Projet Boutique - partie 1

Ibrahim ALAME

14/02/2023

1 Initialisation du projet

1.1 Initialisation du projet

Placez-vous où vous voulez pour créer le nouveau projet en ouvrant un terminal. Par exemple dans HOME :

```
cd
```

Créez un nouveau projet Vue.js :

```
npm init vue@latest
```

1. Par défaut, le nom est prérempli avec vue-project mais vous pouvez bien sûr le changer par exemple par boutique.
2. La deuxième question est sur l'utilisation de TypeScript :

   ```
   Add TypeScript? ... No / Yes
   ```

   Comme nous l'avons vu, choisissez oui.
3. Ensuite répondez non pour JSX. Nous n'utiliserons par JSX qui est un langage de template React.
4. Répondez non pour Vue Router, Pinia, Vitest et Cypress car nous les verrons plus tard dans la formation.
5. Répondez oui à ESLint, qui permet de contrôler la qualité du code et répondez oui à Prettier pour le formatage du code.

Vous devez en être là :

```
17:32:32 ✔ erwan:~/code$ npm init vue@latest
Need to install the following packages:
  create-vue@latest
Ok to proceed? (y) yes

Vue.js - The Progressive JavaScript Framework

✔ Project name: … dymaproject
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add Cypress for both Unit and End-to-End testing? … No / Yes
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in /home/erwan/code/dymaproject...

Done. Now run:
```

Allez dans le dossier :

```
cd boutique
```

Bien sûr adaptez boutique avec le nom que vous avez donné au projet.

Installez les dépendances :

```
npm install
```

Ouvrez VS Code et chargez le projet :

```
code .
```

1.2 Inclusion d'une police

Nous modifions index.html pour charger une police depuis Google API :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite App</title>
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
      href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap"
      rel="stylesheet"
    />
```

```
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.ts"></script>
  </body>
</html>
```

1.3 Modification de App.vue

Nous enlevons tout le code par défaut et ajoutons l'utilisation de scss :

```
<script setup lang="ts"></script>

<template>
  <h1>Bonjour le monde !</h1>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
</style>
```

1.4 Installation de Sass

Installez Sass en dépendance de développement :

```
npm i -D sass
```

1.5 Modification de assets/base.css

Renommez le fichier base.css en base.scss car nous utilisons Sass et mettez pour le moment :

```
:root {
  --font-family: 'Roboto', sans-serif;
}

body {
  font-family: var(--font-family);
}
```

1.6 Installation de l'extension pour navigateur Vue

Installez l'extension Chrome pour Vue.js ou l'extension Firefox suivant votre navigateur.
Le nom de l'extension est Vue.js devtools.

1.7 Extensions pour Visual Studio Code

Vérifiez dans extensions sur VS Code que vous avez bien installé Volar et Volar TypeScript.
Vérifiez que l'extension @builtin typescript est bien désactivée pour le workspace.

2 Mise en place du style

2.1 Modification deassets/base.scss

Utilisez le style de base suivant :

```
:root {
  --primary-1: #3498db;
  --primary-2: #2980b9;
  --danger-1: #e74c3c;
  --danger-2: #c0392b;
  --success-1: #2ecc71;
  --success-2: #27ae60;
  --gray-1: #f6f6f6;
  --gray-2: #ddd;
  --gray-3: #34495e;
  --text-color: #444;
  --text-primary-color: #ffffff;

  --border: 1px solid var(--gray-2);
  --border-radius: 4px;

  --font-family: 'Roboto', sans-serif;
}
// reset
* {
  box-sizing: border-box;
}
h1,h2,h3,h4 {
  margin: 0;
}
ul {
  list-style: none;
  padding: 0;
}
img {
  max-width: 100%;
}
```

```
a {
  color: var(--text-color);
  text-decoration: none;
}
body {
  min-height: 100vh;
  padding: 0;
  margin: 0;
  font-family: var(--font-family);
  color: var(--text-color);
  background-color: var(--gray-1);
}
// flex
.d-flex {  display: flex; }
.flex-row {  flex-direction: row; }
.flex-column { flex-direction: column; }
.justify-content-center {  justify-content: center; }
.align-items-center {  align-items: center; }
.flex-fill {  flex: 1 1 auto; }
// padding
.p-10 {  padding: 10px; }
.p-20 {  padding: 20px; }
.p-30 {  padding: 30px; }
// margin
.m-10 {  margin: 10px; }
.m-20 {  margin: 20px; }
.m-30 {  margin: 30px; }
```

Si vous ne maîtrisez pas certaines propriétés, n'hésitez pas à revoir les chapitres correspondants dans la formation HTML & CSS.

2.2 Création du fichier assets/debug.scss

Créez les classes pour le débug plus facile du CSS :

```
.b1 {  background-color: red; }
.b2 {  background-color: blue; }
.b3 {  background-color: yellow; }
.b4 {  background-color: green; }
.b5 {  background-color: purple; }
```

Photocopieuse

2.3 Modification de App.vue

N'oubliez pas de modifier App.vue pour l'importer :

