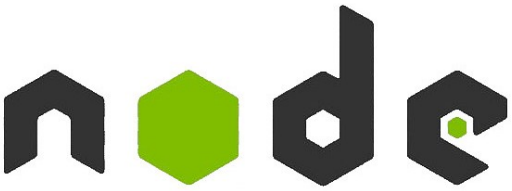
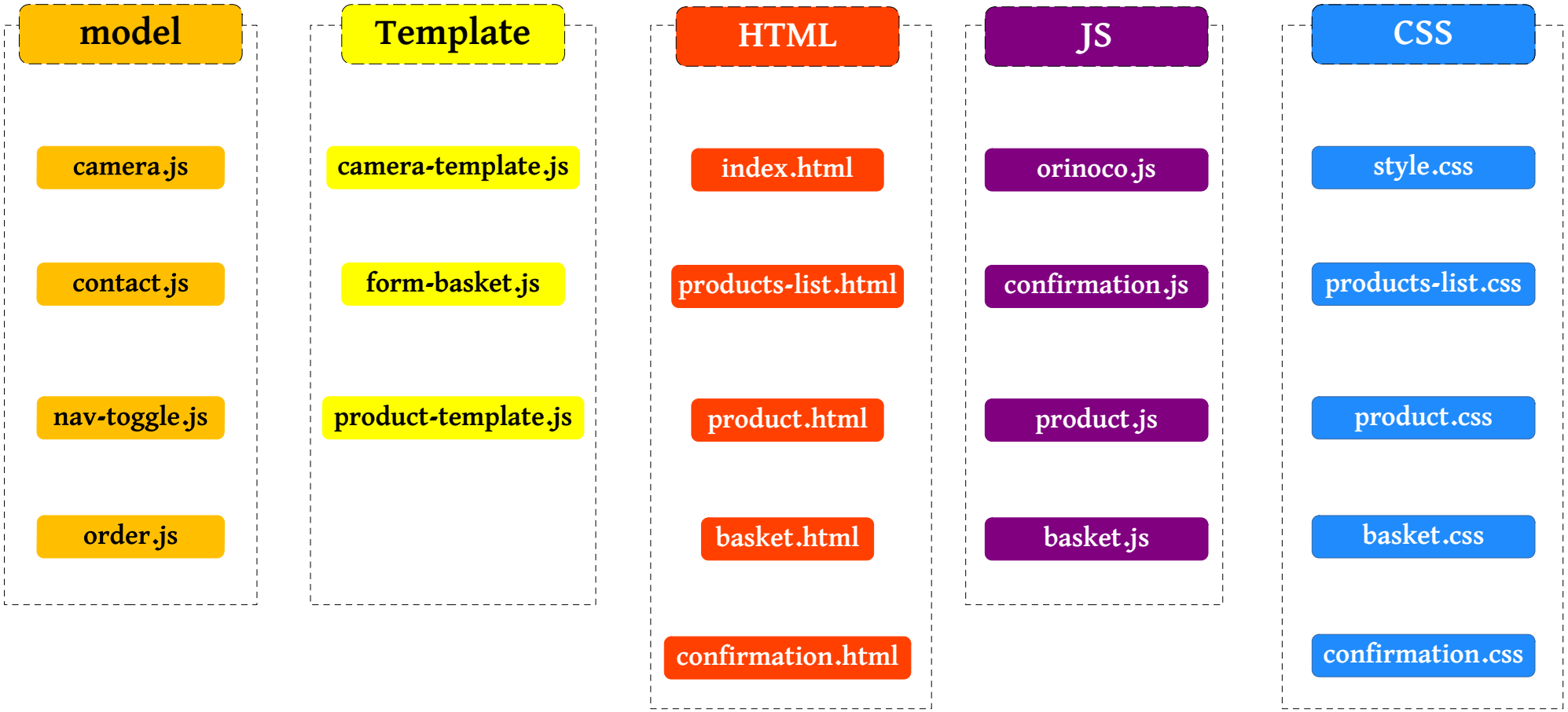
RAPPORT & PLAN DE TESTS

