadget-display>
</div>
```

Lorsque cet événement est « entendu » par le parent, il déclenche une nouvelle méthode ayant pour nom `addToCart`, que nous ajouterons dans `main.js`.

```
methods: {
  addToCart(gadget) {
    this.cart.push(gadget); // Ajoute l'article au panier
  },
  ...
}
```

Si nous vérifions cela dans le navigateur, nous devrions maintenant pouvoir cliquer sur le bouton « Ajouter au panier », ce qui permet aux parents de savoir que l'événement `add-to-cart` s'est produit, ceci déclenchant la méthode `addToCart()`.

💡 Une fois terminé, **comparez votre solution avec `step7-solution.md`** pour voir si vous avez tout bien implémenté ! 🚀