

Programmation web en js

TD n°3 : modules & tableaux

Introduction :

Dans ce TD nous allons simuler un micro-site e-commerce. L'objectif est de travailler avec les modules, les tableaux, et le DOM.

Les actions disponibles seront :

- la recherche d'un produit, sur base de mots-clés
- l'ajout d'un produit au panier,
- la suppression de l'ensemble des produits du panier : "Vider le panier"

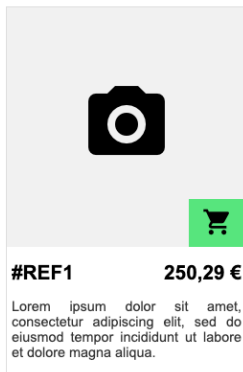
L'application se décomposera de la façon suivante :

- un module définissant la liste des produits, et les actions liées,
- un module définissant le panier, et les actions liées,
- un module gérant l'affichage de l'application,
- un module dont le rôle est de gérer l'exécution de l'application ainsi que les événements associés aux actions disponibles dans la vue,
- le module principal qui importe le précédent et lance l'application.

Éléments à votre disposition :

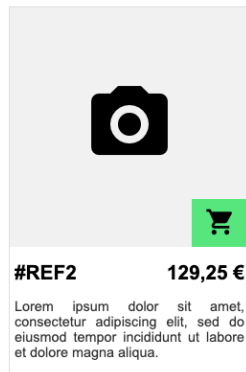
- index.html : propose un squelette graphique pour démarrer le projet ainsi que l'import du module principal
- css/cart.css : les styles de l'application.

Rechercher un produit



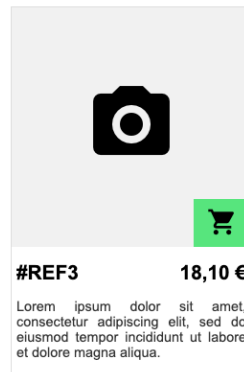
#REF1 250,29 €

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



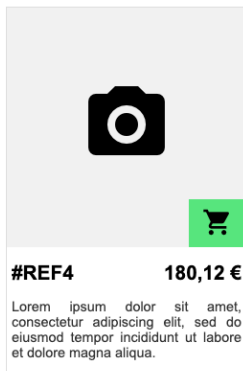
#REF2 129,25 €

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



#REF3 18,10 €

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



#REF4 180,12 €

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Mon panier (3) [Vider le panier](#)

#REF2	x2	258,50€
#REF4	x1	180,12€

Total : 438,62€

Maquette de l'application

N'hésitez pas à apporter des modifications si besoin.

ATTENTION : durant le TD, selon le navigateur utilisé, il est possible que vous rencontriez un souci de sécurité CORS en ouvrant directement le fichier index.html dans le navigateur. Pour contourner ce problème, vous pouvez y accéder en servant le projet localement via votre serveur web préféré (apache, nginx, php -S, ...), ou sur webetu. Vous pourrez ainsi y accéder via le protocole HTTP (plutôt que file:///).

Exercice 1 :

- a. Créer le module **products** (modules/products.js)
- b. Dans ce module, créer une classe pour représenter les produits :
 - ref : la référence du produit,
 - price : le prix unitaire du produit,
 - description : la description du produit,
- c. Déclarer un tableau **products**, représentant la liste des produits de l'application. Créer dans ce tableau quelques produits de tests.
- c. Exporter le tableau de produits

Exercice 2 :

- a. Créer le module **ui** (modules/ui.js)
- b. Dans ce module, créer une fonction privée **displayProduct** qui permet de construire le DOM représentant un produit
- c. Créer ensuite une fonction **buildProductsList** qui reçoit un tableau de produits en paramètre et affiche la liste des produits grâce à **displayProduct**
- d. Note : dans le squelette HTML fourni, penser à supprimer les exemples de produits
- e. Exporter la fonction **buildProductsList**

Exercice 3 :

- a. Créer le module **app** (modules/app.js) qui se charge de faire le lien entre les actions de l'utilisateur, les données et la vue. Le module doit importer les modules **products** et **ui**
- b. Dans ce module, créer une fonction **init** qui permet d'initialiser l'application. Son job est de récupérer les produits et de les fournir à la fonction **buildProductsList** du module **ui** de façon à afficher la liste des produits

Exercice 4 :

- a. Créer le module **main** (modules/main.js) qui se charge de lancer l'application
- b. Ce module doit importer le module **app**
- c. Exécuter la méthode **init** du module **app** dès que l'intégralité du DOM a été chargée
- d. Dans le squelette html, à la fin du body, ajouter une balise script de façon à intégrer ce module principal (n'oubliez pas l'attribut type="module")
- e. La liste des produits devrait alors apparaître

Exercice 5 :

- a. Dans le module **products**, ajouter une fonction **search** qui prend en argument une chaîne de caractères **keywords** et qui retourne le tableau des produits dont la référence ou la description contiennent **keywords**. Utiliser la fonction **filter** sur le tableau **products** pour réaliser cette recherche
- b. Exporter la fonction **search**
- c. Dans la méthode **init** du module **app**, attacher un événement **keyup** au champ de recherche.
- d. La fonction de callback de ce listener permettra de détecter la frappe sur la touche Entrée et appellera en conséquence la méthode **search** du module **products** en lui passant en paramètre la valeur du champ de recherche. Elle actualisera enfin la liste des produits en fonction des résultats trouvés.

