KURDI Lora
Projet n°5

PLAN DE TESTS UNITAIRES

Test de toutes les fonctions des 4 pages JS :

1. home.js
2. product.js
3. cart.js
4. confirmedOrder.js

Fichier JS	Lignes de code testées	Fonction testée	Résultat attendu	Comment vérifier le résultat attendu
home.js	8 à 26	fetch (get)	Le fetch doit retourner une promise contenant un tableau, chaque élément étant également un tableau.	On peut console.log la promise et/ou son contenu, sinon le catch(err) nous indique l'erreur en question (si le serveur ne répond pas par exemple)
home.js	30 à 34	fillIdNamePriceEquivalenceArray	La fonction prend en paramètre un tableau. Elle doit remplir un tableau JS idNamePriceEquivalence, contenant pour chaque produit retourné par la promise précédente, un objet JS ayant pour clés : l'id, le nom et le prix du produit en question.	On peut console.log dans la boucle for pour vérifier que l'on récupère bien un 'produit' de type tableau. On peut console.log le tableau idNamePriceEquivalence après remplissage, ou regarder dans sessionStorage. On peut aussi console.log sa taille, elle doit être égale à celle du tableau entré en paramètre.
home.js	39 à 62	insertNewProduct	La fonction prend en paramètre un tableau et renvoie pour chaque élément une div contenant : un lien, une image et une description (3 fonctions distinctes). La div est ajoutée à un container implémenté en html.	Il doit y avoir autant de divs qui apparaissent sur la page d'accueil que d'éléments dans le tableau d'entrée, avec pour chaque produit : la bonne image, le bon nom, le bon prix, et la bonne descpription.
home.js	66 à 70	addLinkToProduct	La fonction renvoie un lien pour un élément du tableau, et ayant en référence le lien spécifique vers la page produit en utilisant la donnée '_id' de l'élément renvoyé par la promise.	On peut console.log le lien ajouté en href pour l'élément en cours, et vérifier que l'on récupère tous les id du tableau renvoyé par la promise.
home.js	73 à 79	addImageToProduct	La fonction renvoie une figure contenant une image pour un élément du tableau. L'image est récupérée grâce à la donnée 'imageUrl' est assignée comme source de l'image. Cette dernière est ajoutée à la figure.	On peut console.log les éléments html figure et l'image imbriquée, pour vérifier que la construction a fonctionné, pour chaque produit du tableau.
home.js	82 à 99	addDescriptionToProduct	La fonction renvoie une div contenant une brève description du produit : le nom et le prix, qui sont récupérés grâce aux données 'name' et 'price'. Les div name et price sont créées séparémment puis sont ajoutées comme enfant de la div de description.	On peut console.log l'élément divDescription pour voir que la construction a bien fonctionné.
product.js	8 à 39	fetch(get+_id)	Le fetch doit retourner une promise contenant le tableau du produit en question, avec en données : la liste des vernis, l'image, le prix, la description, l'id et le nom. Le fetch récupère ces données pour remplir le texte des éléments html prévus à cet effet, ainsi que rajouter toutes les options de vernis dans le select prévu à cet effet.	On peut console.log la promise et/ou son contenu, sinon le catch(err) nous indique l'erreur en question (si le serveur ne répond pas par exemple)
product.js	44 à 54	addOptionToSelect	Fonction qui prend en paramètre un string et renvoie un élément de type 'option' ayant pour valeur et pour text le string en question.	On peut console.log le string entré en paramètre pour vérifier que l'on boucle bien sur toute la liste des vernis. On peut aussi console.log l'élément html 'option' qui est retourné pour vérifier son contenu (texte, et valeur).
product.js	57 à 59	displayHeart	Fonction qui display flex un conteneur qui contient un coeur en position absolute sur l'image du produit et qui était en display none par défaut. Le coeur est affiché à l'appel de chaque produit dans le fetch.	Vérification visuelle que le coeur s'affiche bien en haut à droite de l'image dans toutes les pages produit.

