Projet Javascript - Suricate

L'objectif est de réaliser une application type "boutique en ligne". Il faut gérer la construction du panier du client à partir d'une liste d'articles définies dans un catalogue.

CONTEXTE

Comme l'illustre la figure ci-dessus, la boutique en ligne présente deux zones principales. Celle de gauche (`#boutique`) contient les différents produits mis en vente par la boutique, celle de droite (`#panier`) correspond au panier d'achat.

Le contenu de la boutique est construit automatiquement au chargement de la page par rapport à des données définies dans un *catalogue*. La structure d'un catalogue est définie plus bas. Différents catalogues doivent pouvoir être utilisés sans que cela ne nécessite de modifier le code qui permet le chargement de la boutique (plusieurs fichiers catalogues sont fournis dans ce but).

Dans la boutique, l'utilisateur peut pour un article choisir la quantité qu'il souhaite acheter à l'aide du champ de saisie correspondant. Cette quantité doit toujours être comprise entre 0 et 9 que l'on utilise les flèches ou que l'on saisisse directement une valeur.

Il doit ensuite cliquer sur le bouton représentant un chariot pour mettre dans le panier cet article avec la quantité désirée. Le bouton de mise en panier est inactif tant que la quantité est 0, actif sinon. L'inactivité se traduit visuellement par une opacité de `0.25` du composant et par l'absence de réaction au clic, l'activité par une opacité à 1 et une mise en panier effective de l'article. Après la mise en panier le compteur de quantité côté boutique est remis à 0.

Lorsqu'il est mis en panier un article apparaît dans la zone du panier. Un même article n'apparaît toujours qu'une seule fois dans le panier, avec sa quantité totale commandée. Donc

lorsqu'un article déjà dans le panier est à nouveau commandé, sa quantité est mise à jour dans le panier. Il n'est pas autorisé de commander plus de 9 fois un même article, même en plusieurs fois. Donc la quantité d'un même article dans le panier ne peut jamais dépasser 9. En cliquant sur le bouton représentant une poubelle on vide l'article quelle que soit sa quantité.

Le montant total des articles dans le panier apparaît dans la zone `#total`. Ce montant est mis à jour à chaque modification du panier, ajout ou suppression d'articles.

Au-dessus du total se trouve une zone de texte qui permet de filtrer les articles de la boutique. Seuls les articles dont le nom (champ `name` dans le catalogue) contient la chaîne de caractères présente dans le filtre sont affichés. La saisie s'adapte à chaque caractère frappé par l'utilisateur.

Le travail que vous devez réaliser consiste à écrire le code javascript permettant de mettre en œuvre ce comportement.

Vous devez utiliser les fichiers contenus dans l'archive **projetJS-suricate.zip** fournie. Cette archive contient :

- Le fichier **index.html**, dont vous ne modifierez pas le contenu,
- Le fichier **style/style.css** et le dossier **style/images**, vous pouvez améliorer si vous le souhaitez,
- Le fichier **scripts/project.js**. Il contient déjà du code qui vous est fourni. Vous le compléterez avec le code javascript que vous écrirez. Toutes les fonctions que vous écrirez dans ce fichier devront être accompagnées d'un commentaire.
- Le dossier **data** qui contient les fichiers de données permettant la définition du catalogue regroupant tous les produits de la boutique. Plusieurs fichiers, d'extension `.js` sont fournis. Le fichier de données est chargé dans le header du document html, comme le script.
- Le dossier **images** contenant les images des produits proposés.

- Chacun des fichiers de données (fournis dans le dossier **data**) définit une variable globale `catalog` qui contient la liste des produits de la boutique.

  Vous pouvez compléter les exemples en définissant vos propres catalogues de produits.

- Les produits sont représentés sous la forme d'une structure JSON. Chacune de ces données comportent 4 champs : `name`, `description`, `image` et `price`. Si *`oneProduct`* est une variable représentant un tel objet produit, alors la notation pointée permet d'accéder au champ correspondant. On donne ci-dessous, quelques exemples pour rappel :

```
let oneProduct = catalog[0] ;

let txt = oneProduct.description ;

let txt = catalog[0].description ;

let anotherProduct = catalog[2] ;

let productImage = anotherProduct.image ;
```

METHODOLOGIE

1. Étudiez le code html du document **index.html** afin de bien en comprendre la structure et de repérer les différents éléments qui constituent la page, et notamment leurs `id`.

   Ce document utilise pour le moment une version incomplète du fichier **scripts/project.js**. Par exemple, les images des produits ne s'affichent pas encore. C'est normal. C'est votre travail dans ce projet de compléter ce fichier de script.

2. Afin d'en comprendre le fonctionnement, étudiez le code de la fonction *`createShop`* et les fonctions qu'elle utilise :
   `createProduct`,
   `createBlock`,
   `createOrderControlBlock`.
   Comme vous le constatez la fonction *`createFigureBlock`* n'est pas implémentée correctement. Écrivez un code correct pour cette fonction qui doit créer l'élément `figure` inclus dans l'élément `div.produit`, permettant d'afficher les images produit.

3. Le code fourni permet la création des éléments `div.controle`, en particulier grâce à la fonction *`createOrderControlBlock`*. Cependant ces contrôles sont pour le moment sans effet.

   Complétez donc le code existant pour mettre en place la gestion de la zone de saisie des quantités, avec le contrôle des valeurs : si l'utilisateur saisit "à la main" une valeur interdite, il faut annuler cette saisie, par exemple en la remplaçant par 0.

   Il vous faut aussi gérer l'activation/désactivation du bouton de mise en panier selon que la quantité vaut 0 ou non. Pour l'aspect visuel vous pouvez simplement exploiter différentes valeurs de la propriété CSS *`opacity`*.

4. Gérez l'action déclenchée par le bouton de mise en panier (lorsque celui est actif).

   Il faut définir au préalable une fonction qui construit un élément *`div.achat`* (pensez à la gestion des *`id`* de ces éléments).

   Vous devez également garantir que la valeur de *`#montant`* est mise à jour.

   N'oubliez pas de gérer le cas où le produit ajouté est déjà présent dans le panier.

5. Gérez l'action des boutons de suppression du panier. Vous devez toujours maintenir la cohérence de *`#montant`*.

6. Un événement `keyup` est émis à chaque fois que l'on relâche une touche dans un élément `input`.

   Faites le nécessaire pour que seuls les produits dont la propriété *`name`* contient le texte présent dans *`#filter`* soient affichés dans la boutique. Le filtrage doit être modifié au fur et à mesure de la saisie des caractères dans *`#filter`*.

   Une manière de procéder peut-être de modifier, en fonction de leur nom, la valeur de la propriété *`style.display`* des éléments représentant les produits.

7. Vous pouvez compléter le comportement de la page avec d'autres fonctionnalités. Dans ce cas vous les indiquerez dans le fichier **lisezmoi.txt** fourni avec votre archive. Voici quelques exemples d'extensions envisageables :
   - permettre la modification de la quantité d'un article dans le panier,
   - ajouter un bouton qui permet de stocker localement sur le navigateur le panier en cours pour le retrouver lors du prochain chargement de la page, on peut utiliser la fonctionnalité "Web Storage". Utilisez l'objet *`localStorage`* et ses fonctions *`setItem`* et *`getItem`* pour mémoriser les articles du panier.

8. Rendez votre travail sous la forme d'une archive. Le nom de cette archive sera de la forme ***votre-prénom*_projet_suricate.zip**. Cette archive contiendra un répertoire dont le nom sera ***votre-prénom*_projet_suricate**.
   Les noms des fichiers (y compris leurs extensions) seront en minuscules.