PLAN DE TESTS UNITAIRES

1. INTRODUCTION

Une application doit fonctionner correctement, et cela, dans tous les cas de figure.
Prendre le temps de tester cette application est primordial si nous ne voulons pas prendre le risque que le produit ne fonctionne pas devant un client par exemple.
Il existe plusieurs sortes de tests pour tester et couvrir le fonctionnement d'une application : les tests unitaires qui nous concernent ici, les tests d'intégration, et les tests fonctionnels.
Le plan de tests devrait être réalisé dès la conception de l'application.
Un test unitaire isole et teste une partie du code.

Nous allons aborder dans ce plan les objectifs de l'application ainsi que les parties du code qui doivent être testées.

2. LES ÉLÉMENTS À NOTRE PORTÉE

N'ayant pas réalisé la partie backend du code, nous ne testerons pas cette partie-là du code et partons du principe que la base de données et l'api fonctionnent correctement.
Nous allons donc couvrir l'ensemble des fonctions et méthodes du frontend de l'application.

3. LES OBJECTIFS DE L'APPLI ET LES TESTS À EFFECTUER

A. La page index.js / Afficher la liste de produits

Ligne 2 : Tester si l'appel à l'api avec fetch retourne les produits

Ligne 17 : Tester si la fonction showProducts() qui prend comme argument les données reçues et affichent les produits sous forme de liste.

Ligne 78 : Tester si la fonction loadCartItems() récupère le nombre d'articles dans le localstorage et les affichent dans le compteur d'articles de la barre nav.

B. La page product.js / Afficher le produit, le personnaliser et le mettre dans le panier

Ligne 15 : Tester si l'appel à l'api avec fetch retourne le produit.

Ligne 30 : Tester si la fonction productCard qui prend comme argument les données du produit reçu pour les afficher dans le dom et les stocker dans le localstorage.

3 fonctions sont intégrées à la précédente :

Ligne 52 : Tester si la fonction lensOption() crée les choix de personnalisation ligne 55 avec la boucle for, et stocke la valeur de l'option ligne 62 dans le localstorage.
Ligne 69 : Tester si la fonction setQty() stocke la valeur de l'input dans le localstorage.

Ligne 79 : Tester si la fonction getAddBtn() appelle au clic les 2 fonctions suivantes : updateCart et addItem et modifie l'état du bouton.

Ligne 95 : La fonction updateCart() : Test 1 => si (le nombre d'articles dans le panier est vide) la valeur inscrite dans le compteur d'article soit égale à la quantité sélectionnée.

Test 2 => si (le nombre d'articles dans le panier est > 0) la quantité sélectionnée soit ajouté au à la valeur inscrite dans le compteur.

Ligne 122 : Tester si la fonction addItem() crée un nouvel objet Item et l'ajoute au tableau.

Autre test à effectuer : si aucune quantité n'est sélectionnée par l'utilisateur, la quantité stockée par défaut dans le localstorage est égale à 1.

C. La page cart.js / Lister les articles choisis dans le panier, afficher le total du panier

Ligne 23 : Tester la fonction showItemsInCart() si le panier est vide (modification du dom) et s'il y a un ou des articles dans le panier (affichage des produits dans le panier).

Ligne 5 : Tester si la fonction totalPrice() affiche la prix total.

Ligne 71 : Tester si le bouton 'vider le panier' vide le localstorage.

D. La page form.js / Valider le formulaire et envoyer au serveur les infos client

Ligne 19 : Tester si pour chaque produit ajouté au tableau l'id est récupéré et ajouté au tableau.

Ligne 28 : Tester si l'évènement crée un nouvel objet contact et le stocke dans le localstorage.

Ndlr : Le formulaire étant validé côté client avec des attributs html, les tests cases ne sont pas réalisés. Si le formulaire n'est pas validé l'envoi n'est pas possible.

Ligne 45 : Tester le post à l'api avec fetch.

Ligne 85 : Tester si la fonction getOrder() stocke les data dans le localstorage.

E. La page order.js / Remercier et donner un numero de commande au client

Ligne 3 : Tester la fonction displayOrderInfo() qui modifie le dom et vide le localstorage.