```
<script setup lang="ts"></script>

<template>
  <h1>Bonjour le monde !</h1>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;
</style>
```

3 Mise en page globale

3.1 Création de l'architecture

Créez un dossier components dans le dossier src.

Dans ce dossier, créez les fichiers Footer.vue, Header.vue, Shop.vue et Cart.vue.

3.1.1 Composant Footer.vue

Mettez dans le composant :

```
<script setup lang="ts"></script>

<template>
  <footer>
    <h1>Footer</h1>
  </footer>
</template>

<style lang="scss" scoped></style>
```

3.1.2 Composant Header.vue

Mettez dans le composant :

```
<script setup lang="ts"></script>

<template>
  <header>
    <h1>Header</h1>
  </header>
</template>
```

```
<style lang="scss" scoped></style>
```

3.1.3 Composant Shop.vue

Mettez dans le composant :

```
<script setup lang="ts"></script>

<template>
  <div>
    <h1>Shop</h1>
  </div>
</template>

<style lang="scss" scoped></style>
```

3.1.4 Composant Cart.vue

Mettez dans le composant :

```
<script setup lang="ts"></script>

<template>
  <div>
    <h1>Cart</h1>
  </div>
</template>

<style lang="scss" scoped></style>
```

Notez que nous utilisons scoped pour limiter la portée des styles déclarés dans ces composants.

3.1.5 Modification de App.vue

Modifications de notre composant racine pour importer et utiliser les composants que nous avons créés :

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop.vue';
import Cart from './components/Cart.vue';
</script>

<template>
  <div class="app-container">
    <TheHeader class="header b1" />
```

```
    <Shop class="shop b2" />
    <Cart class="cart b3" />
    <TheFooter class="footer b4" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
}
.header {  grid-area: header; }
.shop {  grid-area: shop; }
.cart {  grid-area: cart; }
.footer {  grid-area: footer; }
</style>
```

Nous utilisons une grille CSS, n'hésitez pas à revoir le chapitre sur les grilles dans la formation CSS.

4 Mise en page boutique et panier

4.1 Architecture

Dans le dossier components créez un dossier Cart et un dossier Shop.

Déplacez le composant Cart.vue dans Cart et Shop.vue dans Shop.

Commencez à jour les chemins d'imports dans App.vue :

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
</script>
```

Dans le dossier Cart, créez les composants CartProductList.vue et CartProduct.vue.

Même choisi dans le dossier Shop, créé deux composants ShopProductList.vue et ShopProduct.vue.

4.2 Modification de Shop.vue

Nous allons importer et utiliser les nouveaux composants relatifs à la liste des produits :

```vue
<script setup lang="ts">
import ShopProductList from './ShopProductList.vue';
</script>

<template>
  <div>
    <ShopProductList />
  </div>
</template>

<style lang="scss" scoped></style>
```

Dans ce composant nous plaçons le composant qui va être responsable de gérer la liste des produits.

4.3 Modification de ShopProductList.vue

Dans ce composant nous allons utiliser plusieurs instances de notre composant ShopProduct et les disposer en utilisant une grille CSS :

```vue
<script setup lang="ts">
import ShopProduct from './ShopProduct.vue';
</script>

<template>
  <div class="grid p-20">
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
    <ShopProduct />
  </div>
</template>

<style lang="scss" scoped>
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-auto-rows: 300px;
  gap: 20px;
}
</style>
```

4.4 Modification de ShopProduct.vue

Dans ce composant nous affichons juste un titre pour le moment :

```
<script setup lang="ts"></script>

<template>
  <div class="b5">
    <h1>Shop Product</h1>
  </div>
</template>

<style lang="scss" scoped></style>
```

4.5 Modification de Cart.vue

Dans ce composant nous utilisons le composant CartProductList responsable de gérer l'affichage des produits du panier :

```
<script setup lang="ts">
import CartProductList from './CartProductList.vue';
</script>

<template>
  <div class="p-20">
    <h2 class="mb-10">Panier</h2>
    <CartProductList />
  </div>
</template>

<style lang="scss" scoped></style>
```

N'oubliez pas de modifier assets/base.scss pour ajouter la classe mb-10 :

```
.mb-10 {
  margin-bottom: 10px;
}
```

4.6 Modification de CartProductList.vue

Dans ce composant nous allons utiliser plusieurs instances du composant CartProduct qui sont les produits dans le panier.

Nous utilisons des boîtes flexibles pour positionner les produits.

