

# P5\_orinoco\_gruner\_audrey

Créer un site e-commerce



Soutenance du projet P5Openclassroom  
Créez un site E-Commerce



**AUDREY GRUNER**  
Développeuse web freelance



**AUDREY GRUNER**

Développeuse web freelance

## AGENDA REUNION

- **Introduction : objectif de la réunion**
- **Partie 1 : Installation Node**
  - Connexion au serveur de l'API / lignes de commande utilisées
- **Partie 2 : Présentation du site - navigation**
  - Structure HTML – liens - navigation
  - Bootstrap
- **Partie 3 : Evenements et fonctions Javascript**
  - ProductList.js : appencChild ...asynchrone – synchrone - init
  - ProductDetails.js : fonc ajouter panier etc...
  - Panier.js
  - Confirmation.js :
- **Partie 4 : Plan de Tests**
  - Présentation du plan de Tests
- **Partie 5 : Compétences validées**





**AUDREY GRUNER**

Développeuse web freelance

---

## **Lien Dossier GitHub**

<https://github.com/grunera/orinocoESHOP>

## **Lien Page web site Orinoco (GitHub)**

<https://grunera.github.io/orinocoESHOP/>

P5\_orinoco\_gruner\_audrey

# INTRODUCTION



# Présentation :

**Orinoco** est

un site boutique  
e-commerce  
de ventes de peluches.



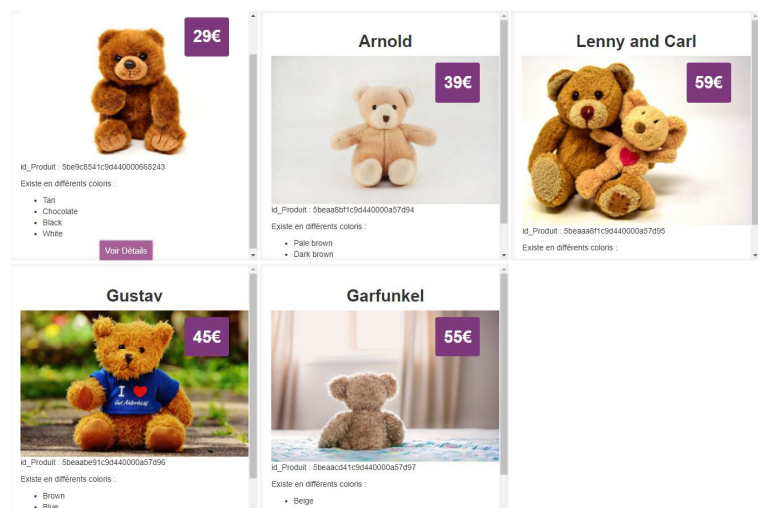
Bienvenue sur Orinoco!  
Découvrez un univers de tendresse et de douceur. Orinoco a sélectionné pour vous le meilleur du **nounours en peluche**!  
Que vous soyez plutôt grosse peluche ou bien **petit nounours** dou dou, Orinoco répondra à toutes vos attentes grâce à une merveilleuse et impressionnante collection.

# Description :

Propose une gamme de 6 produits Teddies avec déclinaison de couleurs. L'utilisateur peut les ajouter à un panier, valide sa commande et reçoit une confirmation de commande.

☑ **Point technique :** les produits et leurs caractéristiques sont stockés sur une API :  
<http://localhost:3000/api/teddies>