product.js	73 à 77	ajout d'un eventListener à la selection d'une option de vernis	La fonction associée ajoute un éventListener au change du l'élément select, elle attribue la valeur choisi à une variable globale 'product_varnish'. Elle réattribue la couleur par défaut au texte, au cas où il était repassé au rouge.	On peut console.log la valeur de vernis qui est récupérée dans la variable globale.
product.js	89 à 99	ajout d'un eventListener à la l'input pour le choix du nombre d'articles	La fonction associée récupère la valeur entrée au clavier par l'utilisateur et la parse en float au cas où l'utilisateur entre un nombre décimal. Si c'est un entier, on récupère la valeur dans une variable globale 'product_quantity' et on redonne les couleurs par défaut au texte (backToInitialColor) au cas où il est passé au rouge. Sinon on avertie l'utilisateur qu'il faut une valeur entière comprise entre 1 et 20, et on fait passer le texte en rouge avec changeColor().	On peut console.log la quantité entrée par l'utilisateur, et vérifier le comportement selon si cest un entier ou un décimal. On peut omettre de choisir une quantité et la laisser à 0, laisser l'alert s'afficher et voir le texte passer au rouge. Essayer de choisir une quantité en entrant une valeur au clavier Est-ce que le texte reprend sa couleur initiale ?
product.js	105 à 118	ajout d'un eventListener au clic des 'boutons' + et – pour le contrôle des quantité	La fonction associée met à jour la variable globale 'product_quantity' et la valeur visible de l'input 'quantity' selon si on clique sur + ou sur - . Dans les 2 cas, on redonne au texte ses couleurs par défaut au cas où il serait passé au rouge grâce à backToInitialColor.	On peut console.log la variable globale product_quantity lorsqu'on clique sur + ou – et vérifier visuellement que la valeur de l'input suit. On peut omettre de choisir une quantité et la laisser à 0, laisser l'alert s'afficher et voir le texte passer au rouge. Essayer de choisir une quantité en cliquant sur les boutons + ou - . Est-ce que le texte reprend sa couleur initiale ?
product.js	122 à 125	backToInitialColor	Fonction qui redonne la couleur initiale aux div 'Quantité' et 'Vernis', si elles étaient passées au rouge quand l'utilisateur s'est trompé. Ne renvoie rien.	Vérification visuelle au clic des boutons + et – de la div Quantité et l'input quantity et au clic d'une option dans la liste des vernis que le texte reprend sa couleur initiale.
product.js	131 à 140	checkVarnish	Fonction qui prend en paramètre un string et qui renvoie 'false' si ce string vaut 'default' ou s'il est vide. Elle renvoie true sinon.	On peut console.log le string récupéré en paramètre et le booléen renvoyé. On peut tester d'une part en ne choisissant pas de vernis et d'autre part en choisissant une option de vernis dans la liste.
product.js	143 à 150	checkQuantity	Fonction qui prend en paramètre un integer et qui renvoie false si elle est inférieure à 1 ou suprérieure à 20. Elle renvoie true sinon.	On peut console.log la valeur récupérée en paramètre et le booléen renvoyé. On peut tester en laissant la quantité à 0, ou en entrant une valeur à la main ou avec les boutons + et -, ou en essayant d'aller au-delà de 20 articles.
product.js	156 à 158	changeColor	Fonction qui prend en paramètre un élement html et qui lui modifie son style en rouge.	On peut console.log l'élément html récupéré en paramètre et vérifier visuellement que le texte passe au rouge lorsque l'on omet volontairement de choisir un vernis ou une quantité.
product.js	166 à 169	displayAndRemoveConfirmationDiv	Fonction qui display flex une div verte qui confirme l'ajout d'un produit au panier, et qui était en display 'none 'par défaut. Elle repasse en display 'none' au bout de 2,5 secondes.	On peut tester que la div apparaît bien au clic du bouton 'Ajouter au panier' et qu'elle disparaît bien au bout de 2,5s. Tester qu'elle a le meme comportement si on décide de rajouter le même produit (si on s'était trompé et qu'on en veut 1 de plus par exemple). Tester qu'elle fonctionne pour toutes les pages produits.

