BOUBTTA Mentor : Soma Bini

Hicham

**Test unitaire du site ORINOCO**



CTO : Paul Gomez OPENCLASSROOMS Projet 5

**Sommaire**

1. La page accueil
2. La page produit
3. La page panier
4. La page de confirmation

I) La page d’accueil

La page d’accueil va afficher toutes les données provenant de l’API de manière ordonnée afin de permettre aux visiteurs du site de choisir parmi le panorama de produits proposés par l’entreprise. Voyons ici comment cela se traduit dans le code.

Lignes de code testées : 3 à 20

Fonction testée : *getdata*

Résultats attendus : La fonction getdata permet de lancer un fetch afin communiquer à l’API et de permettre donc d’accéder aux données.

Comment vérifier le résultat attendu : pour vérifier que la fonction marche correctement il faudra faire un console.log de de data dans le second then pour vérifier que les résultats sont bels et bien dans la console du navigateur web

Problème attendu : Les données peuvent ne pas s’afficher donc le fetch pourrait avoir echouer dans sa tentative de communication avec l’API et cela se traduirait donc par un message d’erreur dans la console comme « data is not a function ».

Lignes de code testées : 22 à 47

Fonction testée : *datacards*

Résultats attendus : La fonction permet de lancer un code HTML contenu dans une boucle qui itérer sur chaque produit contenu dans l’API. On devrait donc voir tous les produits apparaitre dans la page d’accueil et de manière dynamique.

Comment vérifier le résultat attendu : Les produits devraient s’afficher dans la page d’accueil et pour vérifier que l’itération a marchée sur chaque produit il faudrait faire un console.log du container HTML dans lequel se trouve les produits. Normalement, plusieurs balises div devraient avoir été créés pour chaque peluche.

Problème attendu : l’itération peut avoir échouées et donc un seul produit peut avoir été affiché.

1. La page produit

La page produit permet d’afficher individuellement les produits à l’utilisateur qui peut par la suite choisir les différentes options et la quantité souhaité. Cela est possible grâce à l’interface URLsearchparams qui affiche les produits selon individuellement grâce à l’id que l’on ajoute en paramètre de l’adresse de l’API.

Lignes de code testées : 10 à 18

Fonction testée : *coloroption*

Résultats attendus : la fonction permet d’itérer sur chaque option des peluches, car en effet chaque peluches possède un nombre d’options différent. Le résultat est simple, cette fonction que l’on introduit dans la balise selecte ayant pour id « color » doit montrer à l’utilisateur toutes les options que possède un produit.

Comment vérifier le résultat attendu : il faut activer le menu déroulant et regarder si les options sont toutes présentes.

Problème attendu : si la fonction ne marche pas, il devrait y avoir écrit dans le menu déroulant « undefined ».

Lignes de code testées : 40 à 50

Fonction testée : *addproduct*

Résultats attendus : la fonction se déclenche au clic sur le bouton « ajouter au panier ». Elle a pour paramètre le nom et le prix du produit. A chaque clic sur le bouton un objet « product » est créer contenant la quantité, le prix et le nom de l’objet choisis. Après une constante panier est crées et envoyée dans le local storage.

Comment vérifier le résultat attendu : il doit y avoir autant d’objet créer que de produit choisis par l’utilisateur. Pour vérifier cela il faudra alors consulter le local storage pour voir si le panier a bien été envoyé avec les produits à l’interieur.

Problème attendu : si sur le bouton les mauvais parametre n’ont pas été introduit dans la fonction, la console affiche une erreur. Il faut mettre en parametre de la fonction, dans la balise HTML les données (sous forme de data.) issues de l’API.

Lignes de code testées : 52 à 81

Condition testée : si le panier est vide on crée un tableau panier dans lequel on met l’objet product, ensuit on envoie le panier dans le local storage après avoir été stringifier. Dans le cas ou il existe un panier rempli dans le panier : si c’est produit existant on ajoute seulement la quantité, dans le cas contraire on ajoute le produit au tableau panier. A la fin de la condition on envoie au local storage le panier qui est un tableau de produit. Le panier est bien sur stringifier avant.

Résultats attendus : Les produits en plusieurs exemplaire ne doivent avoir que leur quantité qui est mise à jour et les nouveaux produits doivent avoir une place a part entière dans le tableau

Comment vérifier le résultat attendu : Il faut regarder le local storage pour voir si tous les produits choisis sont présents.

Problème attendu : Il peut y avoir un problème au niveau du calcul des quantité et donc un produit peut apparaitre plusieurs fois selon le nombre de fois ou l’utilisateur à cliquer sur le bouton « ajouter au panier »

1. La page Panier

Lignes de code testées : 15 à 22

Fonction testée : *postapi*

Résultats attendus : D’abord, la fonction récupère les données taper par l’utilisateur dans le formulaire, ces données sont ensuite stockées dans des variables, il en est de meme pour l’id du produit. Ensuite on crée un objet contact pour les données du formulaire sont stockées. L’id du produit et les données du formulaire sont mis dans un objet qui est envoyé par la method POST et un fetch à l’API. On profite de ce fetch également pour envoyer dans le local storage .

Comment vérifier le résultat attendu : il faut aller dans le menu network du navigateur pour voir si l’envoie post à marcher. De meme que faire un console.log de l’id de commande serait judicieux, car si on l’obtient cela voudrait dire que la fonction a marché.