*Découvrez nos plus belles peluches !*



P5\_orinoco\_gruner\_audrey

## PARTIE I

### Connexion au serveur de l'api

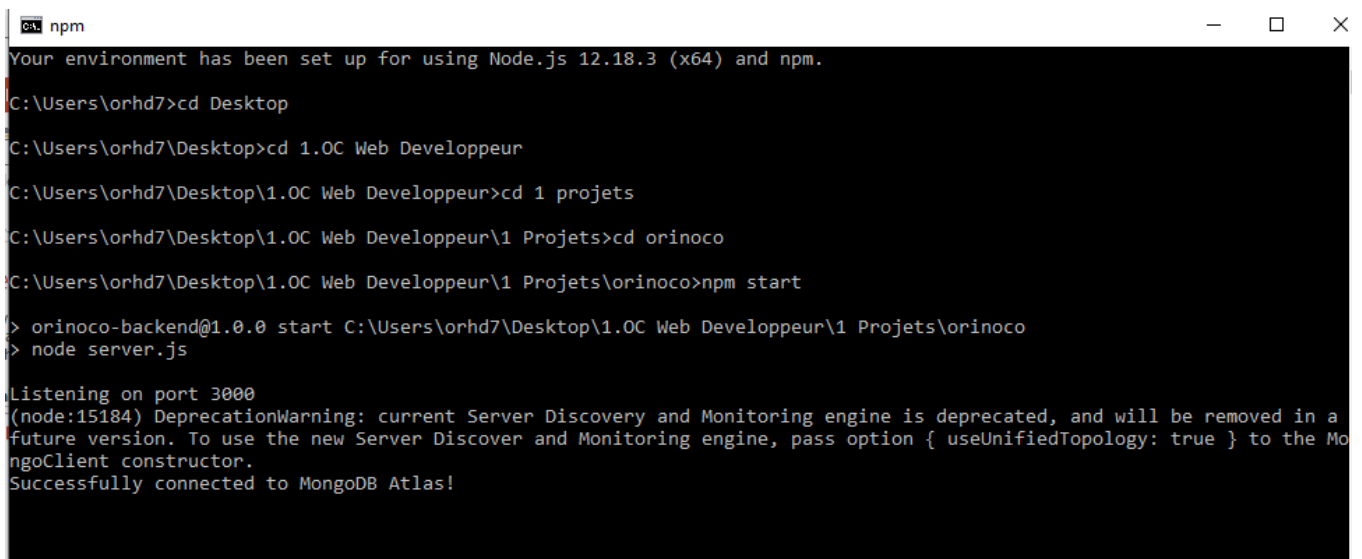


## PARTIE I \_CONNEXION AU SERVEUR DE L'API

---

- ➔ Start de NPM fourni par NODE.js et connexion au serveur Mongo DB Atlas à l'adresse de l'API des TEDDIES.

**<http://localhost:3000/api/teddies>**



```
npm
Your environment has been set up for using Node.js 12.18.3 (x64) and npm.

C:\Users\orhd7>cd Desktop
C:\Users\orhd7\Desktop>cd 1.OC Web Developpeur
C:\Users\orhd7\Desktop\1.OC Web Developpeur>cd 1 projets
C:\Users\orhd7\Desktop\1.OC Web Developpeur\1 Projets>cd orinoco
C:\Users\orhd7\Desktop\1.OC Web Developpeur\1 Projets\orinoco>npm start

> orinoco-backend@1.0.0 start C:\Users\orhd7\Desktop\1.OC Web Developpeur\1 Projets\orinoco
> node server.js

Listening on port 3000
(node:15184) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
Successfully connected to MongoDB Atlas!
```

- ➔ **NB: quelques définitions :**
- ➔ **Node. js** est une plateforme de développement Javascript. Ce n'est pas un **serveur**, c'est juste le langage Javascript avec des bibliothèques permettant de réaliser des actions comme ouvrir/fermer des connexions réseau ou encore créer un fichier....
- ➔ **NPM est un gestionnaire de paquets** (Package Manager), programme permettant d'installer des modules pour Javascript. Un module résoud des problématiques communes, ces modules sont publiés dans un gestionnaire de paquets ici, NPM.
- ➔ **MongoDB Atlas:** Sous le nom d'**Atlas**, MongoDB est une base de données open source/ cloud, concurrent de Amazon Web Services. **Il est basé sur le serveur NoSQL open source MongoDB**