Exercice 6 :

- a. Créer le module **cart** (modules/cart.js) qui se charge du panier et de ses actions
- b. Dans ce module, Déclarer une classe représentant un panier. Le contenu du panier est un tableau, initialisé à vide par le constructeur.
- c. Créer une méthode **addToCart** qui prend en paramètre un produit. La fonction ajoutera à la fin du tableau un objet contenant les propriétés suivantes :
 - **product** : le produit passé en paramètre
 - **qty** : la quantité de ce produit présente dans le panier
- d. Faire en sorte qu'en ajoutant plusieurs fois le même produit via **addToCart** celui-ci ne soit présent qu'une seule fois dans le panier, en incrémentant de 1 sa quantité à chaque appel.
- e. Exporter un objet **cart**

Exercice 7 :

- a. importer le module **cart** dans le module **ui**.
- b. Modifier la méthode **displayProduct** du module **ui** de façon à attacher un événement **click** sur le bouton d'ajout au panier. La fonction de callback de cet événement permettra d'appeler la méthode **addToCart()** en passant en paramètre le produit que l'on souhaite ajouter au panier
- c. pour récupérer le produit en question, s'appuyer sur la closure (scope de déclaration) de votre callback qui doit bénéficier du produit courant dans **displayProduct**

Exemple :

```
function displayProduct(product) {  
    ...  
    add.addEventListener('click', function(e) {  
        cart.addToCart(product)  
    })  
    ...  
}
```

Exercice 8 :

- Dans le module **ui**, créer une fonction **displayCart** qui prend en charge l'affichage du panier
- cette fonction accède au panier courant via le module **cart** et génère la table HTML contenant les produits du panier
- Pour construire le contenu de la table (innerHTML), on chainera les appels aux méthodes **map** et **reduce** sur le tableau représentant le panier : **map** va permettre de transformer chaque élément du panier en une chaîne de caractères correspondant à chaque ligne de la table (<tr>), et **reduce** permettra finalement de les concaténer en une seule String.
- La fonction **displayCart** se chargera également d'afficher le total du panier. Les outils et besoins liés au panier pouvant être amenés à évoluer dans le futur, créer une fonction **genericCalc** dans le module **cart** dont le job est de réaliser un calcul sur base des éléments contenus dans le panier. La fonction **genericCalc** attend en argument une fonction qui sera utilisée comme callback d'un reduce appliqué sur le panier, et qui retourne le résultat de ce reduce.
- À la fin de la fonction **displayCart**, afficher le prix total du panier en utilisant **genericCalc**
- Profiter de la fonction générique pour afficher également le nombre de produits présents dans le panier dans le span#total-products.
- Faire en sorte qu'après l'ajout d'un produit au panier, l'affichage de ce dernier s'actualise

Exercice 9 :

- Dans le module **cart**, créer une fonction **emptyCart** permettant de vider le panier
- Dans le module **app**, ajouter un listener permettant de détecter le clic sur le lien "Vider le panier" et utiliser la fonction emptyCart du module **cart** comme callback de cet événement
- Faire en sorte qu'une fois le panier vidé, la vue s'actualise

Exercice 10 (bonus - facultatif) :

- Dans l'exercice 7, on a ajouté un listener directement dans la fonction **displayProduct** du module **ui** afin de réagir à l'action de l'utilisateur lorsqu'il clique sur le bouton Ajouter au panier d'un produit. Ceci étant, d'un point de vue séparation des rôles (données, vues, interactions), ce n'est pas très propre. Le comportement de l'application devrait être piloté par le module **app**.
- Trouver une solution pour que le module **app** conserve la main sur le comportement à exécuter lorsque l'utilisateur clique sur le bouton d'ajout au panier après que la liste des produits ait été actualisée.

*Astuce : si **app** doit gérer le comportement, alors il faut faire en sorte que **app** fournisse une fonction qui servira de callback à l'événement **click** dans la vue. Cette fonction de callback a besoin du produit courant, pour être sûr que le clic ajoute le bon produit au bon moment ...*

Exercice 11 (bonus - facultatif) :

- Stocker le contenu du panier dans le `localStorage` de façon à ce que le panier réapparaisse en actualisant la page. N'oubliez pas l'impact de **emptyCart**

Exercice 12 (bonus - facultatif) :

- Ajouter une nouvelle propriété **photo** aux produits. La photo prendra pour valeur une url pointant sur une photo
- Adapter le module **ui** de façon à intégrer cette photo dans la vue

Astuce : vous pouvez utiliser un générateur d'images aléatoires pour cet exercice (ex : <http://placeimg.com>), ou stocker des photos localement dans le répertoire de travail.

Exercice 13 : bundler

- Installer **esbuild** : <https://esbuild.github.io/getting-started/#install-esbuild>
- Avec **esbuild**, rassembler tous les modules en 1 seul fichier `out.js` sans scripter la commande dans `package.json` . Modifier la balise `<script>` en remplaçant `main.js` par `out.js`.

Exercice 14 (bonus - facultatif) : suite bundler

- Ajouter un nouveau `script` nommé `build` dans `package.json` afin de pouvoir automatiser la commande de construction du bundle. Vérifier que la commande `npm run build` fonctionne.
- Générer une version minifiée de l'ensemble du code : lancer **esbuild** avec l'option `-minify` (ou modifier la commande scriptée).

- c. Placer le code minifié dans la balise `<script>` et vérifiez le bon fonctionnement de l'application.