Plan de test				
Fichier	**ligne de code**	**fonction testée**	**Résultat attendu**	**Comment vérifier le résultat attendu**
script.js	4	cartProduit = document.getElementById("items")	cibler dans le DOM un élément grâce à son ID	dans l'inspecteur, dans la console, on regarde le résultat de console.log(cartProduit)
script.js	13 - 29	fetch("http://localhost:3000/api/products") .then(response => response.json()) .then(products => { console.log(products); for(let product of products) { cartProduit.innerHTML += `<a href="${`product.html?id=${product._id}&name=${product.name}`}"> <article> <img src="${product.imageUrl}" alt="${product.altTxt}"> <h3 class="productName">${product.name}</h3> <p class="productDescription">${product.description}</p> </article> </a>` } })	un array contenant 8 objets. Chaque objet contient ces informations : imageUrl, altTxt, description, name, colors, price, id. Grâce à fetch et la response, dans la boucle for, avec innerHTML +=, on affiche chaque article avec ses informations détaillées.	dans l'onglet réseau de l'inspecteur, on regarde le statut de la requête " fetch("http://localhost:3000/api/products")". dans l'inspecteur, dans la console, on regarde le résultat de : console.log(products) et de console.log (products.name)

Plan de test				
Fichier	**ligne de code**	**fonction testée**	**Résultat attendu**	**Comment vérifier le résultat attendu**
products.js	6-9	const recuperationUrl = window.location.search const urlSearchParams = new URLSearchParams(recuperationUrl) const produitId = urlSearchParams.get("id")	récupérer l'id du produit pour l'intégrer à l'URL : ciblage de l'url et paramètrage avec UrlSearchParams puis extraction de l'Id	console.log(produitId)

products.js	14-29	fetch(`http://localhost:3000/api/products/${produitId}`)
		.then(response => response.json())
		.then(product => {
		document.querySelector(".item__img").innerHTML =
		`<img src="${product.imageUrl}" alt="${product.altTxt}">`
		// utilisation de la function insertionTxt (l96)
		insertionTxt("title", product.name)
		insertionTxt("price", product.price)
		insertionTxt("description", product.description)
		
		// création d'une boucle pour que chaque option de couleur soit affichée
		let option = product.colors
		for (let i = 0; i < option.length; i++) {
		insertionTxt("colors", `<option value="${option[i]}">${option[i]}</option>`)
		}
		});
			Grâce à fetch et à l'intégration de l'id du produit dans l'url, on récupère les informations du produit puis on insère les informations qui y sont relatives pour afficher le produit et ses détails. La boucle for permet d'ajouter l'option de couleur jusqu'à ce que toutes les options soient affichées	console.log(product.name);
				console.log(option)

products.js	34-91	document.getElementById("addToCart").addEventListener("click", (event) => {})	permet d'écouter les évènements au click de l'utilisateur en ciblant l'élément dans le DOM grâce à son ID addToCart	console.log(document.getElementById ("addToCart").addEventListener)
products.js	39	const { choixCouleur, choixQuantite } = selectValue();	Sélection des quantités et des couleurs choisies en utilisant la function "selectValue" (l90)	console.log(choixCouleur, choixQuantite)
products.js	45	let product = { id: produitId, couleur: choixCouleur, quantite: choixQuantite }	Déclaration d'un objet qui contient les informations utiles sur le produit sélectionné	console.log(product)
products.js	48	let stockProducts = JSON.parse(localStorage.getItem ("keyProducts"));	Déclaration d'une variable à portée globale pour obtenir les informations des produits enregistrés dans le localStorage (l88)	console.log(stockProducts)

products.js	45-52	if (stockProducts == null && product.quantite > 0 && product.quantite <= 100 && product.couleur != 0) {
		stockProducts = []
		stockProducts.push(product)
		alert(`${product.quantite} Kanap ${product.id} ${product.couleur} ont été enregistrés dans le panier`)
		localStorage.setItem("keyProducts", JSON.stringify (stockProducts))
		}
			Si le localStorage est vide ET que la quantité sélectionnée est inférieure ou égale à 100, ET qu'un choix de couleur a été fait, stockProducts est alors un tableau vide dans lequel on push les informations du produit puis on enregistre les produits dans le localStorage sous la key "keyProducts"	console.log(stockProducts)
				
				console.log(stockProducts.push(product))
				