```
<script setup lang="ts">
import CartProduct from './CartProduct.vue';
</script>
```

```vue
<template>
  <div class="d-flex flex-column">
    <CartProduct />
    <CartProduct />
    <CartProduct />
    <CartProduct />
    <CartProduct />
    <CartProduct />
  </div>
</template>

<style lang="scss" scoped></style>
```

4.7 Modification de CartProduct.vue

Dans ce composant nous affichons pour le moment simplement un titre :

```vue
<script setup lang="ts"></script>

<template>
  <div class="mb-10 b5">
    <h1>Product</h1>
  </div>
</template>

<style lang="scss" scoped></style>
```

5 Mise en page du header et du footer

5.1 Modification deassets/base.scss

Ajoutez les classes utilitaires dont nous aurons besoin :

```scss
.px-20 {
  padding-left: 20px;
  padding-right: 20px;
}
.p-30 { padding: 30px; }
// margin
.m-10 { margin: 10px; }
.mb-10 { margin-bottom: 10px; }
.mr-10 { margin-right: 10px; }
.mr-20 {  margin-right: 20px; }
```

5.2 Modification de Header.vue

Nous mettons en place notre header :

```
<script setup lang="ts"></script>

<template>
  <header class="px-20 d-flex flex-row align-items-center">
    <a href="#" class="d-flex flex-row align-items-center mr-20">
      <img src="../assets/logo.svg" />
      <span class="logo">Dyma</span>
    </a>
    <ul class="d-flex flex-row flex-fill">
      <li class="mr-10">
        <a href="#">Boutique</a>
      </li>
      <li>
        <a href="#">Admin</a>
      </li>
    </ul>
    <ul class="d-flex flex-row">
      <li class="mr-10">
        <a href="#">Inscription</a>
      </li>
      <li>
        <a href="#">Connexion</a>
      </li>
    </ul>
  </header>
</template>

<style lang="scss" scoped>
header {
  background-color: var(--primary-1);
  a {
    color: var(--text-primary-color);
    img {
      width: 20px;
      margin-right: 5px;
    }
    .logo {
      font-weight: 700;
      font-size: 20px;
    }
  }
}
</style>
```

5.3 Modification de Footer.vue

Nous mettons en place notre footer :

```
<script setup lang="ts"></script>

<template>
  <footer class="d-flex flex-row justify-content-center align-items-center">
    <p>Copyright © 2014-2022 Dyma</p>
  </footer>
</template>

<style lang="scss" scoped>
footer {
  background-color: var(--gray-3);
  color: var(--text-primary-color);
}
</style>
```

6 Mise en page de la partie Shop

6.1 Modification de ShopProduct.vue

Nous mettons pour le moment les détails du produit en dur :

```
<script setup lang="ts"></script>

<template>
  <div class="product d-flex flex-column">
    <div class="product-image"></div>
    <div class="p-10 d-flex flex-column">
      <h4>Macbook Pro</h4>
      <p>Performances exceptionnelles avec la puce M1 Pro ou M1</p>
      <div class="d-flex flex-row align-items-center">
        <strong class="flex-fill">Prix : 1500€</strong>
        <button class="btn btn-primary">Ajouter au panier</button>
      </div>
    </div>
  </div>
</template>

<style lang="scss" scoped>
.product {
  background-color: #ffffff;
  border: var(--border);
  border-radius: var(--border-radius);
  &-image {
    border-top-right-radius: var(--border-radius);
```

```
    border-top-left-radius: var(--border-radius);
    background-image: url('https://media.ldlc.com/r1600/ld/products/00/05/82/01/LD0005820198_1.jpg');
    background-size: cover;
    background-position: center;
    height: 250px;
  }
}
</style>
```

Photocopieuse

6.2 Classes pour les boutons

Modifiez assets/base.scss pour ajouter les classes pour nos boutons :

```
// buttons

.btn {
  padding: 8px 15px;
  border: 0;
  border-radius: var(--border-radius);
  cursor: pointer;
  font-weight: 500;
  transition: background-color 0.2s;
}

.btn-primary {
  background-color: var(--primary-1);
  color: var(--text-primary-color);
  &:hover {
    background-color: var(--primary-2);
  }
}

.btn-danger {
  background-color: var(--danger-1);
  color: var(--text-primary-color);
  &:hover {
    background-color: var(--danger-2);
  }
}
```

7 Mise en place du panier

7.1 Modification de CartProduct.vue

Nous mettons en place en dur le produit du panier pour terminer la mise en page :

```vue
<script setup lang="ts"></script>

<template>
  <div class="mb-10 p-10 d-flex flex-row align-items-center product">
    <strong class="flex-fill mr-10">Macbook Pro</strong>
    <span class="mr-10">Prix : 1500€</span>
    <button class="btn btn-danger">Supprimer</button>
  </div>
</template>

<style lang="scss" scoped>
.product {
  border: var(--border);
  border-radius: var(--border-radius);
  background-color: var(--gray-1);
}
</style>
```

7.2 Modification de App.vue

Nous modifions légèrement la classe cart :

```vue
<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
}
.header {
  grid-area: header;
}
.shop {
  grid-area: shop;
}
.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}
.footer {
  grid-area: footer;
}
</style>
```

8 Utilisation des props pour les produits

8.1 Création des interfaces

Créez un dossier interfaces dans src.

Créer dans ce dossier un fichier Product.interface.ts :