P5\_orinoco\_gruner\_audrey

## PARTIE II

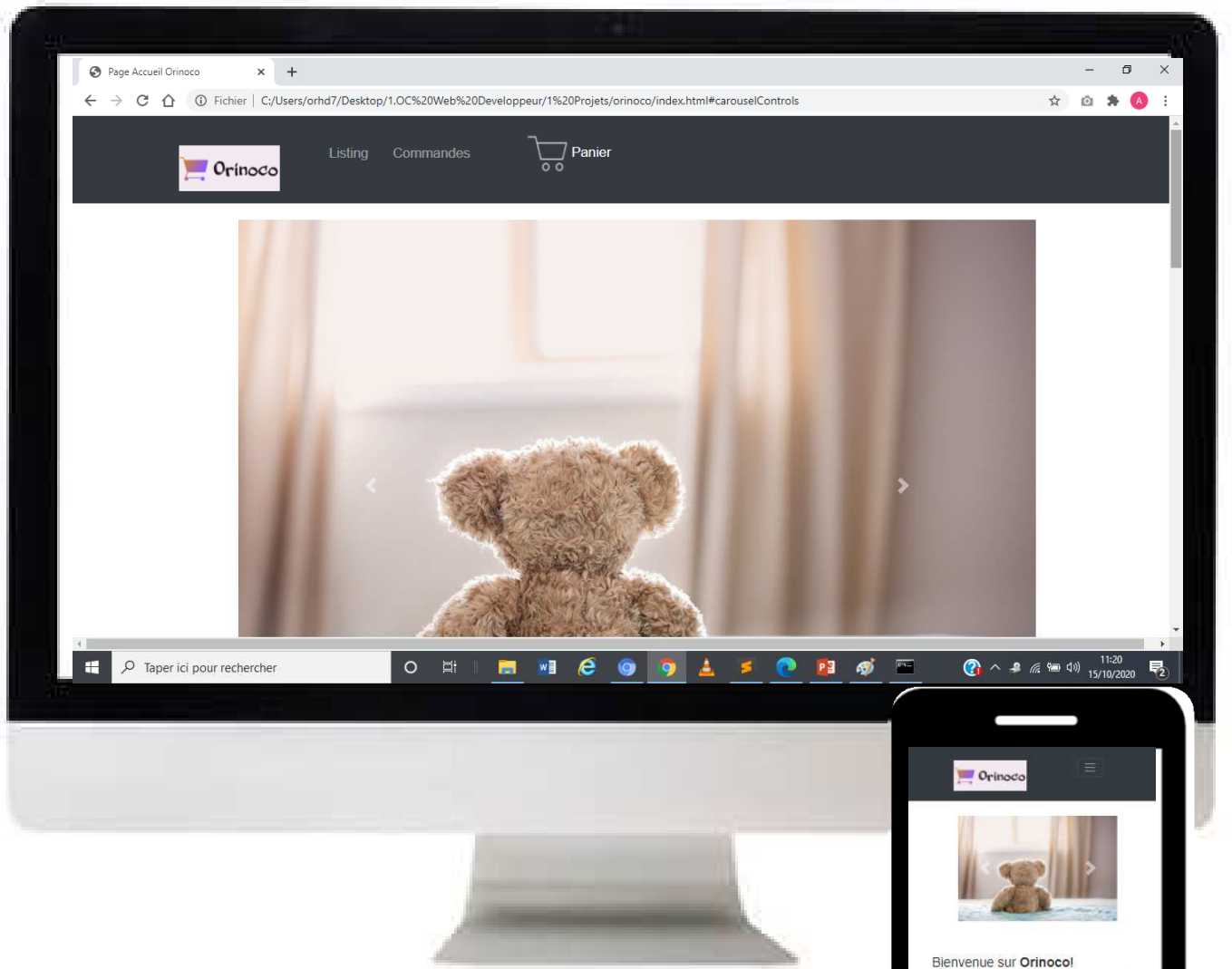
### Présentation du site Orinoco





## PARTIE II \_PRESENTATION DU SITE DESKTOP & RESPONSIVE

### Visualisation Desktop Navigateur Chrome



### Visualisation Responsive Galaxy S5







Site développé avec  
le framework  
**BOOTSTRAP**

## PARTIE II \_STRUCTURE DE NAVIGATION HTML DU SITE

Carrousel  
Bootstrap

Page Panier.html

Page d'accueil = Listing

Votre panier			
Article	Prix	Image	Couleur
Norbert	29€		Tan
Arnold	39€		Pale brown
Lenny and Carl	59€		Brown
Garfunkel	55€		Beige
Total : 182€			