				Onglet Appli/localStorage dans le localStorage pour vérifer que la key "keyProducts" existe

products.js	54-90	else if (stockProducts != null && product.quantite > 0 && product.quantite <= 100 && product.couleur != 0) {
		for (let j = 0; j < stockProducts.length; j++) {
		
		if (quantityAdded(j, product) > 0 && quantityAdded(j, product) <= 100 && stockProducts[j].id == product.id && stockProducts[j].couleur == product.couleur)
		{
		return (
		stockProducts[j].quantite = quantityAdded(j, product),
		localStorage.setItem("keyProducts", JSON.stringify(stockProducts)),
		alert(` Dans le panier, il y a ${stockProducts[j].quantite} Kanap ${product.id} ${product.couleur} actuellement enregistré(s). `)
		)
		}
		else if (quantityAdded(j, product) == 0 && quantityAdded(j, product) > 100 && stockProducts[j].id == product.id && stockProducts[j].couleur == product.couleur) {
		alert(` Dans le panier, il y a ${stockProducts[j].quantite} Kanap ${product.id} ${product.couleur} enregistré(s) or la quantité maximum est limitée à 100. Vous pouvez en ajouter ${100 - stockProducts[j].quantite} au panier.`)
		return false
		}
		}
		for (let k = 0; k < stockProducts.length; k++) {
		
		if (parseInt(product.quantite) > 0 && parseInt(product.quantite) <= 100 && stockProducts[k].id == product.id && stockProducts[k].couleur != product.couleur || stockProducts[k].id != product.id)
		{
		return (
		stockProducts.push(product),
		alert(` Dans le panier, il y a ${parseInt(product.quantite)} Kanap ${product.id} ${product.couleur} actuellement enregistré(s). `),
		localStorage.setItem("keyProducts", JSON.stringify(stockProducts))
		)
		}
		else if (quantityAdded(k, product) = 0 && quantityAdded(k, product) > 100 && stockProducts[k].id == product.id && stockProducts[k].couleur != product.couleur || stockProducts[k].id != product.id) {
		alert(` Dans le panier, il y a ${stockProducts[j].quantite} Kanap ${product.id} ${product.couleur} enregistré(s) or la quantité maximum est limitée à 100. Vous pouvez en ajouter ${100 - stockProducts[j].quantite} au panier.`)
		return false
		}
		}
			Autrement si le localStorage n'est pas vide et que la quantité sélectionnée est inférieure ou égale à 0, Et qu'un choix de couleur a été effectué(les boucles for permettent de répéter les étapes jusqu'à ce que le localStorage soit définitivement mis à jour) :
			- Si l'id et la couleur correspondent à un produit déjà enregistré dans le localstorage on retourne un localStorage dont la quantité de produit est mise à jour.
			- Si le produit sélectionné a le même id Et n'a pas la même couleur OU si l'id n'est pas le même, on va pusher le nouveau produit dans le tableau et l'envoyer dans le localStorage
			- Un message d'erreur apparaît à chaque contexte lorsque la quantité dépasse 100 produit.
				Onglet Appli/localStorage dans le localStorage pour vérifer que la key "keyProducts" existe.
				
				console.log(stockProducts)

Plan de test				
Fichier	**ligne de code**	**fonction testée**	**Résultat attendu**	**Comment vérifier le résultat attendu**
cart.js	8	let stockProducts = JSON.parse(localStorage.getItem("keyProducts"));	récupération des informations contenues dans la key "keyProducts" dans le localStorage	console.log(stockProducts)

cart.js	10-12	function setPanier(key, content) {
		localStorage.setItem(key, JSON.stringify(content))
		};
			Function pour envoyer les données au localStorage	dans l'inspecteur, dans l'onglet Appli, dans le localStorage, l'application de cette function aura créé une key ou aura mis à jour son contenu

cart.js	14-16	function supprimeKeys(key) {
		localStorage.removeItem(key)
		};
			lors de l'appel de cette function on supprime la key devenue inutile dans le localStorage	dans l'inspecteur, dans l'onglet Appli, dans le localStorage, l'application de cette function aura supprimé la key ciblée

cart.js	18-20	function insertionTxt(element, content) {
		document.getElementById(element).innerHTML = content
		}
			Function pour insérer du texte	insertionTxt('totalPrice', prixTotal). Si la fonction est correctement utilisée, on verra apparaître la valeur de prixTotal sur la page cart.html