```
export interface ProductInterface {
    title: string;
    image: string;
    price: number;
    description: string;
}
```

8.2 Données en dur

En attendant d'avancer plus dans le projet et d'utiliser des requêtes HTTP pour récupérer les produits, nous allons mettre les données en dur.

Pour ce faire, créez un dossier data dans src.

Dans ce dossier, créez un fichier product.ts :

```
export default [
  {
    id: 1,
    image: 'src/assets/images/macbookpro.PNG',
    title: 'Macbook Pro',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 1500,
  },
  {
    id: 2,
    image: 'src/assets/images/levono.PNG',
    title: 'Levono Pro',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 2300,
  },
  {
    id: 3,
    image: 'src/assets/images/rider.PNG',
    title: 'Rider',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 1200,
```

```
  },
  {
    id: 4,
    image: 'src/assets/images/ldlc.PNG',
    title: 'LDLC benolo',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 4500,
  },
  {
    id: 5,
    image: 'src/assets/images/asus.PNG',
    title: 'Asus gamer',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 3755,
  },
  {
    id: 6,
    image: 'src/assets/images/rog.PNG',
    title: 'Rog desktop',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 2452,
  },
  {
    id: 7,
    image: 'src/assets/images/msi.PNG',
    title: 'MSI play',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 1478,
  },
  {
    id: 8,
    image: 'src/assets/images/pad.PNG',
    title: 'Think pad',
    description:
      'Lorem ipsum dolor sit amet consectetur adipisicing edolor tempore ipsam cum ipsum reiciendis',
    price: 899,
  },
];
```

8.3 Modification de App.vue

Dans le composant racine, nous allons importer les données et les typer dans une propriété réactive.

Nous allons ensuite les passer à notre composant enfant Shop en utilisant une liaison de données

avec v-bind.

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
import data from './data/product';

import { reactive } from 'vue';
import type { ProductInterface } from './interfaces';
const products = reactive<ProductInterface[]>(data);
</script>

<template>
  <div class="app-container">
    <TheHeader class="header" />
    <Shop :products="products" class="shop" />
    <Cart class="cart" />
    <TheFooter class="footer" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
}
.header {
  grid-area: header;
}
.shop {
  grid-area: shop;
}
.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}
.footer {
  grid-area: footer;
}
</style>
```

`reactive<ProductInterface[]>(data)` : nous passons en type générique un tableau de produits qui doivent respecter l'interface ProductInterface. Nous initialisons la propriété réactive avec nos données contenues dans data.

8.4 Modification de Shop.vue

Dans le composant Shop.vue, nous définissons la prop reçue depuis le composant parent grâce à defineProps, à savoir la liste de tous les produits.

Nous typons la props en passant le type générique { `products: ProductInterface[]` } : cela signifie que nous recevons un objet contenant une propriété products qui a pour valeur un tableau contenant des ProductInterface.

Nous repassons ensuite la prop reçue plus bas dans l'arbre :

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';
import ShopProductList from './ShopProductList.vue';

defineProps<{
  products: ProductInterface[];
}>();
</script>

<template>
  <div>
    <ShopProductList :products="products" />
  </div>
</template>

<style lang="scss" scoped></style>
```

8.5 Modification de ShopProductList.vue

Dans le composant ShopProductList, nous récupérons également la prop que nous typons comme dans le composant parent Shop.

Cette fois, nous utilisons la directive v-for pour boucler sur les produits et nous utilisons une liaison de propriété pour passer le produit à chaque instance du composant ShopProduct.

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';
import ShopProduct from './ShopProduct.vue';
defineProps<{
  products: ProductInterface[];
}>();
</script>

<template>
```

```
  <div class="grid p-20">
    <ShopProduct v-for="product of products" :product="product" />
  </div>
</template>

<style lang="scss" scoped>
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-auto-rows: 400px;
  gap: 20px;
}
</style>
```

8.6 Modification de ShopProduct.vue

Dans le composant ShopProduct, nous récupérons le produit en props et nous utilisons la notation raccourcie de la directive v-text pour afficher les différentes propriétés du produit.

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';
defineProps<{
  product: ProductInterface;
}>();
</script>

<template>
  <div class="product d-flex flex-column">
    <div
      class="product-image"
      :style="{ backgroundImage: `url(${product.image})` }"
    ></div>
    <div class="p-10 d-flex flex-column">
      <h4>{{ product.title }}</h4>
      <p>{{ product.description }}</p>
      <div class="d-flex flex-row align-items-center">
        <strong class="flex-fill">Prix : {{ product.price }}€</strong>
        <button class="btn btn-primary">Ajouter au panier</button>
      </div>
    </div>
  </div>
</template>

<style lang="scss" scoped>
.product {
  background-color: #ffffff;
  border: var(--border);
```

