	Fonctionnalité	Action	Résultat attendu	Résultat observé
1	Récupérer tous les produits de façon dynamique.	Appel de l'API via la méthode GET	Recevoir un tableau d'objets de tous les produits	√
2	Récupération de la promesse et affichage de tous les produits dans leur cartes respectives.	Boucle "for...of" pour séparer chaque objet du tableau. Ajout de l'HTML pour chaque produit en y insérant les éléments correspondant à chaque critère de chaque produit (img / alt / description / name / id)	La page index affiche chaque produit dans une carte. Chaque carte est clicable.	√
3	Le clic d'une carte renvoie à sa propre page avec son id dans l'adresse. On récupère cet Id pour récupérer le produit en question.	Le résultat de SearchParams permet d'obtenir l'Id du produit que l'on cherche. On "concatène" l'adresse de l'API de tous les produits avec l'Id du produit trouvé pour obtenir l'adresse API du produit.	On obtient l'adresse de l'API du produit cliqué.	√
4	On récupère les données concernant le produit afin d'afficher ses contenus dans la page.	On peut alors faire un appel de l'API via la méthode GET pour récupérer les données propres au produit sous forme de tableau. On peut alors afficher chaque données dans le DOM	On obtient un tableau de données dont on se sert pour afficher dynamiquement le produit sur la page HTML (product.html)	√
5	On permet à l'utilisateur de choisir la couleur du produit.	Pour afficher l'ensemble des couleurs dans le panneau déroulant, on crée une boucle "for" qui va "scanner" chaque couleur inscrite dans la clé "couleur" de l'objet produit. Pour chaque couleur trouvée on l'affiche dans le DOM avec	On obtient un panneau déroulant proposant les couleurs disponibles pour le produit.	√
6	On permet à l'utilisateur de choisir la quantité désirée du produit.	On récupère la valeur de l'input productQuantity pour la stocker.	Nouvelle variable avec la valeur de l'input "quantité"	√
7	L'utilisateur est obligé de choisir une quantité minimum de 1 unité et d'un maximum de 100 unités ainsi qu'une couleur sinon il ne pourra rien ajouter au panier. On prévient l'utilisateur si sa sélection ne remplit pas les conditions ci-dessus.	On place un déclencheur d'évènement sur le bouton "ajouter au panier". A chaque fois que celui-ci sera appuyé, plusieurs actions seront effectuées. La première sera de vérifier si l'utilisateur a bien sélectionné la quantité et le produit. Avec une condition "if" on compare si la selection de la couleur est vide ou bien que la quantité est égale à zéro ou alors qu'elle est supérieure à 100. Si tel est le cas pour au moins une condition alors on envoie un message d'erreur à l'utilisateur afin qu'il corrige sa sélection.	Un message d'"erreur" s'affiche si l'utilisateur n'a pas rempli une des deux conditions. Si celles ci sont remplies, alors on poursuit le cheminement du code.	√
8	A l'appuie du bouton "ajouter au panier", on vérifie si le panier est vide ou non. Si il y a déjà quelque chose dans le panier alors on vérifie si le produit que l'ont veut ajouter dans le panier existe déjà dans celui ci.	Si les conditions ont été validés alors on crée une variable de type objet qui récupère toutes les données du produit. On vérifie que le localStorage est vide par une condition "if" qui vérifie si la variable "cart" est bien égale à zéro. Si c'est le cas, alors on lance la fonction "storeItemToCart" pour stocker la variable "cart" dans le localStorage. Si la variable "cart" n'est pas vide alors on récupère le contenu de la variable "cart" en "parsant" dans une nouvelle variable "itemsInCart". On utilisera la fonction map sur celle ci pour vérifier si le produit existe déjà dans le cart (en cherchant l'id et la couleur du produit).	On récupère les données du produit que l'on stocke dans une variable "item". Si le cart n'est pas vide alors on crée la variable itemsInCart et on y stocke ce qui est dans le cart. On crée alors une variable qui stockera le produit trouvé dans celle ci.	√
9	La quantité est mise à jour.	La quantité entrée par l'utilisateur s'ajoute à celle qui se trouve dans le cart. Pour cela, on vérifie grâce à une boucle "for" et une condition "if" si le produit existe déjà dans le cart. Si c'est le cas on met à jour la quantité en faisant le total de la valeur entrée par l'utilisateur et la valeur déjà stockée dans le cart.	On obtient alors la variable totalQuantity qui est l'addition de sameItemInCart.quantity + la valeur de l'input quantity de l'utilisateur.	√
10	Lorsque les vérifications sont faites, on ajoute le nouveau produit dans le panier.	La fonction storeItemToCart va alors "push" ou ajouter la variable "item" dans la variable "cart". On envoie alors ce nouveau tableau dans le localStorage en "stringifiant" son contenu pour qu'il soit compatible avec le format JSON.	Soit on crée un nouveau tableau que l'on stocke au format JSON dans le localStorage, soit on ajoute à ce même tableau un nouvel objet qui correspond au produit ajouté.	√