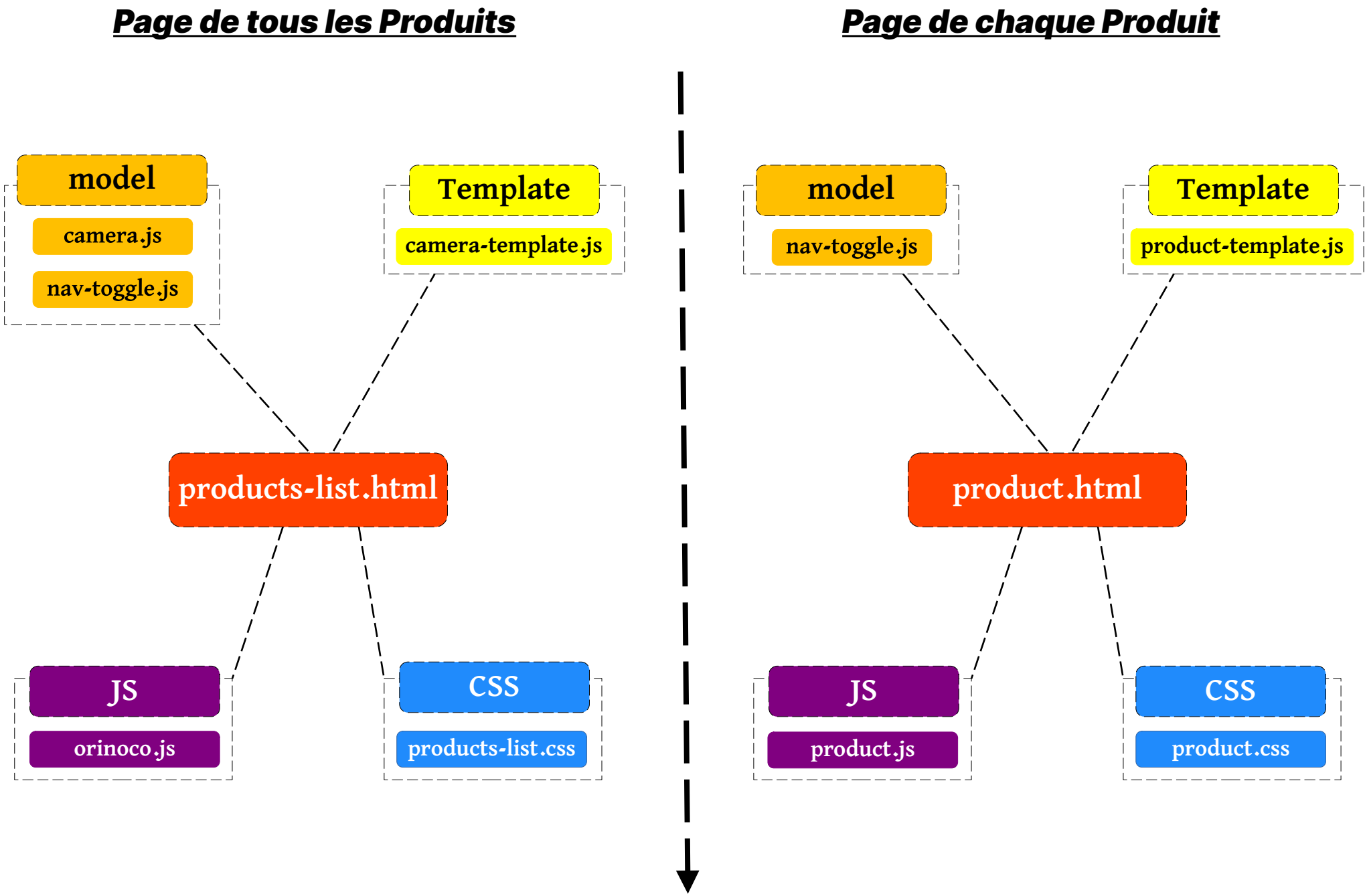


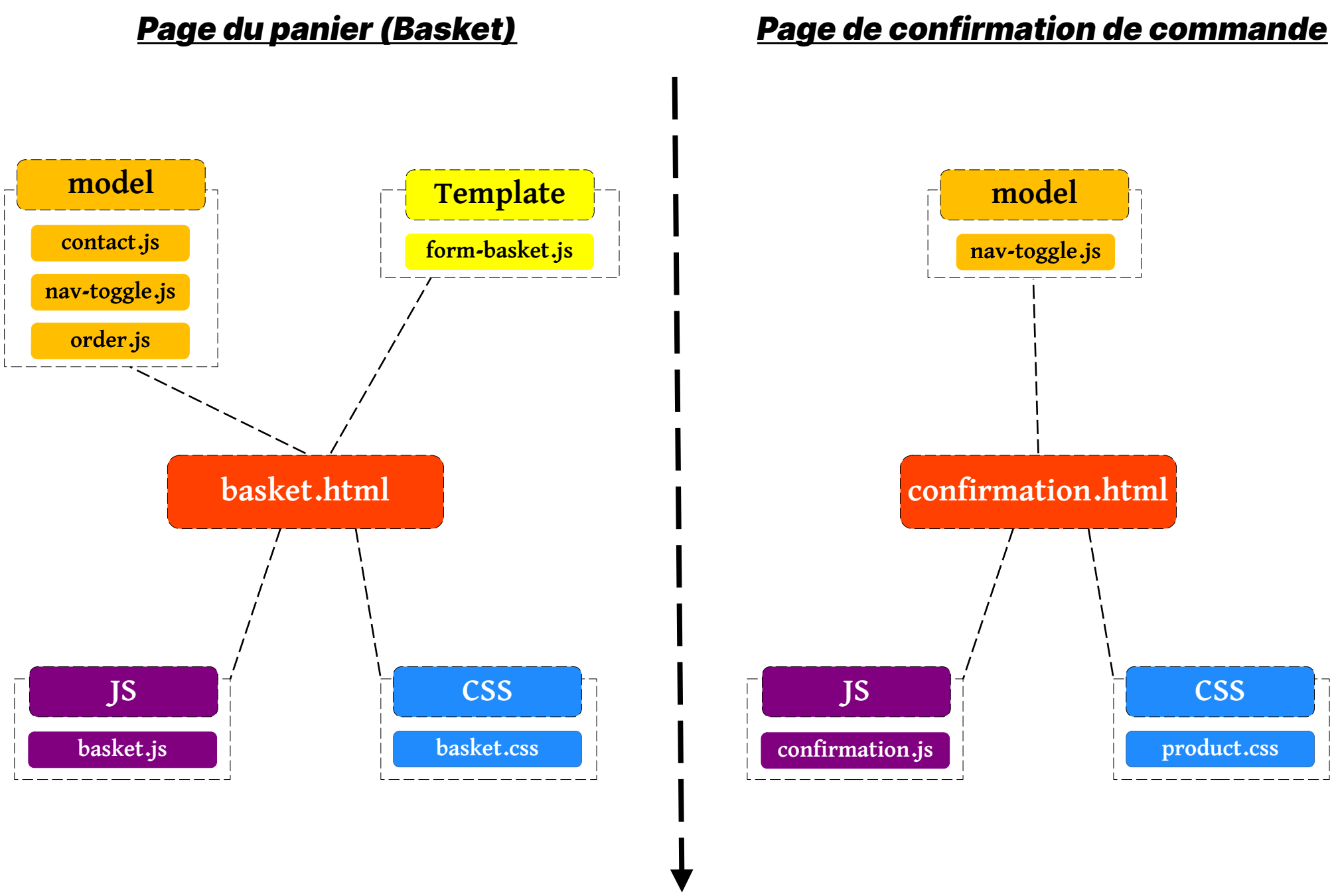
Orinoco

Préparé par
M. EL-KHABOU. E

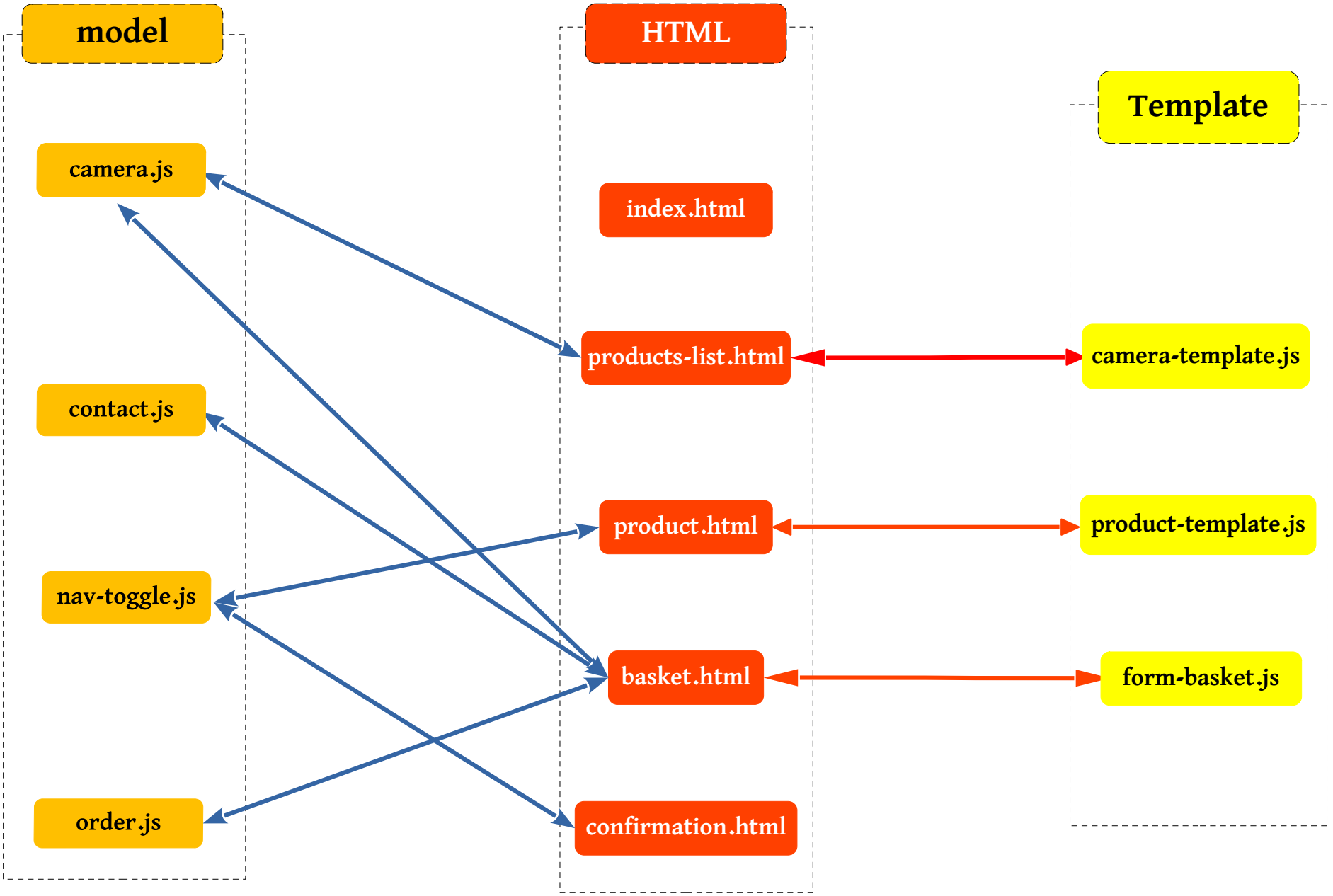
Comprendre la répartition de mes fichiers dans le Front-End




Page de tous les Produits
Page de chaque Produit
model
camera.js
nav-toggle.js
Template
camera-template.js
products-list.html
JS
orinoco.js
CSS
products-list.css
model
nav-toggle.js
Template
product-template.js
product.html
JS
product.js
CSS
product.css


Page du panier (Basket)
Page de confirmation de commande
model
contact.js
nav-toggle.js
order.js
Template
form-basket.js
basket.html
JS
basket.js
CSS
basket.css
model
nav-toggle.js
confirmation.html
JS
confirmation.js
CSS
product.css

Les relations entre HTML, Les fichier model et les Templates sans le JS



Model : camera.js

Nous avons créé une classe que nous appelons la classe « Camera » et qui permet de créer un objet (camera):

Quand nous récupérons la liste des Cameras de l'API (service), nous utiliserons cette classe ou cet objet créé pour qu'il s'affiche sur la page y effarante : par exemple, nous afficherons cet objet sur la page **product-list.html**.