Problème attendu : lorsque le fetch et le post sont mal executés, notamment lorsque les routes de l’API ne sont respectées, la console renvoi le message « post error 400 bas request ».

Lignes de code testées : 60 à 74

Fonction testée : *isFromValid*

Résultats attendus : La fonction reprends les values du formulaire, qui sont ensuite soumises a des regex. Si toutes sont égale a true la fonction retournera comme résultat « true » sinon se sera false. La fonction est appelée lorsque le l’utilisateur clique sur le bouton commander.

Comment vérifier le résultat attendu : Si la fonction est égale à true et donc que tous les champs sont bien remplis les données sont envoyées avec la fonction *postapi .* Dans le cas contraire un message d’erreur apparaitre pour indiquer à l’utilisateur une erreur dans le formulaire

Problème attendu : Il est possible qu’il y ait une erreur au niveau des regex et que des champs qui devraient être invalide soit valide et ainsi permettre par mégarde d’envoyer quand même le client sur la page de confirmation.

Lignes de code testées : 75 à 80

Fonction testée : *dellIndexPanier*

Résultats attendus : La fonction doit permettre d’effacer un à un les produits du panier en cliquant sur l’icone poubelle et dans le code avec array.splice. Une fois le produit supprimé on met à jour le panier en revoyant un nouveau panier dans le local storage, la page est ensuite mise à jour avec widow.location.href = « panier.html »

Comment vérifier le résultat attendu : il faut cliquer sur l’icone car la fonction s’exécute grâce au onclick dans la balise HTML de l’icône. Si le produit s’efface entièrement et que le prix final est mis à jour cela veut dire que la fonction marche.

Problème attendu : si le produit ne disparait pas du tableau il faudra vérifier la localisation du widow.location.href

Lignes de code testées : 82 à 86

Fonction testée : *viderpanier*

Résultats attendus : la fonction permet de vider entièrement le tableau en effaçant tous ce que contient le local storage, ensuite la page est mise à jour avec widow.location.href= « panier.html »

Comment vérifier le résultat attendu : Le tableau devrait normalement se vider entièrement, le formulaire devrait avoir disparu pour laisser place à un message invitant l’utilisateur à choisir un produit. Pour s’assurer que tous fonctionnent bel et bien, il faut aller voir le local storage et vérifier que soit bien vide.

Problème attendu : Les produit peuvent disparaitre mais rester dans le local storage et cela se traduit par une non mise a jour du prix final.

Lignes de code testées : 87 à 189

Fonctions testées : validName, validPrenom, validAdress, validCity, validEmail, validPhone.

Résultats attendus : Ces fonctions vérifient si les champs remplis par l’utilisateur sont corrects. Les values de l’utilisateur sont soumises à des regex dans le code, ce sont elles qui permettent de déterminer si les values sont bonnes ou fausses.

Comment vérifier le résultat attendu : Si les values sont mauvaises un message d’erreur s’affiche et les border de l’input concerné deviennent rouges. Si les values sont bonnes les border deviennent verte et un message indiquant que le contenu est bon s’affiche sous le label.

Problème attendu : Il est possible qu’une value soit compté comme bonne alors qu’elle devrait être perçue comme mauvaise par le code (comme rajouter des chiffres à un prénom). Si cela arrive, il faudra vérifier si l’écriture des regex est bonne.

Lignes de code testées : 193 à 248

Fonction testée : main

Résultats attendus : La fonction main est la principale fonction de la page et elle incluse dans une condition. Elle permet en amont de calculer les prix des produits et du montant total, de créer une ligne dans le tableau pour chaque produit choisis et de lancer les fonctions permettant à l’utilisateur de savoir si ces données du formulaire sont bonnes ou fausses, de même que d’exécuter la fonction postapi vu précédemment.

Comment vérifier le résultat attendu : C’est la fonction principale de la page, celle ou est appelé toutes les autres fonctions, pour vérifier que celle-ci marche il faudra tester toutes les fonctionnalités de la page.

Problème attendu : comme c’est la fonction qui lance la majorité des fonctions de la page donc les problèmes peuvent être multiple. Tout dépend à quel niveau est situer le problème. Il faudra faire un console.log de la zone ou est situé le problème.

1. La page confirmation

Lignes de code testées : 1 à 8

Fonction testée : *clearend*

Résultats attendus : cette fonction permet de vider tous ce que le local storage contient lorsque le l’utilisateur appuie sur le bouton pour revenir à la page d’accueil.

Comment vérifier le résultat attendu : il faut cliquer sur le bouton pour revenir à la page d’accueil et ensuite retourner sur la page panier pour vérifier que le tableau a bien été vidé.

Lignes de code testées : 9 à 22

Fonction testée : *principale*

Résultats attendus : c’est la principale fonction de la page, elle doit permettre d’afficher un message de remerciement à l’acheteur, un résumé du montant de l’achat et l’id de commande.

Comment vérifier le résultat attendu : toutes les informations citées plus haut doivent s’afficher

Problème attendu : les données du local storage peuvent ne pas s’afficher, en général il y a écrit à la place de ces manquantes « undefined », si cela arrive vérifiez bien au niveau du nom des clefs du local storage et du chemin pour entrer dans le tableau afin de chercher les données souhaitées.