product.js	219 à 245	updateCart	Fonction qui prend en paramètre un id, un nom, et une option de vernis et qui met à jour le tableau JS ‘cart’ en fonction de ce qu’il contient déjà. Soit cest la quantité d’un produit qui est mise à jour si celui ci est existant (mm id et mm vernis), soit c’est une ligne entière qui est ajoutée au tableau ‘cart’ avec le nom du produit, son vernis et sa quantité.	On peut console.log les valeurs qui sont entrées en paramètres. On peut console.log le ‘tableau’ cart pour chaque cas de figure. Vérifier le contenu du tableau en testant chaque cas : ajout d’un nouveau produit, ajout du meme produit avec le meme vernis, ajout du meme produit avec un vernis différent.
product.js	252 à 306	ajout d’un eventListener au bouton ‘Ajouter au panier’	Un écouteur d’évènement est ajouté au clic du bouton. La fonction associée récupère les valeurs booléennes des fonctions checkVarnish et checkQuantity pour vérifier que l’utilisateur a bien choisi une option et a entré une quantité comprise en 1 et 20. Si ces 2 valeurs sont à true, alors on rempli le tableau ‘cart’ : si il est vide et que c’est la première fois, on push directement, sinon on le met à jour grâce à la fonction updateCart. Le contenu du panier est stocké dans sessionStorage. La fonction calcule le nombre total d’articles et sauvegarde la valeur dans sessionStorage. La fonction affiche cette valeur dans une parenthèse qui apparaît dans le menu du header. La fonction permet l’accès au panier (ajout du lien vers la page panier) lorsque celui contient des articles (lien vide lorsque le panier est vide). Si une des 2 valeurs booléennes renvoyées par checkVarnish ou checkQuantity est ‘false’ alors on change la couleur au rouge.	On peut vérifier dans sessionStorage que toutes les valeurs et objets se mettent à jour correctement en fonction du cas de figure. (ajout au panier et nombre d’articles). On peut délibéremment ne pas ajouter de vernis et ne pas choisir/entrer de quantité pour vérifier que le passage au rouge et les alert fonctionnent et que le texte reprend sa couleur initiale lorsque les choix ont été faits. On peut vérifier que l’on n’a pas accès au panier lorsque celui ci est vide et que l’on y a accès lorsqu’il y a des articles.
cart.js	13 à 19	searchName	Fonction qui boucle sur le tableau idNamePriceEquivalence construit sur la page d’accueil. Elle prend en paramètre un id et renvoie le nom du produit.	On peut console.log l’id qui est récupéré, le nom qui est renvoyé et vérifier que cela est cohérent.
cart.js	21 à 27	searchPrice	Fonction qui boucle sur le tableau idNamePriceEquivalence construit sur la page d’accueil. Elle prend en paramètre un id et renvoie le prix du produit.	On peut console.log l’id qui est récupéré, le prix qui est renvoyé et vérifier que cela est cohérent.
cart.js	30 à 39	updateCart	Fonction qui prend en paramètre un id et une quantité, et met à jour dans le tableau cart la quantité du produit qui possède l’id en question. La variable cart dans sessionStorage est également mise à jour.	On peut console.log l’id et la quantité en question et tester l’ajout ou le retrait d’un article grâce aux boutons + et – sur la page panier. Varifier que le panier dans sessionStorage est correctement mis à jour.
cart.js	48 à 70	updateTotalPrice	Fonction qui mets à jour le montant total de la commande, en bouclant sur un tableau (ce sera ‘cart’). La valeur du montant total est affichée dans l’élément html prévu à cet effet. Le nombre d’articles total est également mis à jour dans cette fonction pour l’afficher dans la parenthèse près du panier.	On peut vérifier visuellement que le montant total calculé, affiché et sauvegarder dans sessionStorage est le meme quand on ajoute ou supprime un article du panier (grâce aux boutons + et – ou grâce à l’input quantité)