Et, pour construire cet objet camera, on passe le produit comme paramètre au constructeur :

```
class Camera{
        constructor(product) {
                this.id = product._id;
                this.name = product.name;
                this.price =`${product.price / 100}.00 €`;
                this.description = product.description;
                this.imageUrl = product.imageUrl;
        }
}
```

Model : nav-toggle.js

Une fonction presque existante sur toutes les pages du site, puisqu'elle contrôle le menu de navigation au clic de ce dernier:

une nav-bar du menu qui contient les liens de navigation et qui, lorsque l'écran a une largeur maximale de 667 pixels, les liens de navigation se transforment en menu à outils dans une icône (bootstrap) :

```
function menuToggle() {
	document.getElementById("myDropdown").classList.toggle("show");
}

window.onclick = function(event) {
	if (!event.target.matches('.dropbtn')) {

		var dropdowns = document.getElementsByClassName("dropdown-content");
		var i;
		for (i = 0; i < dropdowns.length; i++) {
			var openDropdown = dropdowns[i];
			if (openDropdown.classList.contains('show')) {
				openDropdown.classList.remove('show');
			}
		}
	}
}
```

Model : order.js (commande)

Nous créons aussi La **classe « Order »** permet de créer l'objet order (commande)

Cet objet order doit contenir les données de contact du client, ainsi que Les produits commandés existants dans le LocalStorage :

```
class Order{
        constructor(contact, products) {
                this.contact = contact;
                this.products = products;
        }
}
```

Model : contact.js

Nous créons aussi La classe « Contact » pour nous permettre de créer l'objet (contact) :

Cet objet contact contient les données du client qui passe la commande sur le site, et qui doit contenir les informations ou les éléments suivants :

- Le prénom = (firstName)
- Le nom = (lastName)
- L'adresse = (address)
- Le code postale = (postCode)
- La ville = (city)
- L'email = (email)
- Les commentaires = (commentary)

Pour créer cet objet contact, nous passons ces paramètres au constructeur :

```
class Contact{
        constructor(firstName, lastName, address, city, postCode, tel, email, commentary) {
                this.firstName = firstName;
                this.lastName = lastName;
                this.address = address;
                this.postCode = postCode;
                this.city = city;
                this.tel = tel;
                this.email = email;
                this.commentary = commentary;
        }
}
```

Template : camera-template.js

La fonction **buildCamera** permet d'afficher une camera dans un card bootstrap.

Quand nous récupérons les produits de l'API, on affiche les produits sur la page d'affichage de produits (products-list.html) Grace à cette fonction **buildCamera**.

Nous appelons la fonction buildCamera qui prend en charge l'objet camera pour afficher chaque produit sur la page products-list.html :

```
        function buildCamera(camera){
                return `
                <div class="p-2">
                <a href="../pages/product.html?id=${camera.id}">
                        <div class="card" style="width: 18rem;">
                                <img id="cardImg" src="${camera.imageUrl}" alt="Card image cap">
                                <div class="card-body">
                                        <h5 id="cardTitle">${camera.name}</h5>
                                                <p id="cardDescription">${camera.description}</p>
                                                <p id="cardPrice" class="price">${camera.price}</p>
                                </div>
                        </div>
                </a>
                </div>`;
        }
```