cart.js	22-24	function txtAddEgal(element, content) {
		document.getElementById(element).innerHTML += content
		}
			Function pour insérer du texte avec l'opérateur += (additionne jusqu'à ce qu'il n'y ait plus de texte à ajouter)	Si cette fonction est bien utilisée, on verra s'afficher autant d'article qu'il en existe dans le localStorage sur la page cart.html

cart.js	28	let tableauQuantite = [];	Déclaration d'un tableau pour y insérer les quantités de chaque produit sélectionné	console.log(tableauQuantite)
cart.js	30	let tableauTotaux = [];	Déclaration d'un tableau pour y insérer les prix de chaque produit sélectionné	console.log(tableauTotaux)

cart.js	32-34	function reducer(accumulator, currentValue) {
		return accumulator + currentValue;
		}
			Function utilisée à deux reprises pour l'addition des quantités d'articles et l'addition des prix. Cela donne le résultat total de quantité et le prix total de la commande	console.log(prixTotal)

cart.js	36-42	function targetArticle(e) {
		const target = e.target.closest('article');
		let id = target.dataset.id;
		let color = target.dataset.color;
		return { id, color };
		}
			Function pour cibler l'article dans le html et récupérer l'id et la couleur associés à l'article pour savoir de quel produit il s'agit	console.log({id, color})

cart.js	45-48	function regexSame(regex) { return /^[A-Za-zàéèêëîïôöûüùç.-\s]{3,20}$/.test(regex) } function regexAddress(regexAddress) { return /^[a-zA-Z0-9\s,'-]{3,}$/.test(regexAddress) } function regexEmail(regexEmail) { return /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/.test(regexEmail) }	3 Function pour alléger le code, dont l'une peut être utilisée plusieurs fois (ex : prénom,nom,ville). Permet de contrôler les saisies des utilsateurs grâce aux regex.	console.log(regexSame()) renvoie true ou false

cart.js	52	const products = [];	Définition d'une constante pour créer un tableau d'Id dont on aura besoin pour obtenir l'orderId de la commande	console.log(products)
cart.js	59-70	if (stockProducts == null) { insertionTxt('cart__items', `<article class="cart__item"> <div class="cart__item__img"> <img src="/front/images/logo.png" alt="logo de Kanap"> </div> <div class="cart__item__content"> <div class="cart__item__content__description"> <p>Votre panier est vide !<p> </div> </div> </article>`); }	Si le localStorage est vide, afficher un texte pour le préciser en utilisant la function "insertionTxt".	si cette fonction est bien utilisée, on verra s'afficher un texte indiquant que le panier est vide.
cart.js	72-178	else if (stockProducts != null) { for (let i = 0; i < stockProducts.length; i++) {}	Si le localStorage n'est pas vide, afficher les produits qui y sont enregistrés et gérer les événements. La boucle for permet de répéter les opérations jusqu'à ce que tout soit à jour.	Si cette fonction est bien utilisée, les différents opérations seront prises en compte (affichage des produits enregistrés dans le localStorage notament)
cart.js	75-102	fetch(`http://localhost:3000/api/products/${stockProducts[i].id}`) .then(response => response.json()) .then(product => {}	Pour tous les produits enregistrés dans le localStorage on récupère les informations qui y sont relatives grâce à fetch et à l'intégration dans l'url de l'id du produit stocké. La réponse de la requête fecth permet d'afficher les spécificités de chaque produit	Si la requête fecth est bien réalisée, on peut le vérifier à son statut dans l'onglet réseau de l'inspecteur. console.log(stockProducts[i].id) console.log(product.price)