### Validez votre commande

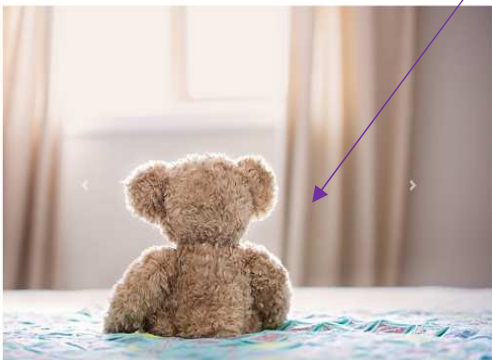
Vous y êtes presque!  
Remplissez le questionnaire et validez votre commande  
Vos Coordonnées

Nom   
Prénom   
Mail   
Confirmer mail

### Votre Adresse


Adresse   
Ville

 Listing 



Bienvenue sur Orinoco!  
Découvrez un univers de tendresse et de douceur. Orinoco a sélectionné pour vous le meilleur du **nounours en peluche**!  
Que vous soyez plutôt grosse peluche ou bien **petit nounours doux**, Orinoco répondra à toutes vos attentes grâce à une merveilleuse et impressionnante collection.


*Découvrez nos plus belles peluches !*



29€

id= "boutique"


Arnold



39€

id= "boutique"


Lenny and Carl



59€

id= "boutique"


Gustav



45€

id= "boutique"

Garfunkel



55€

id= "boutique"

Bouton

« voir  
détails »

Page produitDetails.html

 Listing 

### PELUCHE Norbert

29€

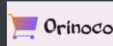



id\_Product : 5be1c5d41c9440000000000000000000  
Description : Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.  
Choisissez votre couleur :  

Tan

Ajouter au panier

En stock  
Sélectionnez ou rembournez  
Expédiez le jour même  
Livraison Express 48h

 Listing 

Merci pour votre commande!

Votre commande est confirmée :

Nous vous confirmons que la commande e75de0a0-1523-11eb-ba2b-c768c753b18c, pour un montant de 227 €, a bien été prise en compte  
A très vite sur Orinoco!

Page confirmation.html

Landing page initialisée par envoi  
formulaire

## PARTIE II \_STRUCTURE FICHIERS

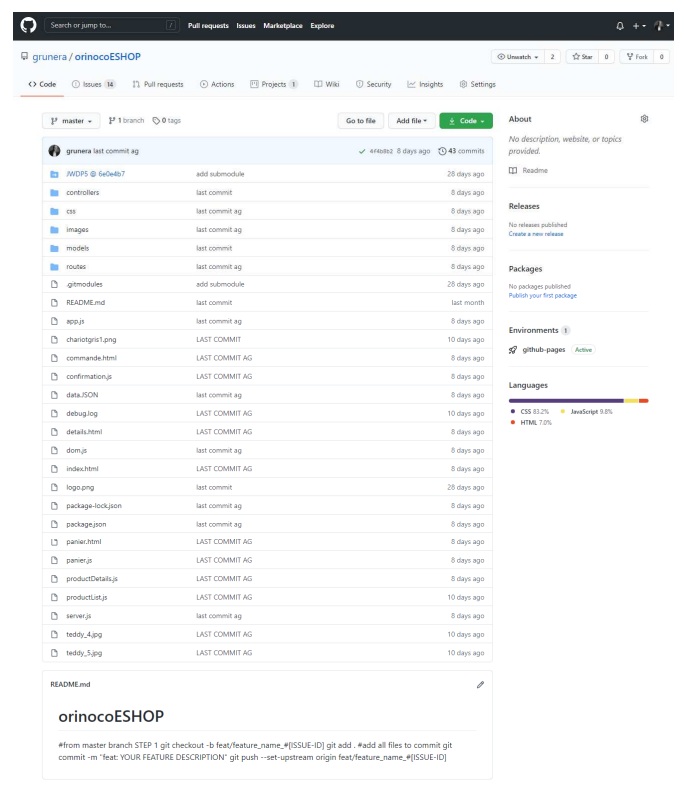
- 4 fichiers html :