Template : product-template.js

Nous créons aussi la Structure du produit dans product.html :

```
function buildCameraProduct(camera){
        return `
        <div class="infoProduct">
                <div class="product">
                        <img id="cardImgProduct" src="${camera.imageUrl}" alt="Card image">
                        <div class="cardBodyProduct">
                                <h5 id="cardTitleProduct">${camera.name}</h5>
                                        <p id="cardDescriptionProduct">${camera.description}</p>
                                        <p id="cardPriceProduct">${camera.price / 100}.00 €</p>
                                                <div class="choiceLenses" role="search" aria-label="Choix du produit">
                                                        <label class="font-weight-bold" tabindex="0">Choisir un optique:</label>
                                                                <select id="cameraLenses" class="lenses-product"></select>
                                                </div>
                                                <div id="quantity">
                                                        <span id="plus" onclick="addQuantity(event)"><i class="fas fa-plus-circle"></i></span>
                                                                <input type="number" id="plusOrMinusClick" value="1"min="1"/>
                                                        <span id="minus" onclick="minusQuantity(event)"><i class="fas fa-minus-circle"></i></span>
                                                </div>
                        </div>
                </div>
        </div>`;
        }

//Fonction pour le tableau "lenses" dans une balise <select>
function buildCameraLensesSelectList (lenses) {
        const optionList = document.querySelector("#cameraLenses").options;

        lenses.forEach(option =>
        optionList.add(
                new Option(option, option)
        )
        );
}
```

```
//-----------------------------Quantité-----------------------------//
function addQuantity (event){
        event.preventDefault();
        const quantityClicks = document.getElementById("plusOrMinusClick");
        quantityClicks.valueAsNumber++;
}

function minusQuantity (event){
        event.preventDefault();
        const quantityClicks = document.getElementById("plusOrMinusClick");

        if (quantityClicks.valueAsNumber == 1){
                return;
        }
        document
                .getElementById("plusOrMinusClick")
                .value = quantityClicks.valueAsNumber -1;
}
```

Template : form-basket.js

Nous construisons ce template afin de mettre en place l'architecture du formulaire que le client doit remplir avec ses informations personnelles avant qu'il passe la commande.

Ce formulaire se chargera aussi de la validité des données entrées par le client :

```
const form = () => {
        const positionForm = document.getElementById("form");
        const structureFormulaire = `
        <form name="orderForm" class="form-order" onsubmit="handleSubmit(event)" role="search" aria-label="Formulaire">
                <h4 class="formTitle">Merci de complèter ce formulaire pour valider la commande :</h4>
                        <div class="input-form">
                                <div class="name">
                                        <input type="text" id="lastName" name="lastName" class="contact" pattern="[A-Z][A-Za-z' -]+" title="Le format requis : lettres uniquement" placeholder="Nom" aria-labelledby="Nom" required>
                                        <input type="text" id="firstName" name="firstName" class="contact" pattern="[A-Z][A-Za-z' -]+" title="Le format requis : lettres uniquement" placeholder="Prénom" aria-labelledby="Prénom" required>
                                </div>
                                <div class="address">
                                        <input type="text" id="adress" name="adress" class="contact-address" minlenght="5" maxlenght="40" title="Le format requis : lettres et chiffres uniquement" placeholder="Adresse" aria-labelledby="Adresse" required>
                                </div>
                                <div class="cp">
                                        <input type="text" id="postCode" name="postCode" class="contact" pattern="([A-Z]+[A-Z]?\-)?[0-9]{1,2} ?[0-9]{3}" minlenght="5" title="Le format requis : 5 chiffres minimum uniquement" placeholder="Code Postale" aria-
                                        labelledby="Code Postale" required>
                                        <input type="text" id="city" name="city" class="contact" pattern="[A-Z][A-Za-z' -]+" title="Le format requis : lettres uniquement" placeholder="Ville" aria-labelledby="Ville" required>
                                </div>
                                <div class="tel-mail">
                                        <input type="tel" id="tel" name="tel" class="contact-tel" pattern="0[1-9][0-9]{8}" max-lenght="10" title="Le format requis : 10 chiffres maximum uniquement" placeholder="Téléphone" aria-labelledby="Téléphone" required>
                                        <input type="email" id="email" name="email" class="contact-mail" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$" title="Le format requis : exemple@mail.com" placeholder="E-mail" required>
                                </div>
                                <div class="commentary">
                                        <textarea type="text" id="commentary" name="commentary" class="commentary" maxlength="200" title="Informations complémentaires" placeholder="Informations complémentaires à la livraison" aria-labelledby="Informations
                                        complementaires"></textarea>
                                </div>
                        </div>
                <!--Bouton validé la commande-->
                <div class="btn-basket" role="button" aria-label="Bouton" tabindex="0">
                        <button type="submit" id="validOrder" class="button">Valider la commande</button>
                </div>
        </form>
        `;

        positionForm.insertAdjacentHTML("afterend", structureFormulaire);
        };
        form();
```