cart.js	77 à 91	updateAll	Fonction qui prend en paramètre : l'id du produit que l'on manipule, sa variable qui réfère à sa quantité, la ligne à laquelle le produit se situe dans le tableau du panier affiché, la nouvelle valeur de la quantité. Cette fonction permet de mettre à jour : la quantité du produit, le panier grâce à updateCart(), le prix total grâce à updateTotalPrice(), et le sous total à afficher dans la bonne ligne et en recalculant le montant (multiplication de la nouvelle quantité par le prix du produit en le cherchant avec searchPrice()).	On peut console.log tous les paramètres d'entrée et vérifier qu'ils correspondent au cas testé (modifier la quantité d'un produit du panier). Vérifier que le sous-total se mets à jour, le total, la parenthèse, et la panier dans sessionStorage.
cart.js	98 à 162	addQuantityControl	Fonction qui ajoute à chaque ligne du tableau affiché du panier, une div de contrôle des quantités (‘quantityControl’) : le texte ‘Quantité’, le bouton -, l'input, le bouton +. La fonction ajoute les classes nécessaires au style et à la gestion des eventListener (classe ‘control’)	Vérification visuelle que la div apparaît pour chaque ligne du tableau du panier.
cart.js	*inclus* 127 à 142	ajout d'un eventListener aux boutons + et – de la div ‘quantityControl’	La fonction associée met à jour la variable globale de la quantité du produit ‘quantity’ selon que l'on clique sur le bouton + ou – et si la valeur à modifier est respectivement, ≥ à 0, ou ≤ à 20. Dans les 2 cas, toutes les variables et objets sont mis à jour grâce à la nouvelle quantité et à la fonction updateAll() .	On peut console.log la quantité avant et après modification, essayer de cliquer sur + quand la valeur à 20, et essayer de cliquer sur – quand la valeur est à 0. Vérifier dans sessionStorage que toutes les variables se mettent à jour comme attendu.
cart.js	*inclus* 145 à 158	ajout d'un eventListener à l'input de la div ‘quantityControl’	La fonction associée met à jour la variable globale de la quantité du produit ‘quantity’ si la valeur entrée au clavier est > à 0 ou ≤ 20. Dans ce cas également, toutes les variables et objets sont mis à jour grâce à la fonction updateAll(). Sinon des alert sont levées.	On peut console.log la quantité avant et après modification, essayer de cliquer sur + quand la valeur à 20, et essayer de cliquer sur – quand la valeur est à 0. Vérifier dans sessionStorage que toutes les variables se mettent à jour comme attendu.
cart.js	166 à 181	deleteProduct	Fonction qui ajoute un eventListener à un élément html au clic et qui supprime un produit du tableau affiché, du tableau ‘cart’ et de l'objet dans sessionStorage. Met également à jour le montant total de la commande et le nombre total d'articles dans le panier.	On peut console.log l'index du produit supprimé, le tableau ‘cart’ et la variable dans sessionStorage après suppression du produit, vérifier visuellement que le tableau affiché se mets correctement à jour, au clic du bouton ‘Supprimer’ dans la ligne que l'on souhaite supprimer. Vérifier dans sessionStorage que le nombre d'articles et le total se mettent correctement à jour.
cart.js	186 à 210	addDeleteButton	Fonction qui ajoute une div de suppression du produit sous la div de contrôle des quantités, pour chaque ligne du tableau affiché du panier. La div contient une icône ‘poubelle’ et un texte ‘Supprimer’ sur le quel on applique la fonction deleteProduct().	On peut vérifier que la div est bien ajouté pour chaque ligne du tableau, et qu'au cllic de ‘Supprimer’ c'est la bonne ligne qui est supprimée.
cart.js	217 à 253	addNewLine	Fonction qui ajoute une ligne dans un élement tbody à partir d'un tableau donné en paramètre. Pour chaque élément et dans sa ligne, sont ajoutés 4 cellules. La 1ère contenant le nom et le vernis du produit, la 2ème contenant le prix unitaire du produit, la 3ème contenant le contrôle des quantités et la div de suppression du produit (addQuantityControl() et addDleleteButton()), et la 4ème le montant du sous-total.	On peut console.log les paramètres d'entrées : l'id du produit, le nom, le vernis, la quantité, le prix. Vérifier que les lignes sont complètes et correctes pour chaque élément du panier.

