NSI - Projet n°3 – Gérer un budget

L'objectif est de réaliser une application pour apprendre à une classe de CE1 à gérer un budget à partir d'une liste d'articles définie dans un catalogue.

L'application simulant une 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 avec les articles sélectionnés par la classe.

Dans la boutique, pour chaque article, l'utilisateur peut 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. Cette contrainte de validité est garantie lorsque l'on utilise les flèches mais doit être gérée si l'utilisateur saisit directement une valeur. Il doit ensuite cliquer sur le bouton représentant un chariot de course 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 est remis à 0.

Lorsqu'il est mis dans le 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 jamais 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 supprime complètement un article du panier (quelle que soit la 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 produits de la boutique. Seuls les produits dont le nom (champ *name* dans le catalogue, voir ci-dessous) contient la chaîne de caractères présente dans le filtre sont affichés. La saisie s'adapte à chaque caractère frappé par l'utilisateur.

Enfin, un message d'erreur apparaît si l'utilisateur dépasse le budget prévu.

Le travail à réaliser consiste à écrire le code javascript permettant de mettre en œuvre le comportement décrit avant.

Le fichier NSIProjet3.zip contient :

- le fichier NSIProjet3.html à renommer index.html,
- le fichier style/NSIProjet3.css et le dossier style/images,
- le dossier images contenant les images des produits proposés,
- le fichier scripts/NSIProjet3.js. Il contient déjà du code mais il doit encore être complété,
- le dossier data qui contient les fichiers de données permettant la définition du catalogue regroupant tous les produits de la boutique. Le fichier de données est chargé dans le header du document html, comme le script.

Lorsque le projet sera terminé, déposer dans mon casier de l'ENT un fichier zip nommé PR3 suivi du nom du groupe et contenant l'ensemble des dossiers et fichiers.

Remarques

- Chacun des fichiers de données (fournis dans le dossier data) définit une variable globale *catalog* qui contient la liste des produits de la boutique.
- Les produits sont représentés sous la forme d'une structure (un objet javascript). Chacune de ces données comportent 4 champs : *name, description, image* et *price*. On accède aux différents champs d'un objet produit grâce à la notation pointée :

```
var produit = catalog[4];   // l'objet correspondant au produit d'indice 4 est stocké dans produit
var txt = produit.description;     // récupération de la description de produit, stockée dans txt
var produitSrc = produit.image;     // récupération du champ 'image' de produit, stocké dans produitSrc
```

Il est conseillé de traiter les questions progressivement en respectant l'ordre suggéré.

1) Etudier le code html du document NSIProjet3.html afin de bien en comprendre la structure. Ce document utilise pour le moment une version incomplète du fichier scripts/ NSIProjet3.js. En particulier, les images des produits ne s'affichent pas encore.
2) Dans l'archive vous est fourni le fichier NSIProjet3-static.html. Celui-ci est une copie statique (donc sans code javascript actif) du code HTML correspondant au contenu de la page en action. Ce contenu est donc un exemple de résultat produit par les actions du code javascript lorsqu'il est actif.
   Etudier ce code html afin de comprendre ce que doit produire l'exécution de votre code javascript. En particulier, étudier le code correspondant à chaque produit dans *#boutique* et chaque achat dans *#panier*. Vous analyserez en particulier les id qui apparaissent dans les différents éléments. Ils sont de la forme *i-text* et vous ferez le lien entre ce *i* et l'indice de l'article dans le catalogue. Vous remarquerez également l'égalité de cette valeur i lorsque le produit et l'achat correspondent (étudiez par exemple l'élément *#0-product* et *#0-achat*).
3) Afin d'en comprendre le fonctionnement, étudier attentivement le code de la fonction *createShop* et les fonctions qu'elle utilise : *createProduct, createBlock, createOrderControlBlock*.
   La fonction *createFigureBlock* n'est pas implémentée correctement. Ecrire un code correct pour cette fonction qui doit créer l'élément figure inclus dans l'élément *div.produit*.
4) 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 le *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*.
   Etudiez la documentation de la fonction *indexOf* des chaînes de caractères.
5) 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. Compléter 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, il s'agit d'exploiter différentes valeurs de la propriété CSS *opacity*.
6) Gérer l'action déclenchée par le bouton de mise en panier (lorsque celui est actif).
   Il sera certainement pertinent de définir au préalable une fonction qui construit un élément *div.achat*. S'inspirer du code fourni pour réaliser cette fonction.
   Il faut également garantir que la valeur de *#montant* est mise à jour.
   Ne pas oublier pas de gérer le cas où le produit ajouté est déjà présent dans le panier.
7) Gérer l'action des boutons de suppression du panier. Il faut toujours maintenir la cohérence de *#montant*.
8) Gérer l'affichage du message d'erreur si le budget est dépassé. Ne pas oublier d'enlever le message d'erreur lorsque le budget est à nouveau respecté.

Compléments : La gestion d'événements en JavaScript

En JavaScript, un **événement** est une action qui se produit et qui possède deux caractéristiques essentielles :

- C'est une action qu'on peut "écouter", c'est-à-dire une action qu'on peut détecter car le système va nous informer qu'elle se produit ;
- C'est une action à laquelle on peut "répondre", c'est-à-dire qu'on va pouvoir attacher un code à cette action qui va s'exécuter dès qu'elle va se produire.

Par exemple, on va pouvoir afficher ou cacher du texte suite à un clic d'un utilisateur sur un élément, on changer la taille d'un texte lors du passage de la souris d'un utilisateur sur un élément.

Les événements et leur prise en charge sont l'un des mécanismes principaux du JavaScript qui vont nous permettre d'ajouter un vrai dynamisme à nos pages Web.

Pour écouter et répondre à un événement, on utilise un **gestionnaire d'événements** divisé en deux parties : une partie qui va servir à écouter le déclenchement de l'événement, et une partie gestionnaire en soi qui va être le code à exécuter dès que l'événement se produit.

Aujourd'hui, en JavaScript, pour implémenter un gestionnaire d'événements, il est recommandé d'utiliser la méthode *addEventListener* plutôt que d'utiliser des attributs HTML de type événement, qui est une méthode plus ancienne, moins flexible et moins performante.

- Utilisation des attributs HTML de type événement
  Exemple : HTML : <button onclick="maFonction()">texte</button>
- Utilisation de la méthode addEventListener avec des objets Element
  Exemple :
  HTML : <button id="monBouton">texte</button>
  Javascript :

```
var monBouton = document.getElementById("monBouton");
monBouton. addEventListener("click",maFonction());
```

  On remarque qu'on passe deux arguments à la méthode : le nom de l'événement qu'on souhaite prendre en charge ainsi que le code à exécuter (qui prendra souvent la forme d'une fonction) en cas de déclenchement de cet événement.

  Dans le code de la fonction, on peut accéder à l'objet qui a reçu l'événement grâce au mot-clé *this*.