Plan de Tests

Fichiers	Lignes	Fonctions Testées	Résultat attendu	Comment vérifier le résultat attendu	Problème possible	Validé
orinoco.js	**Ligne 4 à 21**	Récupérer la liste des produit du serveur par l'API : `async function loadCameras() { const response = await fetch` appelée au début du fichier doivent s'afficher convenablement dans la page de listes des produits	La fonction asynchrone `fetch`()lance une requête et renvoie une promesse, mais puisqu'il y a `await`, la fonction asynchrone sera suspendue jusqu'à la finition de la demande pour faire l'affichage	catch(e) { console.log(e); }	En cas de problème de réseau par exemple, il se peut que la liste d'éléments récupérés et a afficher soit nulle.	Voir ✓
orinoco.js	**Ligne 31 à 33**	Fonction permettant d'afficher la liste de produit récupérée par la fonction précédente `fetch` du serveur avec une boucle forEach pour chaque élément: `cameras.forEach((camera) => { mainContainer.innerHTML += buildCamera(new Camera(camera)); })`	Tous les 5 produits récupérés par la fonction `async function loadCameras() { const response = await fetch` appelée au début du fichier doivent s'afficher convenablement dans la page de listes des produits	Se positionner dans le répertoire back-end et appeler le serveur avec la commande **node server,** puis démarrer la page **products-list.html**	Le nombre d'élément du DOM peut être Différent du nombre attendu, voir inexistant., surtout si le serveur est déconnecté ou rencontre un problème de communication	Voir ✓
Fichiers	**Lignes**	**Fonctions Testées**	**Résultat attendu**	**Comment vérifier le résultat attendu**	**Problème possible**	**Validé**
product.js	**Ligne 6 à 9**	PRODUIT SÉLECTIONNÉ PAR _id DE L'API : `function getCamera() { const cameraId = new URL (location.href) .searchParams.get("id"); const url = `http://localhost:3000/api/cameras/${cameraId}`; fetch(url)`	Récupère les données de la caméra sélectionnée avec l'API Fetch, / _id de la caméra : Création du lien pour afficher la caméra sélectionnée ; La caméra sélectionnée apparaît sur la Page	REF. au dossier backend/routes/camera.js -> router.get('/:id', cameraCtrl.getOneCam Era); Le lien fonctionne et nous envoie vers la caméra sélectionnée	« id » absent du « backend » ou objet Vide : Le lien ne fonctionne pas ou la caméra ne correspond pas avec celle qui a été sélectionnée	Voir ✓
product.js	**Ligne 22 à 26 Et Ligne 46 à 53**	Récupération des données de la caméra et injection du html depuis le js grâce au .innerHTML : `let mainContainerProduct = document.getElementById ('cardProduct'); mainContainerProduct.innerHTML = buildCameraProduct(selectedCamera); buildCameraLensesSelectList(selectedCamera.lenses);`	Chaque caméra a bien sa page avec ses propres données (nom, prix, image, options de lentilles et sa description)	console.log(camera);	Absence « key : value » La valeur retournée pourrait être indéfini, ou La valeur ajouté pourrait ne pas être prise en compte	Voir ✓
product.js	**Ligne 32 à 38 Et Ligne 54**	ÉCOUTE DU BOUTON ET AJOUT DE LA SÉLECTION AU PANIER : `const btn_addCart = document.querySelector("#add_product_to_basket");`	Récupération des données API correspondant a l''ID' dans une variable produit. L'objet CAMERA stocke les valeurs.	Tester le bouton Ajouter au panier ou appeler la fonction en omettant de Le produit pourrait ne pas être ajouté au sélectionner une option	Le produit pourrait ne pas être ajouté au sélectionner une option localstorage.	Voir ✓
product.js	**Ligne 39 à 40**	ÉCOUTE DE L'EVENEMENT « CLICK » AVEC ENVOI DU CHOIX A LA PAGE PANIER / LA PRISE EN COMPTE DU TYPE DE LENTILLE CHOISIE : `let selectLenses = document.getElementById("cameraLenses"); console.log('Lense selectionné == > ' + selectLenses.value);`	Affichage DE L'ALERTE : Vous Avez ajouté le produit dans Votre panier ! AVEC Affichage dans la page HTML du menu déroulant des lentilles	console.log('Lense selectionné == > ' + selectLenses.value);	Aucune lentille n'a été sélectionnée (alert), la lentille ne correspond pas, ou la page ne s'ouvre pas ou les données ne correspondent pas ou la caméra n'apparaît pas	Voir ✓

