	Page web / Fichier JS	Fonction testée	Lignes testées	Fonctionnalité	Action de test	Résultat attendu	Résultat observé
1	index.html / script.js	request()	5-13	Récupération de la liste des articles du back-end sous forme d'array puis envoie de chaque article à la fonction makeItem().	Ajout de la commande "console.log(data)" à la ligne 8 (juste après la réponse du serveur).	Récupération d'un array dans la console	OK
2	index.html / script.js	makeItem()	22-51	Crée le code html (avec lien unique) pour chaque article reçu en paramètre.	Simulation dans le navigateur.	Affichage des articles sur la page en nombre et qualité égale à ce qui a été reçu dans la console au test n°1.	OK
3	index.html / script.js	showError()	55-63	En cas d'échec de la requête, affichage d'un message en lieu et place des articles sur la page d'accueil pour informer qu'il n'y a aucun article disponible.	Arrêt du serveur de back-end et rafraîchissement de la page. Ajout de la commande "console.log(e)" dans la fonction de callback de l'élément catch.	Les articles doivent être remplacés par un message.	OK
4	product.html / product.js		4-6	Récupération de l'id de l'article passé dans l'url pour pouvoir le traiter.	Ajout de la commande "console.log(id)" à la ligne 6.	L'id reçu dans la console doit être le même que celui présent dans l'url.	OK
5	product.html / product.js	request()	15-28	Récupération de la liste des articles du back-end, s'il existe un article dont l'id est identique à celui récupéré dans l'url, on l'envoie en paramètre à la fonction makeItem().	Ajout de la commande "console.log(response)" à la ligne 19.	On doit récupérer le status 200 dans la console.	OK
6	product.html / product.js	makeItem()	38-62	Crée et implémente le code html pour l'article dont l'id a été récupéré dans l'url.	Simulation dans le navigateur.	Affichage des détails de l'article avec implémentation de l'élément select contenant les couleurs disponibles.	OK
7	product.html / product.js	buttSubmit.addEventListener	63-86	Affiche une bordure rouge autours des champs mal renseignés	Entrer des mauvaises valeurs dans le premier champs, puis dans le deuxième, puis dans les deux et enfin dans aucun.	La bordure rouge doit apparaître uniquement autours des champs mal renseignés. Les autres doivent récupérer le style initial.	OK
8	product.html / product.js		89-100	Création d'un objet correspondant au produit choisi.	Ajout de la commande "console.log(product)" à la ligne 100.	Récupération d'un objet dans la console contenant les caractéristiques du produit.	OK
9	product.html / product.js		100-129	Ajout du produit dans le localStorage et affichage d'un message confirmant cet ajout.	Ajout de la commande "console.log(productInLocalStorage)" à la ligne 126. Ensuite ajouter un produit au panier, ajouter une 2e fois le même produit, ajouter le même produit mais dans une couleur différente et enfin ajouter un autre produit.	De nouveaux objets doivent apparaître pour chaque produit de modèle et couleur différente. Dans le cas d'un ajout d'un produit de même modèle et couleur qu'un produit existant, seule la quantité doit être mise à jour. A chaque ajout de produit un message de confirmation doit apparaître à l'écran.	OK
10	product.html / product.js	applyStyle(args)	139-145	Applique différents styles aux 2 input de la page suivant si les données saisies sont validées ou non.	Simulation dans le navigateur. On teste toutes les combinaisons possibles de bonnes et mauvaises valeurs.	Dans chaque cas, quand l'input a une valeur invalide il prend une bordure rouge et quand il a une valide valide il reprend sa bordure noire.	OK
11	product.html / product.js	error()	149-159	En cas d'échec de la requête, remplacement de l'élément article (dans lequel il devrait y avoir les détails du produit) par un élément paragraphe contenant un message d'erreur.	Arrêt du serveur de back-end et rafraîchissement de la page.	Le produit doit être remplacé par un message.	OK
12	cart.html / cart.js		21-88	Si le localStorage n'est pas vide, crée le code html des articles à placer dans la page panier.	Ajouter plusieurs articles au panier, y compris des articles identiques mais de couleur différente. Cet ajout ne doit pas être fait à la suite. Il faut d'abord ajouter un autre article avant d'ajouter un modèle déjà existant dans une couleur différente (ceci pour tester le rangement par ordre alphabétique).	Tous les articles doivent être présents sur la page panier. Les articles doivent apparaître triés par ordre alphabétique.	OK
13	cart.html / cart.js		89-97	Si le localStorage est vide, affichage d'un élément paragraphe informant l'utilisateur que le panier est vide et ajustement du nombre d'articles et du prix total.	Vidage du localStorage et rafraîchissement de la page.	Affichage du message "votre panier est vide pour le moment" et remis à 0 du nombre d'articles et du prix total.	OK
14	cart.html / cart.js		100-118	Récupération du clic sur chaque input et traitement de la nouvelle valeur.	Modification de la quantité d'un article. Ajout de la commande "console.log("Quantité nulle")" à la ligne 115 lorsque le nombre d'article tombe à 0.	Si le nombre d'article tombe à 0, récupération du message "Quantité nulle" dans la console. Sinon, mise à jour de la quantité de l'article concerné dans le localStorage.	OK
15	cart.html / cart.js	getId(e)	124-127	Récupération de l'id du produit, passé en data de l'élément article parent, dont la valeur de l'input a changé.	Ajout de la commande "console.log(idToFind)" à la ligne 125.	Récupération de l'id dans la console et vérification qu'il s'agit bien d'un id de produit et que c'est le bon.	OK