index.html : page d'accueil + Listing  
details.html : page produit  
panier.html : page panier et formulaire  
confirmation.html : page confirm.commande

- 4 fichiers Javascript :

productList.js  
productDetails.js  
panier.js  
confirmation.js

- 2 feuilles de style :  
style.css  
style2.css



P5\_orinoco\_gruner\_audrey

## PARTIE III

# Evènements & fonctions Javascript



## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

➤ **Préambule : Définition de Javascript** : JavaScript est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web, qui permet de créer du contenu mis à jour de façon dynamique, de contrôler le contenu multimédia, d'animer des images

➤ **PAGE D ACCUEIL / LISTING PRODUITS = [Index.html](#) / [productList.js](#)**

➤ **La page se charge et fait apparaître dynamiquement la liste des produits stockés sur l'API avec leur :**

- image
- Nom
- Prix
- id produit
- déclinaisons colorielles



# EVENEMENTS ET FONCTIONS

**PAGE 1**  
= LISTING PRODUITS

Index.html  
produitList.js

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

index.html <> productList.js

### Page d'accueil = Listing

- ☑ la fonction de récupération de données fonctionne :  
**async function getProducts()**

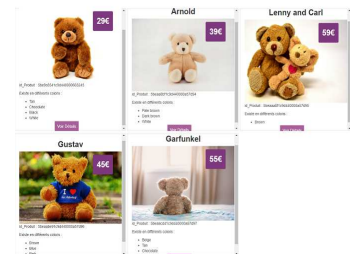
Fetch vers localhost:3000/api/teddies / Async function / reponse JSON

```
6
7
8  async function getProducts(){
9      const response = await fetch('http://localhost:3000/api/teddies')
10     return response.json()
11 }
```



Bienvenue sur Oursiwin!  
Découvrez un univers de tendresse et de douceur. Oursiwin a sélectionné pour vous le meilleur des teddies en peluche!  
Que vous soyez plutôt gros ours ou petit ours, nous avons tout pour vous. Cliquez sur les images pour découvrir nos teddies et leurs caractéristiques.

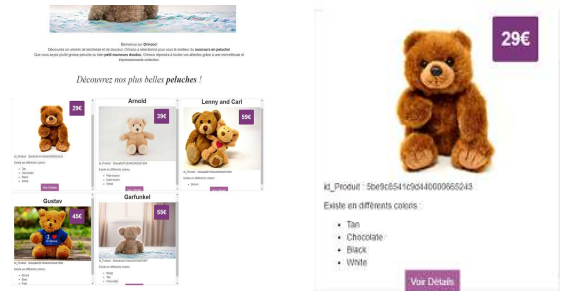
Découvrez nos plus belles peluches !



