Plan de test

Fichier.js	Ligne	Fonction	Résultat attendu	Comment vérifier
timer.js	1 à 8	setTimeToTimer	Cette fonction a pour objectif de stocker les variables nécessaires aux fonctions checktimer et create Timer. Elle doit retourner 1 pour data.timer et un nombre pour data.now. Elle doit retourner un objet appelé setuptime pour data.SetupTime ou retourner null.	Appeler la fonction et vérifier les résultats dans un console.log
timer.js	11 à 15	chekTimer	Exécuter la fonction getTeddies si le résultat de data.SetupTime est null sinon si data.nowow - data SetupTime est supérieur au data timer exécuté la fonction getTeddies. Sinon exécuter la fonction displayData.	Modifier les fonctions appelées par des consoles log, essayer sans item dans le localstorage pour tester la condition null puis effectuer un localStorage.Set Item("setup Time", data now) et tester les autres conditions. Ne pas oublier de supprimer la clé après le test.
timer.js	18 à 30	createTimer	Créer la clé setupTime avec pour valeur data.now dans le localStorage si elle n'existe pas sinon si le timer est passé nettoyer le localStorage et recréer la clé setupTime.	Vérifier les conditions avec des consoles log puis vérifié que la clé setuptime est bien crée dans le localstorage si elle n'existe pas puis vérifier que le localstorage est bien nettoyé puis recréer si le timer est dépassé.
fetch.js	1 à 24	getTeddies	Fetch de l'api si la réponse est ok exécuter les fonctions createTimer, storeAPI, idStorage et displayData. Sinon renvoyer une erreur et exécuter la fonction displayFetchError	Remplacer les fonctions par des console.log et tester les conditions.
fetch.js	26 à 29	apiStorage	Récupèrer les données de l'API et les stocker dans le localStorage avec pour clé teddies	Vérifier que la fonction créer bien un array avec les données de l'api dans le localstorage
fetch.js	32 à 39	idStorage	Pour chaque item présent dans l'array de teddies, créer un array avec pour clé l'id de l'item en question	Vérifier que la fonction créer bien un array pour chaque ours présent dans teddies et qu'ils ont bien leur id en clé
index.js	1 à 30	displayData	Créer le code HTML avec des valeurs dynamiques et l'insèrer dans le document pour chaque ours présent dans la clé teddies	Vérifier que le code HTML créé contient les bonnes valeurs dynamiques
index.js	32 à 41	displayFetchError	Créer le code HTML à insérer en cas de problèmes avec l'api	Ne pas lancer l'api et vérifier que le code HTML crée et inséré soit bon
product.js	1 à 9	getTeddyData	Cette fonction a pour objectif de stocker les variables nécessaires aux fonctions présente dans le product.js. Elles permettent de récupérer l'id passé dans l'URL et de récupérer les données dans le localstorage avec cette id.	Appeler la fonction et vérifier les résultats dans des console.log que les data retournée sont les bonnes
product.js	11 à 46	createTeddyCard	Créer le code HTML avec les valeurs dynamiques récupérer et l'insérer dans le document	Vérifier que le code HTML créé contient les bonnes valeurs dynamiques
product.js	48 à 57	createColorList	Créer le code HTML avec les valeurs dynamiques récupérer et l'insérer dans le document	Vérifier que le code HTML créé contient les bonnes valeurs dynamiques
product.js	59 à 83	addToBasket	Ajoute un addeventlistener sur le bouton d'ajout au panier qui au clic vérifie qu'il y a bien une couleur et une quantité sélectionnée pour l'ajouter au panier	Tester sans quantité, ni couleur et vérifier l'alert. Tester avec quantité mais sans couleur et vérifier l'alert. Tester sans quantité mais avec couleur et vérifier l'alert. Tester avec quantité et avec couleur et vérifier l'alert.
product.js	87 à 123	addProductToLocalStorage	Récupère les données de l'ours ajouté au panier et les stocks dans un array. Si le localstorage ne contient pas de clé appelée basket la créer et y mettre l'array avec les données stockées. S'il existe et qu'il y a déjà un objet avec la même couleur et la même id mettre à jour la quantité sinon si c'est un nouvel ours l'ajouter à l'array.	Vérifier les différentes conditions avec des consoleselog et ensuite vérifier que les modifications apportées au local storage fonctionne comme voulu.
product.js	125 à 130	displayData	Regroupement des créations HTML en une fonction	
panier.js	1 à 5	getBasket	Si le localstorage ne contient pas basket alors apeller displayEmptyBasket sinon exécuter productInBasket et waitContentLoaded	Vérifier les conditions avec des console.log

panier.js	8 à 41	displayBasket	Récupère chaque produit disponible dans le local storage basket, pour chaque produit créer le HTML avec des valeurs dynamiques et l'insérer dans la page panier puis exécuter les fonctions updatevaluelocalstorage, deleteProduct et displayTotal	vérifier que le code HTML créé contient les bonnes valeurs dynamiques
panier.js	43 à 61	updateValueLocalStorage	Créer pour chaque input un event qui pour chaque changement de valeur modifie la quantité d'objets dans le local storage si elle est entre 1 et 99	vérifier avec des console.log
panier.js	63 à 85	deleteProduct	créer pour chaque croix un event qui au clic supprime le produit voulu dans le localstorage, si il ne reste plus aucun produit dans le basket le supprime	vérifier avec des console.log
panier.js	87 à 93	displayTotal	Créer le HTML et l'insérer dans le document pour afficher le prix total de la commande	vérifier la page html et vérifier les calcules
panier.js	95 à 101	updateHTM	Supprime le HTML crée et le recharge avec les modifications effectuées	modifier les quantités, supprimer les produits et vérifier que tout se recharge avec les modifications prises en compte
panier.js	104 à 111	displayEmptyBasket	Créer le code HTML et l'afficher si le panier est vide	vérifier la page avec un panier vide
panier.js	113 à 139	displayForm	Créer le formulaire pour valider la commande	Vérifier le formulaire sur la page de panier
panier.js	141 à 158	formEvent	Ajout d'un event sur le bouton du formulaire qui vérifie si les patterns sont respecté au clic	Vérifier les conditions avec des console.log et tester sur le site
panier.js	160 à 181	formValueToLocalStorage	Récupère les données dans le formulaire et les stocks dans un array	Vérifier avec des console.log
panier.js	168 à 194	order	Fetch POST de l'array crée au click du formulaire	Vérifier avec des console.log
panier.js	197 à 202	orderResponse	Récupère la réponse suite au post de l'api et le stock dans le local storage, supprime le basket item et nous renvoie sur la page de confirmation.	Vérifier la création des clé contact, orderID dans le localstorage. Vérifier la suppression de la clé basket. Vérifier qu'on est bien redirigé vers la page de confirmation
confirmation.js	1 à 23	displayConfirmation	Récupèrer les données dans le localstorage et les charger dans le document puis les supprimer du localstorage	Vérifier les conditions avec des console.log
confirmation.js	26 à 44	redirect	Redirige l'utilisateur en cas d'erreur ou d'actualisation de la page	Vérifier les conditions avec des console.log
carticon	1 à 12	displayNumberOfProduct	Affiche le nombre de produit sur l'icone du panier	Vérifier les conditions avec des console.log