```
  border-radius: var(--border-radius);
  &-image {
    border-top-right-radius: var(--border-radius);
    border-top-left-radius: var(--border-radius);
    background-size: cover;
    background-position: center;
    height: 250px;
  }
}
</style>
```

9 Ajout d'un produit dans le panier

9.1 Logique à mettre en place

De la même manière que nous faisons descendre la liste de produits dans l'arbre des composants dans cet ordre `App` → `Shop` → `ShopProductList` → `ShopProduct`, nous souhaitons maintenant propager de l'information dans ce sens :

`ShopProduct` → `ShopProductList` → `Shop` → `App`.

En effet, nous souhaitons que lorsqu'un utilisateur clique sur le bouton d'ajout au panier dans un composant ShopProduct, que cet événement remonte le long de l'arbre des composants et qu'il puisse nous informer sur quel produit l'utilisateur souhaite ajouter.

Pour ce faire, nous allons utiliser un événement personnalisé, qui contiendra l'id du produit à ajouter au panier. Cet événement sera émis depuis le composant ShopProduct concerné, puis sera réémis par ShopProductList puis par Shop.

Une fois que l'événement est arrivé au composant App, nous pourrons utiliser l'id du produit pour l'ajouter au panier. En effet, dans notre composant App nous centralisons les informations relatives au panier et aux produits.

9.2 Modification de App.vue

Dans le composant racine, nous modifions notre propriété réactive pour ajouter une propriété cart contenant les produits du panier. Nous initialisons le panier avec un tableau vide.

Nous créons une fonction addProductToCart() qui va permettre d'ajouter un produit dans le panier en utilisant son id unique. La logique est simple : nous récupérons le produit à l'aide de son id depuis la propriété réactive state, ensuite nous vérifions qu'il n'est pas déjà dans le panier, et enfin nous l'ajoutons au panier.

Notez bien que nous utilisons { ...product } pour créer une copie de l'objet qui ne soit pas un Proxy vers l'objet du produit contenu dans la propriété réactive.

rappelez-vous en effet que les propriétés réactives contiennent des Proxys vers les objets que nous passons.

De cette manière, nous obtenons un nouvel objet qui a une nouvelle référence et qui est totalement différent de l'objet dans la propriété réactive.

Nous créons également un événement personnalisé @add-product-to-cart qui va déclencher la fonction addProductToCart() lorsqu'il est reçu. Ici, addProductToCart() est donc le gestionnaire pour cet événement.

Notez bien que l'événement personnalisé est en kebab-case.

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
import data from './data/product';

import { reactive } from 'vue';
import type { ProductInterface } from './interfaces';

const state = reactive<{
  products: ProductInterface[];
  cart: ProductInterface[];
}>({
  products: data,
  cart: [],
});

function addProductToCart(productId: number): void {
  const product = state.products.find((product) => product.id === productId);
  if (product && !state.cart.find((product) => product.id === productId)) {
    state.cart.push({ ...product });
  }
}
</script>

<template>
  <div class="app-container">
    <TheHeader class="header" />
    <Shop
      :products="state.products"
      @add-product-to-cart="addProductToCart"
      class="shop"
    />
    <Cart class="cart" />
    <TheFooter class="footer" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;
```

```
.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
}
.header {
  grid-area: header;
}
.shop {
  grid-area: shop;
}
.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}
.footer {
  grid-area: footer;
}
</style>
```

9.3 Modification de Shop.vue

Ici nous ajoutons la directive v-on pour écouter l'événement add-product-to-cart lorsqu'il est émis par le composant ShopProductList

Nous ne faisons que le retransmettre en démontrant un événement avec defineEmits qui est identique à celui reçu.

L'événement qui va être retransmis est addProductToCart (notez bien le camelCase) et il contient un seul argument qui est le productId de type number.

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';
import ShopProductList from './ShopProductList.vue';

defineProps<{
  products: ProductInterface[];
}>();

const emit = defineEmits<{
  (e: 'addProductToCart', productId: number): void;
}>();
</script>

<template>
  <div>
```

```
    <ShopProductList
      @add-product-to-cart="emit('addProductToCart', $event)"
      :products="products"
    />
  </div>
</template>

<style lang="scss" scoped></style>
```

9.4 Modification de ShopProductList.vue

Même logique, nous ne faisons que retransmettre l'événement provenant du composant ShopProduct vers le composant Shop :

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';
import ShopProduct from './ShopProduct.vue';

defineProps<{
  products: ProductInterface[];
}>();

const emit = defineEmits<{
  (e: 'addProductToCart', productId: number): void;
}>();
</script>

<template>
  <div class="grid p-20">
    <ShopProduct
      @add-product-to-cart="emit('addProductToCart', $event)"
      v-for="product of products"
      :product="product"
    />
  </div>
</template>

<style lang="scss" scoped>
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-auto-rows: 400px;
  gap: 20px;
}
</style>
```

9.5 Modification de ShopProduct.vue

La logique est la suivante :

@click : Ici nous créons une liaison d'événement avec v-on sur l'événement click sur le bouton d'ajout au panier.

`const emit = defineEmits< (e: 'addProductToCart', productId: number): void; >();` : Nous créons un événement personnalisé `'addProductToCart'` qui va contenir un seul argument : le productId.