cart.js	81-102	txtAddEgal('cart__items', `<article class="cart__item" data-id="${stockProducts[i].id}" data-color="${stockProducts[i].couleur}"> <div class="cart__item__img"> <img src="${product.imageUrl}" alt="${product.altTxt}"> </div> <div class="cart__item__content"> <div class="cart__item__content__description"> <h2>${product.name}</h2> <p>couleur : ${stockProducts[i].couleur}</p> <p>prix : ${product.price} €</p> <!-- multiplication du prix par la quantité de produit --> </div> <div class="cart__item__content__settings"> <div class="cart__item__content__settings__quantity"> <p>Qté : </p> <input type="number" class="itemQuantity" name="itemQuantity" min="1" max="100" value="${stockProducts[i].quantite}"> </div> <div class="cart__item__content__settings__delete"> <p class="deleteItem">Supprimer</p> </div> </div> </div> </article>`);	Affichage des produits et de leurs spécificités	Si tout ce passe bien, il y a autant de produit affiché qu'en contient le localStorage
cart.js	105-109	tableauTotaux.push(parseInt(product.price) * parseInt(stockProducts[i].quan const prixTotal = tableauTotaux.reduce(reducer) insertionTxt('totalPrice', prixTotal)	Ce code permet la gestion de l'affichage du prix total de la commande : - pour chaque produit, on push son prix dans tableauTotaux - utilisation de la méthode reduce et la function "reducer" pour cumuler les prix - on affiche dans le html le prix total	console.log(tableauTotaux) console.log(prixTotal) Si tout va bien, le prix total doit être affiché sur la page cart.html
cart.js	112-114	tableauQuantite.push(parseInt(stockProducts[i].quantite)) const quantiteTotale = tableauQuantite.reduce(reducer) insertionTxt('totalQuantity', quantiteTotale)	Ce code permet la gestion de l'affichage de la quantité totale d'articles et utilise la même procédure que pour la gestion du prix total	console.log(tableauQuantite) console.log(quantiteTotale) Si tout va bien, la quantité totale d'article doit être affichée sur la page cart.html

cart.js	116-147	.then(() => { document.querySelectorAll('.itemQuantity').forEach ((inputQuantity) => { inputQuantity.addEventListener('change', (e) => { let { id, color } = targetArticle(e); let searchProduct = stockProducts.find((product) => product.id === id && product.couleur === color); if (searchProduct != undefined) { let quantite = inputQuantity.value let product = { id: id, couleur: color, quantite: quantite } for (let y = 0; y < stockProducts.length; y++) { if (product.quantite <= 100 && stockProducts[y].id == product.id && stockProducts[y].couleur == product.couleur) { stockProducts[y].quantite = product.quantite setPanier("keyProducts", (stockProducts)) }else if (product.quantite > 100 && stockProducts[y].id == product.id && stockProducts[y].couleur == product.couleur) { alert(`la quantité de kanap ${product.id} ${product. couleur}  est limitée à 100`) } } } // actualisation automatique des données location.reload() }) }) })	Ce code permet la gestion de la modification des quantités depuis la page panier. Pour modifier le produit, on utilise "change" dans addEventListener. A l'intérieur du addEventListener, on utilise de la function "targetArticle" pour cibler l'article et récupérer l'id/couleur associés. Puis on utilise "find" pour chercher le produit avec le même id et la même couleur dans notre localStorage. Puis on change la quantité du produit trouvé si elle est inférieure ou égale à 100. Puis on met à jour les quantités que l'on enregistre dans le localStorage. Pour finir, on demande à ce que les données soient automatiquement mises à jour sur la page html.	console.log({id, color) console.log(searchProduct) console.log(quantite) console.log(product) console.log(stockProducts) dans l'inspecteur, dans l'onglet Appli, dans LocalStorage, on peut voir que les quantités d'un produit changent lorsque l'utilisateur modifie la valeur de l'inputQuantity. Aussi, la quantité totale d'article et le prix total se mettent à jour automatiquement.

cart.js	148-173	.then(() => {
		document.querySelectorAll('.deleteItem').forEach((btnSupprimer) => {
		btnSupprimer.addEventListener('click', (e) => {
		let { id, color } = targetArticle(e)
		
		let finalProductList = stockProducts.filter((product) => product.id !== id || product.id === id && product.couleur !== color)
		
		if (finalProductList != undefined) {
		let stockProducts = finalProductList
		if (stockProducts.length != 0) {
		setPanier("keyProducts", (stockProducts))
		}
		else {
		supprimeKeys("keyProducts")
		}
		}
		location.reload()
		});
		})
		})
			Ce code sert à supprimer un produit du panier lorsque l'utilisateur clique sur le bouton supprimer.
			Pour supprimer le produit, on utilise le paramètre "click" dans addEventListener.
			On utilise la function "targetArticle" pour cibler l'article et récupérer l'id/couleur associés.
			Puis, on cherche grâce à "filter", un produit qui a un id différent OU un produit qui a un id correspondant et une couleur différente.
			Puis on enregistre dans le localStorage seulement les produits trouvés précédemment.
			Et si le localstorage n'est vide, on mettre à jour. S'il est vide, on supprime la clé "KeyProducts" puisqu'il n'y a plus de produit enregistré dedans.
			Pour finir, on actualise automatique des données
				console.log(btnSupprimer.
				addEventListener())
				console.log({id, color})
				console.log(finalProductList)
				console.log(stockProduct)
				
				dans l'inspecteur, dans l'onglet Appli, dans LocalStorage, on peut observer les articles être supprimés. S'il n'y a plus d'article, on voit que la key "keyProducts" n'existe plus.
				