16	cart.html / cart.js	getColor(e)	133-136	Récupération de la couleur du produit, passée en data de l'élément article parent, dont la valeur de l'input a changé.	Ajout de la commande "console.log(colorToFind)" à la ligne 134.	Récupération de la couleur dans la console et vérification qu'il s'agit bien d'une couleur de produit et que c'est la bonne.	OK
17	cart.html / cart.js	removeItem(x, y)	142-152	Suppression d'un article du panier.	Suppression d'articles directement dans le navigateur.	La suppression doit être effective si on valide la message d'alerte. Si on annuler le message et si ce message est apparu suite à une saisie de la valeur 0 dans l'input, ce dernier repasse à une valeur de 1.	OK

18	cart.html / cart.js	adjustQuantityAndPrice()	156-169	Calcule la quantité d'articles et le prix total des articles présents dans le localStorage et implémente les éléments html associés.	Test directement dans le navigateur et ajoutant des articles puis en modifiant les quantités sur la page panier.	Les quantités et prox s'ajustent automatiquement au clic sur les flèches des input ou lorsque les input perdent le focus.	OK
19	cart.html / cart.js	eventListeners des différents input	171-245	Récupération des modifications dans les champs du formulaire et modification du style des input si ce qui est entré ne correspond pas à ce qui est attendu. Ajout également d'un message sous les input en cas de mauvaise saisie.	Tests directement dans le navigateur. On essaie d'entrer des caractères spéciaux, des lettres dans le champs code postal, autre chose qu'un adresse e-mail dans le champs mail.	Les bordures des inputs doivent s'épaissir et devenir rouges en cas de mauvaise saisie. Un texte doit également apparaître sous l'input. Après une bonne saisie, le texte sous l'input doit disparaître et le style initial de l'input doit être réappliqué.	OK
20	cart.html / cart.js	subButton.addEventListener('click')	248-258	Récupération du clic sur le bouton "Commander !". Vérification que tous les champs sont bien renseignés et envoie vers la fonction suivante. Si un des champs est mal renseigné, affichage d'une alerte.	Tests directement dans le navigateur. On clique sur le bouton sans avoir rempli les champs. On essaie plusieurs combinaisons de champs bien et mal remplis. Tous les champs doivent avoir été testé comme unique champ invalide. On fini en renseignant correctement tous les champs.	Une alerte doit apparaître si au moins un des champs est mal renseigné et le code doit cesser de s'exécuter. Dès que tous les champs sont correctement renseignés, le clic mènera à la page de confirmation.	OK
21	cart.html / cart.js	requetePost()	262-290	Création de l'objet "order" contenant un objet de contacts et un array de produits. Construction de l'en-tête de la requête puis envoie au back-end. Récupération de l'orderId.	Ajout d'un breakpoint à la ligne 287. Test avec le server en service puis hors service.	L'utilisateur doit être redirigé vers la page de confirmation dont un id doit avoir été placé dans l'url. On doit retrouver l'orderId dans le débogueur ainsi que les bonnes valeurs pour order et en-tete. En cas d'échec, l'utilisateur doit voir apparaître un message et on doit retrouver le status de la requête dans la console.	OK
22	cart.html / cart.js	verificationFinale()	294-315	Vérification que le panier n'a pas été vidé puis implémentation de l'objet de contacts et l'array de produits à envoyer avec la requête POST.	Ajout de la commande "console.log(products)" à la ligne 304 et de la commande "console.log(contact)" à la ligne 311 (associés à un breakpoint à la ligne 312 pour ne pas changer de page) pour vérifier que tout a correctement été implémenté. Puis essaie de vider le localStorage.	Si le localStorage a été vidé une alerte doit apparaître pour prévenir l'utilisateur. Sinon on doit retrouver l'objet de contacts et l'array de produits dans la console.	OK
23	cart.html / cart.js	finalisation(orderId)	320-324	Effacement du localStorage et redirection vers la page de confirmation avec l'orderId passé dans l'url.	Pas de test particulier à mettre en place.	La page de confirmation doit avoir l'orderId dans l'url et le localStorage doit désormais être vide.	OK
24	cart.html / cart.js	verifNames(n), verifMail(adresse), verifAdd(v)	331-353	Vérification via des regex que ce qui est renseigné dans les inputs correspond bien à ce qui est attendu.	Utilisation du site regex101.com pour tester les regex et vérifier qu'elles correspondent aux besoins.	Les saisies qui ne correspondent pas à ce qui est attendu doivent être détéctées.	OK
25	confirmation.html / cart.js		2-8	Si le titre de la page appelant ce script est "confirmation" alors on récupère l'orderId passé dans l'url et on l'implémente dans le message de confirmation. Si ce n'est pas le cas on donne accès aux différentes fonctions de la page panier.	Tests directement dans le navigateur. On affiche les pages panier et confirmation.	Les pages doivent se comporter correctement.	OK

script.js	Nombre de lignes de code (hors sauts de ligne et commentaires) 46	Nombre de lignes de code testées 45	Pourcentage testé 97,83 %	Pourcentage total de ligne de code testées : **97,06 %**
products.js	Nombre de lignes de code (hors sauts de ligne et commentaires) 122	Nombre de lignes de code testées 118	Pourcentage testé 96,72 %	
cart.js	Nombre de lignes de code (hors sauts de ligne et commentaires) 297	Nombre de lignes de code testées 287	Pourcentage testé 96,63 %	