`@click="emit('addProductToCart', product.id)"` : Nous l'émettons avec emit() lors d'un clic sur le bouton. Le premier argument est le nom de l'événement en camelCase et le deuxième argument est l'id du produit. Voici le code du composant :

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';

defineProps<{
  product: ProductInterface;
}>();

const emit = defineEmits<{
  (e: 'addProductToCart', productId: number): void;
}>();
</script>

<template>
  <div class="product d-flex flex-column">
    <div
      class="product-image"
      :style="{ backgroundImage: `url(${product.image})` }"
    ></div>
    <div class="p-10 d-flex flex-column">
      <h4>{{ product.title }}</h4>
      <p>{{ product.description }}</p>
      <div class="d-flex flex-row align-items-center">
        <strong class="flex-fill">Prix : {{ product.price }}€</strong>
        <button
          class="btn btn-primary"
          @click="emit('addProductToCart', product.id)"
        >
          Ajouter au panier
        </button>
      </div>
    </div>
  </div>
</template>

<style lang="scss" scoped>
.product {
  background-color: #ffffff;
```

```
  border: var(--border);
  border-radius: var(--border-radius);
  &-image {
    border-top-right-radius: var(--border-radius);
    border-top-left-radius: var(--border-radius);
    background-size: cover;
    background-position: center;
    height: 250px;
  }
}
</style>
```

10 Affichage et suppression pour la partie panier

10.1 Logique à mettre en place

De la même manière que précédemment pour la partie Shop, nous souhaitons faire descendre l'information du contenu du panier le long de l'arbre des composants de cette manière :

$$\texttt{App} \rightarrow \texttt{Cart} \rightarrow \texttt{CartProductList} \rightarrow \texttt{CartProduct}.$$

De cette manière, nous pourrons afficher le contenu du panier. Nous allons simplement utiliser une prop sur les différents niveaux de composants pour passer cette information.

Nous souhaitons également faire remonter l'information dans le sens inverse lorsque l'utilisateur clique sur le bouton "Supprimer" d'un élément du panier :

$$\texttt{CartProduct} \rightarrow \texttt{CartProductList} \rightarrow \texttt{Cart} \rightarrow \texttt{App}.$$

Pour ce faire, nous allons utiliser un événement personnalisé, qui contiendra l'id du produit à supprimer du panier. Cet événement sera émis depuis le composant CartProduct concerné, puis sera réémis par CartProductList puis par Cart.

Une fois que l'événement sera arrivé au composant App, nous pourrons utiliser l'id du produit pour le supprimer du panier. La logique est donc similaire à celle utilisée pour la partie Shop.

10.2 Modification de App.vue

Dans App.vue :

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
import data from './data/product';

import { reactive } from 'vue';
import type { ProductInterface } from './interfaces';
```

```
const state = reactive<{
  products: ProductInterface[];
  cart: ProductInterface[];
}>({
  products: data,
  cart: [],
});

function addProductToCart(productId: number): void {
  const product = state.products.find((product) => product.id === productId);
  if (product && !state.cart.find((product) => product.id === productId)) {
    state.cart.push({ ...product });
  }
}

function removeProductFromCart(productId: number): void {
  state.cart = state.cart.filter((product) => product.id !== productId);
}
</script>

<template>
  <div class="app-container">
    <TheHeader class="header" />
    <Shop
      :products="state.products"
      @add-product-to-cart="addProductToCart"
      class="shop"
    />
    <Cart
      :cart="state.cart"
      class="cart"
      @remove-product-from-cart="removeProductFromCart"
    />
    <TheFooter class="footer" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
```

```
}
.header {
  grid-area: header;
}
.shop {
  grid-area: shop;
}
.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}
.footer {
  grid-area: footer;
}
</style>
```

Nous définissons une fonction removeProductFromCart() pour supprimer un article du panier lorsque nous recevons l'événement removeProductFromCart.

- `@remove-product-from-cart="removeProductFromCart"` : nous écoutons l'événement remove-product-from-cart et enregistrons la fonction removeProductFromCart() comme gestionnaire d'événement.

  Notez bien que nous n'utilisons pas de parenthèses avec removeProductFromCart. En JavaScript cela signifie que nous passons la fonction avec tous les arguments récupérés.

  Notez également bien que l'événement côté template est en kebab-case (`remove-product-from-cart`) et qu'il soit être en camelCase côté script (`removeProductFromCart`).

- `:cart="state.cart"` : nous passons le contenu du panier avec une props au composant Cart.

10.3 Modification de Cart.vue

Voici le composant Cart.vue :

```
<script setup lang="ts">
import CartProductList from './CartProductList.vue';
import type { ProductInterface } from '@/interfaces';

const props = defineProps<{
  cart: ProductInterface[];
}>();

const emit = defineEmits<{
  (e: 'removeProductFromCart', productId: number): void;
}>();
</script>

<template>
```

```
  <div class="p-20">
    <h2 class="mb-10">Panier</h2>
    <CartProductList
      :cart="cart"
      @remove-product-from-cart="emit('removeProductFromCart', $event)"
    />
  </div>
</template>

<style lang="scss" scoped></style>
```

Il ne fait que transmettre dans un sens une prop et dans l'autre un événement. Vous pouvez le voir comme un "relais" des informations dans les deux sens.