				Les articles sont supprimés de la page html au click et les quantités et le prix total de la commande sont mis à jour.

cart.js	182	const btnCommander = document.getElementById("order")	Sélection du bouton "commander !"	console.log(btnCommander)
cart.js	184-313	btnCommander.addEventListener("click", (event) => { event.preventDefault();}	écoute des événements lorsque l'utilisateur clique sur le bouton "Commander !". Suppression des paramètres par défaut de la function grâce à event. preventDefault().	console.log(btnCommander. addEventListener)

cart.js	189-195	const contact = {
		firstName: document.getElementById("firstName").value,
		lastName: document.getElementById("lastName").value,
		address: document.getElementById("address").value,
		city: document.getElementById("city").value,
		email: document.getElementById("email").value
		}
			Création d'un objet "contact" pour contrôler, valider et enregistrer les valeurs du formulaire saisies par l'utilisateur	console.log(contact)
				console.log(contact.firstName)

cart.js	197-208	function controlePrenom() {
		if (regexSame(contact.firstName)) {
		insertionTxt('firstNameErrorMsg', ``)
		return true
		}
		else
		{
		insertionTxt('firstNameErrorMsg', `Les chiffres et les symboles ne sont pas autorisés. Le nombre de caractères doit être compris entre 3 et 20.`)
		return false
		}
		}
			Ce code sert à contrôler les données saisies par l'utilisateur, pour les input prénom:
			- si le champ est bien rempli (regex ok), ne pas mettre de texte
			- autrement, insérer un texte indiquant le champ à corriger
				console.log(regexSame(contact.
				firstName))
				renvoie true ou false selon les données saisies.
				console.log(controlePrenomNomCity)
				
				Si les données ne sont pas validées, un texte s'affichera sous l'input où il y a une erreur.

cart.js	210-221	function controleNom() {
		if (regexSame(contact.lastName)) {
		insertionTxt('lastNameErrorMsg', ``)
		return true
		}
		else
		{
		insertionTxt('lastNameErrorMsg', `Les chiffres et les symboles ne sont pas autorisés. Le nombre de caractères doit être compris entre 3 et 20.`)
		return false
		}
		}
			Ce code sert à contrôler les données saisies par l'utilisateur, pour les input nom:
			- si le champ est bien rempli (regex ok), ne pas mettre de texte
			- autrement, insérer un texte indiquant le champ à corriger	

cart.js	223-234	function controleCity() {
		if (regexSame(contact.city)) {
		insertionTxt('cityErrorMsg', ``)
		return true
		}
		else
		{
		insertionTxt('cityErrorMsg', `Les chiffres et les symboles ne sont pas autorisés. Le nombre de caractères doit être compris entre 3 et 20.`)
		return false
		}
		}
			Ce code sert à contrôler les données saisies par l'utilisateur, pour les input adresse :
			- si le champ est bien rempli (regex ok), ne pas mettre de texte
			- autrement, insérer un texte indiquant le champ à corriger	

cart.js	236-247	function controleAddress() {
		if (regexAddress(contact.address)) {
		insertionTxt('addressErrorMsg', ``)
		return true
		}
		else
		{
		insertionTxt('addressErrorMsg', `Les symboles ne sont pas autorisés. Le nombre de caractères est de 3 minimum.`)
		return false
		}
		}
			Ce code sert à contrôler les données saisies par l'utilisateur, pour l'input adresse postale.
			- si le champ est bien rempli (regex ok), ne pas mettre de texte
			- autrement, insérer un texte indiquant le champ à corriger
				console.log(regexAddress(contact.address))
				renvoie true ou false selon les données saisies.
				console.log(controleAddress)
				
				Si les données ne sont pas validées, un texte s'affichera sous l'input où il y a une erreur.

cart.js	249-260	function controleEmail() { if (regexEmail(contact.email)) { insertionTxt('emailErrorMsg', ``) return true } else { insertionTxt('emailErrorMsg',`Veuillez saisir une adresse email valide. `) return false } }	Ce code sert à contrôler les données saisies par l'utilisateur, pour l'input email. - si le champ est bien rempli (regex ok), ne pas mettre de texte - autrement, insérer un texte indiquant le champ à corriger	console.log(regexEmail(contact.email)) renvoie true ou false selon les données saisies. console.log(controleEmail) Si les données ne sont pas validées, un texte s'affichera sous l'input où il y a une erreur.