cart.js	264 à 278	fillCartPage	Fonction qui prend en paramètre un tableau, qui récupère les données suivantes pour chaque élément : l'id, le nom, le vernis, le prix, la quantité. Pour chaque élément, lui appliquer la fonction addNewLine() . La fonction affiche le total de la commande à l'arrivée sur la page du panier et le calcule grâce à la fonction updateTotalPrice appliqué au tableau.	Vérifier que les données affichées dans le tableau de la page panier correspondent au données sauvegardées dans l'objet cart de sessionStorage, rempli au passage des pages product.
cart.js	310 à 319	checkText	Fonction qui vérifie qu'un string entré en paramètre est conforme à la regex définie pour les champs texte du formulaire de contact. Si oui renvoie true, sinon renvois false.	On peut console.log le string entré en paramètre, et console.log qqch quand on est dans le cas true et autre chose quand on est dans le cas false.
cart.js	322 à 330	checkAddress	Fonction qui vérifie qu'un string entré en paramètre est conforme à la regex définie pour les champs adresse et complément (qui accepte les lettres, les chiffres et les espaces). Si oui renvoie true, renvoie false sinon.	On peut console.log le string entré en paramètre, et console.log qqch quand on est dans le cas true et autre chose quand on est dans le cas false.
cart.js	334 à 342	checkPostalCode	Fonction qui vérifie qu'un string entré en paramètre est conforme à la regex définie pour le champ du code postal et qui accepte 5 digits. Si oui renvoie true, renvoie false sinon.	On peut console.log le string entré en paramètre, et console.log qqch quand on est dans le cas true et autre chose quand on est dans le cas false.
cart.js	346 à 355	checkEmail	Fonction qui vérifie qu'un string entré en paramètre est conforme à la regex définie pour le champ de l'adresse e-mail (qui accepte au début du string des lettres, des chiffres et des caractères, un @, des lettres, un '.' et se termine par 2 à 3 lettres). Si oui renvoie true, renvoie false sinon.	On peut console.log le string entré en paramètre, et console.log qqch quand on est dans le cas true et autre chose quand on est dans le cas false.
cart.js	358 à 32	SetInitialColor Non utilisée, ne pas prendre en compte		
cart.js	373 à 436	addEventToFields	Fonction qui ajoute à chaque élément du formulaire un écouteur d'évènement au change. Si ce sont des champs de type texte, comme le nom, le prénom et la ville, alors on vérifie l'input grâce à checkText(), si oui on met à jour la valeur du champ. Si c'est un champ d'adresse ou de complément, idem avec checkAddress(), si c'est le champ code postal idem avec checkPostalCode(), si c'est le champ email idem avec checkEmail(). Si ce n'est pas conforme aux regex respectives, une classe est ajoutée au champ pour modifier son sytle et indiquer à l'utilisateur que le champ est faux. La fonction sauvegarde la valeur de l'email dans sessionStorage. La fonction rempli un tableau 'contact' si le champ est requis (tous sauf complément et code postal).	On peut console.log la valeur entrée par l'utilisateur pour chaque champ, la valeur booléenne renvoyée par les fonctions de vérification, le contenu de l'objet contact à la fin de la boucle.

cart.js	471 à 496	send	Fonction qui fetch avec une méthode POST un objet ‘data’ contenant l’objet ‘contact’ et le tableau ‘products’. Le fetch récupère l’orderId renvoyé par la promise et la valeur est sauvegardée dans sessionStorage. Si la valeur est bien renvoyée, alors l’utilisateur est redirigé vers la page de confirmation de commande.	On peut console.log l’objet data pour vérifier sa structure. On peut console.log l’orderId récupéré par la promise.
cart.js	501 à 512	ajout d’un eventListener au bouton ‘Commander’	La fonction associée ajoute un écouteur d’évènement au clic du bouton. Si le nb d’articles et le panier sont remplis et que la validation des champs du formulaire sont tous à ‘true’ alors le fetch est lancé grâce à send(). Sinon, on alerte l’utilisateur que son panier est vide et qu’il ne peut commander. On en profite pour effacer les champs et supprimer l’email sauvegardé.	On peut essayer de commander quand tous les champs sont correctement remplis et quand ils ne le sont pas ou pas correctement, avec panier plein ou vide.
confirmedOrder.js	pas de fonction	récupération des valeurs de sessionStorage	Récupération de : orderId, emailAdress et totalPrice pour affichage sur la page dans les éléments html prévus à cet effet.	Vérifier visuellement que les données sont affichées.