10.4 Modification de CartProductList.vue

Voici le composant :

```
<script setup lang="ts">
import CartProduct from './CartProduct.vue';
import type { ProductInterface } from '@/interfaces';

const props = defineProps<{
  cart: ProductInterface[];
}>();

const emit = defineEmits<{
  (e: 'removeProductFromCart', productId: number): void;
}>();
</script>

<template>
  <div class="d-flex flex-column">
    <CartProduct
      v-for="product of cart"
      :product="product"
      @remove-product-from-cart="emit('removeProductFromCart', $event)"
    />
  </div>
</template>

<style lang="scss" scoped></style>
```

Même chose, ce composant rapporte les informations dans les deux sens à chaque instance du composant CartProduct. Pour rappel, il y a un composant CartProduct pour chaque produit grâce à la directive v-for.

10.5 Modification de CartProduct

Voici le composant :

```
<script setup lang="ts">
import type { ProductInterface } from '@/interfaces';

defineProps<{
  product: ProductInterface;
}>();

const emit = defineEmits<{
  (e: 'removeProductFromCart', productId: number): void;
}>();
</script>

<template>
  <div class="mb-10 p-10 d-flex flex-row align-items-center product">
    <strong class="mr-10">{{ product.title }}</strong>
    <span class="mr-10">Prix : {{ product.price }}€</span>
    <button
      class="btn btn-danger"
      @click="emit('removeProductFromCart', product.id)"
    >
      Supprimer
    </button>
  </div>
</template>

<style lang="scss" scoped>
.product {
  border: var(--border);
  border-radius: var(--border-radius);
  background-color: var(--gray-1);
}
</style>
```

Le composant émet un événement removeProductFromCart lorsque l'utilisateur clique sur le bouton de suppression d'un article du panier.

L'événement contient l'id unique du produit à supprimer.

Il affiche les informations du produit contenues dans la prop reçue.

11 Gestion des quantités

11.1 Création de l'interface ProductCartInterface.ts

Dans le dossier src/interfaces crée le fichier ProductCart.interface.ts :

```ts
import type { ProductInterface } from "./Product.interface";

export interface ProductCartInterface extends ProductInterface {
    quantity: number;
}
```

La nouvelle interface étend simplement l'interface ProductInterface en ajoutant une propriété pour la quantité.

11.2 Création d'un index pour les interfaces

Créer un fichier src/interfaces/index.ts :

```ts
export * from './Product.interface';
export * from './ProductCart.interface';
```

Cela permet simplement de simplifier les importations des interfaces dans nos composants.

11.3 Modification de CartProduct.vue

Nous utilisons notre nouvelle interface et nous affichons la quantité de chaque produit dans le panier :

```vue
<script setup lang="ts">
import type { ProductCartInterface } from '@/interfaces';

defineProps<{
  product: ProductCartInterface;
}>();

const emit = defineEmits<{
  (e: 'removeProductFromCart', productId: number): void;
}>();
</script>

<template>
  <div class="mb-10 p-10 d-flex flex-row align-items-center product">
    <strong class="mr-10">{{ product.title }}</strong>
    <span class="flex-fill mr-10">x {{ product.quantity }}</span>
    <span class="mr-10">Prix : {{ product.price }}€</span>
    <button
      class="btn btn-danger"
      @click="emit('removeProductFromCart', product.id)"
    >
      Supprimer
    </button>
  </div>
</template>
```

```
<style lang="scss" scoped>
.product {
  border: var(--border);
  border-radius: var(--border-radius);
  background-color: var(--gray-1);
}
</style>
```

11.4 Modification de CartProductList.vue et Cart.vue

Utilisez la nouvelle interface :

```
import type { ProductCartInterface } from '@/interfaces';

const props = defineProps<{
  cart: ProductCartInterface[];
}>();
```

11.5 Modification de App.vue

Nous gérons maintenant l'incrémentation ou la décrémentation de la quantité lors de l'ajout ou de la suppression d'un élément du panier.

Voici le composant :

```
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
import data from './data/product';

import { reactive } from 'vue';
import type { ProductInterface } from './interfaces';
import type { ProductCartInterface } from './interfaces';

const state = reactive<{
  products: ProductInterface[];
  cart: ProductCartInterface[];
}>({
  products: data,
  cart: [],
});

function addProductToCart(productId: number): void {
  const product = state.products.find((product) => product.id === productId);
  if (product) {
```

```
    const productInCart = state.cart.find(
      (product) => product.id === productId
    );
    if (productInCart) {
      productInCart.quantity++;
    } else {
      state.cart.push({ ...product, quantity: 1 });
    }
  }
}
function removeProductFromCart(productId: number): void {
  const productFromCart = state.cart.find(
    (product) => product.id === productId
  );
  if (productFromCart?.quantity === 1) {
    state.cart = state.cart.filter((product) => product.id !== productId);
  } else {
    productFromCart.quantity--;
  }
}
</script>

<template>
  <div class="app-container">
    <TheHeader class="header" />
    <Shop
      :products="state.products"
      @add-product-to-cart="addProductToCart"
      class="shop"
    />
    <Cart
      :cart="state.cart"
      class="cart"
      @remove-product-from-cart="removeProductFromCart"
    />
    <TheFooter class="footer" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
```

```
  grid-template-rows: 48px auto 48px;
}
.header {
  grid-area: header;
}
.shop {
  grid-area: shop;
}
.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}
.footer {
  grid-area: footer;
}
</style>
```