cart.js	263-269	if (controlePrenom() && controleNom() && controleCity() && controleAddress
		setPanier("contact", (contact))
		}
		else {
		alert('Tous les champs du formulaire doivent être correctement saisis.')
		return controlePrenom(), controleNom(), controleCity(), controleAddress
		}
			Ce code sert à valider les contrôles effectués précédemment.
			- Si les contrôles répondent aux exigences, valider les champs saisis et envoyer les données dans le localStorage dans la clé "contact".
			- Sinon, envoyer une alerte et retourner les contrôles pour que l'utilisateur corrige les champs qui n'ont pas été correctement saisis.
				console.log(controleEmail)
				console.log(contact)
				
				dans l'inspecteur, dans l'onglet Appli, dans LocalStorage, création d'une la key "contact" et envoi des données saisies de l'utilisateur dans le localStorage.
				En cas d'erreur, les saisies doivent être corrigées, un texte s'affiche sous l'input qui exige une correction.

cart.js	275-283	if (stockProducts != undefined) {
		for (let d = 0; d < stockProducts.length; d++) {
		products.push(stockProducts[d].id)
		}
		}
		else {
		alert("Votre panier est vide. Pour faire une commande, sélectionnez le K
		return false
		}
			Ce code sert à envoyer dans un tableau, les id des produits enregistrés dans le localStorage si celui-ci n'est pas vide. S'il est vide lors du click, envoyer un message d'erreur.	console.log(stockProducts)
				console.log(stockProducts[d].id)

cart.js	286	let orderProducts = { contact, products }	Ce code sert à créer un objet contenant un objet de contact et un tableau d'id de produit	console.log(orderProducts)

cart.js	288-312	if (orderProducts != undefined) {
		fetch(`http://localhost:3000/api/products/order`,
		{
		method: "POST",
		headers: {
		'Accept': 'application/JSON',
		'Content-Type': 'application/JSON'
		},
		body: JSON.stringify(orderProducts)
		})
		.then(response => response.json())
		.then((order) => {
		window.location.href = `confirmation.html?id=${order.orderId}`;
		supprimeKeys("keyProducts");
		})
		}
		else {
		alert("une erreur est survenue, veuillez réitérer votre commande")
		return false
		}
			Ce code sert à obtenir le numéro de commande, c'est à dire l'orderId, et à rediriger l'utilisateur vers la page confirmation.html dont l'url contient l'orderId.
			Si orderProducts contient les informations adéquates :
			-faire une requête fetch avec la méthode POST où l'on envoye les données contenues dans orderProducts au serveur
			- Ensuite on récupére l'orderId et on redirige l'utilisateur vers la page confirmation en intégrant "order.orderId" dans l'url et on supprime la key keyProducts
			
			Si l'orderProducts ne contient pas les informations adéquates, on alerte l'utilisateur et on return false.
				console.log(orderProducts)
				console.log(order.orderId)
				
				dans l'inspecteur, dans l'onglet Appli, dans LocalStorage, on peut oberser si la commande est validée, qu'il n'y a plus de keyProducts.
				Si la commande est validée, l'utilisateur est redirigé vers la page confirmation.html
				Si une erreur est survenue, une alerte indique à l'utilisateur de réitérer sa commande.

Plan de test				
Fichier	**ligne de code**	**fonction testée**	**Résultat attendu**	**Comment vérifier le résultat attendu**

confirmation.js	3-6	function supprimeKeys(key) {
		localStorage.removeItem(key)
		};
			Function pour effacer les données inutiles contenues dans le localStorage	Si la function est bien utilisée, la key ciblée devrait être supprimée du localStorage

confirmation.js	11-14	const recuperationUrl = window.location.search;
		const urlSearchParams = new URLSearchParams (recuperationUrl);
		
		const id = urlSearchParams.get("id");
			Ce code sert à récupération le numéro de commande c'est à dire l'id via l'url.
			On cible l'url puis utilise le paramétrage avec searchParams
			Puis on extrait l'Id
				console.log(id)

confirmation.js	17	document.getElementById('orderId').innerHTML = id;	Ce code sert à afficher le numéro de commande sur la page confirmation.html	le numéro doit être afficher sur la page html
confirmation.js	20	supprimeKeys("contact");	Suppression de la dernière key présente dans le local Storage.	dans le localStorage, la key "contact" a disparu.