11	Lorsque le localStorage a été enregistré, on propose à l'utilisateur d'aller ou non sur la page "panier"	La fonction goToCart permet d'afficher une message de confirmation, si l'utilisateur clique sur "OK" alors on charge la page "cart.html".	Un message doit s'afficher pour donner le choix à l'utilisateur d'aller sur la page panier ou de rester sur la page courante.	√
12	Les produits stockés dans le panier doivent s'afficher sur la page cart.html	On récupère la variable cart depuis le local storage. Si le cart est vide, on affiche alors dans le DOM que le panier est vide. Sinon par une boucle "for" on affiche dans le DOM pour chaque objet du cart l'image du produit son text alternatif, son nom, sa quantité et sa couleur. Puisque le prix situé dans le localStorage n'est pas sécurisé, il est nécessaire de l'afficher depuis l'API. Pour cela, il s'agit de récupérer chaque Id depuis le cart et de faire une requête de type fetch avec la méthode GET pour récupérer le prix correspondant à chaque produit se trouvant dans le cart puis l'afficher dans le DOM.	On peut voir chaque produit sur la page HTML dans le cas où le panier n'est pas vide sinon un message signifiant à l'utilisateur que le panier est vide.	√
13	L'utilisateur peut changer la quantité voulue pour chaque produit.	On établit un déclencheur d'évènement sur l'input de quantité pour chaque produit en se servant d'une boucle 'forEach". On trouve dans le cart quel est le produit qui se rattache à l'input manipulé grâce à la fonction "findIndex". On met alors à jour la quantité rentrée via la variable "newQuantity" par l'utilisateur dans le cart.	La variable newQuantity doit comprendre les changements de l'input qui doivent s'afficher instantanément sur le DOM.	√
14	La quantité totale sera alors mise à jour ainsi que le prix total.	Pour calculer le total de quantité, il suffit de lister chaque produit contenu dans le cart grâce à une boucle "forEach" et additionner la quantité de chaque produit pour ensuite afficher le résultat dans le DOM. Pour le prix total, le procédé est le même. On liste chaque produit contenu dans le cart par une boucle "forEach". On additionne ensuite le prix du produit multiplié par la quantité. On affiche ensuite le résultat dans le DOM et on fait une recharge rapide de la page pour que les changements soient instantanément visibles.	Le total des quantité doit s'afficher dans le DOM.	√
15	L'utilisateur peut supprimer un article depuis le panier en appuyant sur le bouton "supprimer".	On établit un déclencheur d'évènement sur le bouton "supprimer" pour chaque produit en se servant d'une boule 'forEach". Avec la fonction "filter", on sépare le produit des autres et on enregistre le cart nouvellement modifié. On recharge la page pour metre à jour l'affichage.	Le prix total doit s'afficher dans le DOM.	√
16	L'utilisateur doit remplir un formulaire avant de passer commande. Chaque champs du formulaire est contraint à une vérification des caractères utilisés.	On définit l'input de chaque champs du formulaire avec une boucle "forEach". On y attribue un déclencheur d'évènement lié au changement. On définit le motif autorisé pour chaque champs. On vérifie alors que l'input correspond au motif déclaré précédemment via la fonction "match" et que l'input n'est pas vide. Pour tous les champs qui remplissent ces conditions on affiche un border vert sinon un border rouge avec un message d'erreur via le DOM.	Lors d'un changement d'un champs, un border vert ou rouge doit s'afficher ainsi qu'un message d'erreur.	√

17	L'utilisateur appuie sur le bouton "commander" pour lancer la commande. Il est alors vérifier que le formulaire est bien rempli et que le panier n'est pas vide. La commande est alors envoyée sur le serveur. Et l'utilisateur est redirigé vers la page de confirmation.	On établit un déclencheur d'évènement sur le bouton "commander" afin de vérifier par une condition si chaque champs a bien été rempli et si le cart n'est pas vide. Si tel est le cas alors on crée un objet "contact" qui contient les valeurs de chaque champs du formulaire et on ajoute celui ci à la variable "cart". Sinon on affiche un message d'erreur dans le DOM spécifiant à l'utilisateur qu'il n'a pas bien rempli le formulaire et/ou que le panier est vide. Si les conditions sont remplies alors on envoie une requête fetch via la méthode POST afin de recevoir une promesse contenant le numéro de commande. Une fois ce numéro obtenu on crée l'adresse url de la page confirmation en y ajoutant le numéro de commande. On efface le localStorage et on redirige l'utilisateur vers la page de confirmation.	l'API doit bien recevoir la requête dans le bon format et retourner le numéro de commande. Le localStorage doit s'effacer automatiquement. La page confirmation doit se charger. Si le formulaire n'est pas rempli ou si le panier est vide alors un message doit s'afficher sur la page.	√
18	Une fois sur la page de confirmation, l'utilisateur est notifié par un message lui confirmant que la commande a été passée comprenant le numéro de sa commande.	On récupère le numéro de commande situé dans l'adresse HTML avec la fonction searchParams. Et on l'affiche sur le DOM.	Le numéro de commande doit s'afficher correctement sur la page courante.	√