12 Affichage du total et affichage initial du panier

12.1 Création dossier pour les styles

Dans le dossier assets, créez un dossier scss et placez-y les fichiers base.scss et debug.scss. N'oubliez pas de modifier en conséquence les importations dans App.vue :

```
<style lang="scss">
@use './assets/scss/base.scss' as *;
@use './assets/scss/debug.scss' as *;
```

12.2 Ajout d'une classe

Dans le fichier base.scss, ajoutez la classe suivante :

```
.btn-success {
    background-color: var(--success-1);
    color: var(--text-primary-color);
    &:hover {
        background-color: var(--success-2);
    }
}
```

12.3 Modification de Cart.vue

Nous utilisons une propriété utilisée pour calculer le total du panier :

```vue
<script setup lang="ts">
import type { ProductCartInterface } from '@/interfaces';
import { computed } from 'vue';
import CartProductList from './CartProductList.vue';
const props = defineProps<{
  cart: ProductCartInterface[];
}>();
const totalPrice = computed(() =>
  props.cart.reduce((acc, product) => acc + product.price * product.quantity, 0)
);
const emit = defineEmits<{
  (e: 'removeProductFromCart', productId: number): void;
}>();
</script>

<template>
  <div class="p-20 d-flex flex-column">
    <h2 class="mb-10">Panier</h2>
    <CartProductList
      class="flex-fill"
      :cart="cart"
      @remove-product-from-cart="emit('removeProductFromCart', $event)"
    />
    <button class="btn btn-success">Commander ({{ totalPrice }}€)</button>
  </div>
</template>

<style lang="scss" scoped></style>
```

12.4 Modification de App.vue

Nous utilisons une propriété exploitée pour calculer pour savoir si le panier est vide ou non. Si le panier est vide, nous n'affichons pas le composant panier grâce à la directive v-if. Nous utilisons également une liaison de classe avec la directive v-bind pour appliquer la classe gridEmpty si le panier est vide. Cette classe permet de modifier simplement la grille CSS.

```vue
<script setup lang="ts">
import TheHeader from './components/Header.vue';
import TheFooter from './components/Footer.vue';
import Shop from './components/Shop/Shop.vue';
import Cart from './components/Cart/Cart.vue';
import data from './data/product';

import { computed, reactive } from 'vue';
import type { ProductInterface } from './interfaces';
import type { ProductCartInterface } from './interfaces';
```

```
const state = reactive<{
  products: ProductInterface[];
  cart: ProductCartInterface[];
}>({
  products: data,
  cart: [],
});

function addProductToCart(productId: number): void {
  const product = state.products.find((product) => product.id === productId);
  if (product) {
    const productInCart = state.cart.find(
      (product) => product.id === productId
    );
    if (productInCart) {
      productInCart.quantity++;
    } else {
      state.cart.push({ ...product, quantity: 1 });
    }
  }
}

function removeProductFromCart(productId: number): void {
  const productFromCart = state.cart.find(
    (product) => product.id === productId
  );
  if (productFromCart?.quantity === 1) {
    state.cart = state.cart.filter((product) => product.id !== productId);
  } else {
    productFromCart.quantity--;
  }
}

const cartEmpty = computed(() => state.cart.length === 0);
</script>

<template>
  <div
    class="app-container"
    :class="{
      gridEmpty: cartEmpty,
    }"
  >
    <TheHeader class="header" />
    <Shop
      :products="state.products"
      @add-product-to-cart="addProductToCart"
      class="shop"
```

```
    />
    <Cart
      v-if="!cartEmpty"
      :cart="state.cart"
      class="cart"
      @remove-product-from-cart="removeProductFromCart"
    />
    <TheFooter class="footer" />
  </div>
</template>

<style lang="scss">
@use './assets/base.scss' as *;
@use './assets/debug.scss' as *;

.app-container {
  min-height: 100vh;
  display: grid;
  grid-template-areas: 'header header' 'shop cart' 'footer footer';
  grid-template-columns: 75% 25%;
  grid-template-rows: 48px auto 48px;
}

.gridEmpty {
  grid-template-areas: 'header' 'shop' 'footer';
  grid-template-columns: 100%;
}

.header {
  grid-area: header;
}

.shop {
  grid-area: shop;
}

.cart {
  grid-area: cart;
  border-left: var(--border);
  background-color: white;
}

.footer {
  grid-area: footer;
}
</style>
```