# index.html / home.js

Test de la réponse de l’API

La fonction « displayTeddies() » sert à obtenir les informations d’un type de produit présent sur l’API, dans notre cas les oursons en peluche.

On va générer sur la page un contenant pour chaque ourson présent dans la base de données. On affichera :

* L’image de l’ourson ;
* Son nom ;
* Un lien sur tout le contenant qui permettra d’accéder à la fiche détaillée de l’ourson.

Si cette fonction rencontre un problème, Il n’y aura rien sur la page d’accueil en dehors de la partie HTML, c’est à dire le header, le titre « ours en peluche faits à la main » et la couleur d’arrière-plan.

Cela peut-être causé par un défaut dans la requête XHR, le serveur ne répond pas, dans ce cas-là il y aura un statut 404.

Un défaut de la boucle « for », ligne 13 – 15 qui permet l’affichage des oursons à proprement parlé.

# product.html / product.js

Test de la réponse de l’API pour un ID de produit et ajout dans le localStorage des données utiles pour le panier

La fonction « init() » initie le chargement des données pour la page produit, elle inclut tout ce qui doit se passer sur la page produit.

* Dans un premier temps, la récupération des données du produit grâce à son ID stocké dans l’URL de la page, déclenché lors du clic sur la page précédente index.html.

S’il y a un problème à ce moment-là, un statut 404 apparaîtra. La page n’aura qu’un header et l’arrière-plan de présent.

* + Une boucle « for » permet de faire apparaître le choix des différents coloris. Ligne 20-21, si elle rencontre un problème, il sera impossible de sélectionner la couleur. Le menu déroulant sera visible, mais sans texte à l’intérieur.
* Dans un second temps, le déclenchement de la fonction « tedItheCart() » (24-58) au clic sur le bouton ajouter au panier. Cette fonction permet l’ajout des données voulues au localStorage : l’id, le nom, le prix, la couleur, la quantité.

Une erreur peut survenir au moment du stockage des informations dans la variable tedStore (30-36), il y aura un message d’erreur indiquant qu’il est impossible de trouver l’id correspondant sur la page.

Une autre erreur peut apparaître pour le stockage des données dans le localStorage du navigateur (39 et 48). Les données peuvent ne pas être envoyé au bon format et seront donc illisible une fois sur la page panier. On voit apparaître [object Object] dans la console, cela signifie que les données n’ont pas été transformé au format JSON avant des stocker dans le localStorage.

Enfin, ligne 40 à 46, si cette partie ne fonctionne pas, il y aura plusieurs fois le même ours avec une couleur identique sur la page panier, au lieu de l’avoir une fois,  avec les quantités additionnées.

Fonctionnement normal :

* J’ajoute Norbert, couleur chocolat en quantité 2,
* J’ajoute une nouvelle fois Norbert couleur chocolat, en quantité 3,
* Dans le panier j’ai une colonne, avec un Norbert couleur chocolat, en quantité 5.

Si la fonction rencontrait un problème nous aurions deux Norbert, une fois en quantité 2, une autre fois en quantité 3.

* + Et enfin une alerte confirmant l’ajout au panier.

# cart.html / cart.js

Test de la récupération des données dans le localStorage, envoi du formulaire complété et des id produits.

La fonction « cartResume() » permet l’affichage des données dans le localStorage, le nom, le coloris, le prix et enfin la quantité. On utilise une boucle qui permet de générer une boite pour chaque item dans le panier et de faire l’addition du montant de chaque ourson dans le panier, pour afficher le total une fois la boucle terminée.

Si rien de s’affiche en dehors du header et du formulaire sur la page, c’est la boucle (58-61) ne fonctionne pas correctement. C’est elle qui génère le récapitulatif du produit, nom, couleur, prix, quantité, et fait le calcul du total qui est ensuite affiché en dessous.

La fonction « sendingData() » est une requête de type « post » qui permet d’envoyer des données à l’API, à savoir les données dans le formulaire, nom, prénom, email, adresse et ville ainsi que les id des produits dans le panier. Avec l’envoi une fois ces étapes terminées vers une page de confirmation de commande.

S’il y a un problème, le numéro de commande ne s’affichera pas sur la page suivante, il y aura écrit « undefined » à la place. C’est qu’il y a eu une erreur sur les données qui ont été envoyé à l’API (bad request 400). Il faudra alors vérifier les données de la variable « teddiesOrder » (74-77) et leur format.

La fonction « suppItem() » permet de supprimer un item du localStorage une fois arrivée sur la page panier. Si cette fonction rencontre un problème, il sera impossible de supprimer un item. L’item peut disparaître sans que le total soit modifié, ce qui voudra dire qu’il y a un problème au moment de la récupération du localStorage, et/ou du renvoi des données. Si rien de se passe au clic, il faudra vérifier les arguments de la fonction. Enfin si le total ne se modifie, c’est qu’il peut y avoir un problème du côté de la fonction modifyTedTotal (33-42).

La fonction « modifyQuantity() » permet de modifier la quantité de chaque item présent sur la page et recalculer le total. Les problèmes possibles sont les mêmes que la fonction précédente.

La fonction « modifyTedTotal » permet d’actualiser le total lorsqu’on supprime ou modifie la quantité d’un item. Si la modification du total ne se fait, c’est peut-être la boucle (36-38) qui ne se déclenche pas, ou que le résultat de la boucle ne s’affiche pas (41).

# order.html / order.js

Affichage des données dans l’URL, et nettoyage du localStorage en arrivant sur la page.

La fonction « thankYou() » déclenche le nettoyage du localStorage, ce qui permet d’avoir une page panier ou il n’y a plus d’items après confirmation de la commande, et dans un second temps, d’afficher un message contenant l’ID unique de la commande et le total de la commande.