Fichiers	Lignes	Fonctions Testées	Résultat attendu	Comment vérifier le résultat attendu	Problème possible	Validé
basket.js	**Ligne 8 à 27**	Récupération des données du panier du Local Storage et les dispatcher dans le HTML **Const basketRecovery = JSON.parse(localStorage.getItem('myBasket'));**	La fonction récupère le dernier élément du local storage et regroupe les mêmes cameras par leur ID, puis les affiche dans le Panier.html (basket)	Il faut carrément choisir deux ou trois produit et voir si nous les retrouvons sur la page panier (basket.html)	Les informations sont erronées ou le tableau est vide	Voir ✅
basket.js	**Ligne 29 à 31**	Affichage du prix total de la commande dans le panier : **const totalPriceDisplay = `** **<div id="total-price" class="total-cart-price"> Le prix total de vos achats est de : ${totalPrice}.00 €</div> `**	Le prix total de la commande doit s'afficher sur la page HTML du Panier (Basket)	il faut jouer sur la quantité en choisissant un produit, et voir si le total se calcule et s'affiche bien sur la page du panier	La valeur retournée pourrait ne pas correspondre au total attendu	Voir ✅
basket.js	**Ligne 34 à 39**	insertion des produits récupérés du local storage avec ajout du prix total : **basketDisplay.innerHTML = structureProductBasket;** **divTotalPrice.innerHTML = totalPriceDisplay;** **localStorage.setItem('totalCommand', totalPrice);** **}**	Les produits choisis ainsi que le prix total de ces produits doivent tous s'afficher sur la page du panier	il faut choisir différents produits (minimum 2) avec des quantités supérieures à 2 et avec différentes lentilles pour voir si le tout s'affiche bien sur la page du panier	La fonction pourrait ne rien récupérer du local Storage et la valeur retourné serait nulle	Voir ✅
basket.js	**Ligne 44 à 47**	Fonction pour supprimer l'article de la commande, avec affichage d'un message de confirmation de suppression de l'article : **function deleteItem(event, itemId) {** **event.preventDefault();** **let choice = confirm("Voulez-vous retirer l'article du panier ?");**	La fonction permet de vider le panier , afficher un message 'votre panier est vide ' et redirection vers la page d'accueil, et si le panier contient tjrs des items sélectionnés, il faut un reload du panier pour afficher ces derniers.	Il faut Tester l'écouteur d'événement	Le panier pourrait ne pas se vider	Voir ✅
basket.js	**Ligne 60 à 66**	Fonction pour obtenir un Id de la commande : **function getOrderId(responseId) {** **let orderId = responseId.orderId;** **console.log(orderId);**	Fonction pour obtenir un Id de commande / au clic du bouton, on arrive sur la page de confirmation affichant un numéro de commande aléatoire	Modifier les valeurs de totalCommande() puis la valeur de la variable numeroCde et appeler la fonction	La valeur de retour pourrait ne pas être prise en compte	Voir ✅
basket.js	**Ligne 69 à 116**	Fonction pour gérer la soumission du formulaire : Récupération des données saisie dans le formulaire, récupère les id des produits et créer le contact = créer la commande avec le contact de cette dernière : **async function sendCommandToServer(order) {** **const config = { method: 'POST', headers: { 'Accept': 'application/json', 'Content-Type': 'application/json', },** **body: JSON.stringify(order)** Puis **eturn await response.json();**	Acceptation des caractères saisies dans le formulaire, et redirection vers la page de confirmation avec un message de validation, et un numéro de commande.	Jouer sur plusieur type de quantité, de caméra, de lentille et de contact, pour voir la différence dans l'affichage du résultat.	"data" inexistante.	Voir ✅
confirmation.js	**Ligne 02 à 29**	La fonction permet d'afficher l'identifiant et le total de la commande tout en supprimant le contenue du local storage	Affichage du message de confirmation de la commande, avec un numéro de commande, et le total de cette dernière, tout en supprimant le panier, c'est a dire le local storage	c'est simple, il faut juste passer commande	La valeur retournée pourrait ne pas correspondre au résultat attendu	Voir ✅