- **La méthode FETCH()** renvoie une promesse (un objet de type Promise) qui va se résoudre avec un objet **Response**. La promesse sera résolue dès que le serveur renvoie les en-têtes HTTP cad avant même qu'on ait le corps de la réponse. = méthode globale fetch() qui procure un moyen facile et logique de récupérer des ressources à travers le réseau de manière asynchrone. Source : [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- **function getProducts est une fonction asynchrone** . On fait appel à la fonction asynchrone getProducts et on attend son résultat grâce au mot clé await que l'on met devant l'appel de la fonction. Donc async / await sont 2 mots clés qui vont bloquer l'exé du code asynchrone jusqu'à qu'il retourne un résultat.
- en javascript il n'y a qu'un seul fil d'exécution du code source. Chaque ligne est exécutée l'une après l'autre en attendant la fin de l'exécution de la ligne précédente. En asynchrone la ligne suivante n'attend pas que la ligne asynchrone ait fini son exécution.  
> La fonction asynchrone est placée dans une sorte de file d'attente qui va exécuter toutes les fonctions qu'elle contient: c'est l'event loop
- **> Les Promises** : lorsqu'on exécute un code async celui-ci va nous retourner une promesse qu'un résultat nous sera envoyé prochainement. Lorsqu'on récupère 1 promise, on peut utiliser la fonction then() pour exécuter le code dès que la promesse est résolue et sa fonction catch() dès que survient une erreur/  
> **les callbacks dans une fonction d'événement** : `element.addEventListener('click',function(e){`: la fonction envoyée à addEventListener est une callback: elle est appelée plus tard dès que l'utilisateur clique sur l'événement. Les callbacks sont la base de l'asynchrone en JS .
- **return response.json** : JSON = Javascript Object Notation: il s'agit d'un format textuel se rapprochant en syntaxe à celui des objets en JS. En JS l'objet est assigné à une variable alors qu'en JSON on ne fait que décrire une structure. le navigateur sait directement le lire pas besoin de le PARSER. Léger.

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

index.html <> productList.js



☑ L'affichage de la liste produit fonctionne au load de la page fonctionne grâce à 2 fonctions utilisées :

1. function displayListProducts avec argument Products qui crée une div générale contenant individuellement chaque produit (let product of products & displayItemProduct (qui fait appel aux arguments product et list Element)

```
14
15 function displayListProducts(products){
16     const listElement = document.createElement('div')
17     for(let product of products){
18         displayItemProduct(product,listElement)
19     }
20
21     document.getElementById('boutique').appendChild(listElement)
22 }
```

➡ on affecte des attributs à la variable : **document.createElement** pour créer une div contenant **la liste des produits (listElement)**.

➡ **displayItemProduct(product,listElement)**: on appelle la fonction affichant chaque produit avec comme arguments (product) et (listElement)

➡ on crée la boucle **"for(let product of products)** pour scanner chaque élément de la boucle produits.

➡ **Document.getElementById** pour intégrer le nœud enfant ('listElement') dans l'html au niveau de l'#id boutique.

NOTES : Une **définition de fonction** (aussi appelée **déclaration de fonction** ou **instruction de fonction**) est construite avec le mot-clé **function**, suivi par :

- Le nom de la fonction.
- Une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
- Les instructions JavaScript définissant la fonction, entre accolades, { }.



## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

index.html <> productList.js

### 2. function displayItemProduct

qui crée l'affichage de tous les éléments de l'item produit (name/img/id/prix/colors. )

```
23
24 function displayItemProduct(product, listElement){
25     const itemElement = document.createElement('div');
26     itemElement.className = "produit";
27
28     const myH2 = document.createElement('h2');
29     const myPara1 = document.createElement('img');
30     const myPara2 = document.createElement('p');
31     const myPara3 = document.createElement('p');
32     myPara3.className = "price";
33     const myPara4 = document.createElement('p');
34     const myList = document.createElement('ul');
35
36     myH2.textContent = product.name;
37     myPara1.src = product.imageUrl;
38     myPara2.textContent = 'id_Produit : ' + product._id;
39     myPara3.textContent = product.price/100 + '€';
40     myPara4.textContent = 'Existe en différents coloris : ';
41
```

#### Document.createElement().....

- Création de la variable itemElement à laquelle on associe la **méthode document.createElement()** pour créer un élément HTML de la page , une 'div'.
- On crée pour chaque item du produit un élément HTML 'h2' 'img' 'p' , une liste 'ul' pour les déclinaisons couleurs.
- Les couleurs seront affichées sous forme de liste avec la const myList = document.createElement('ul');

#### myH2.textContent = product.name...

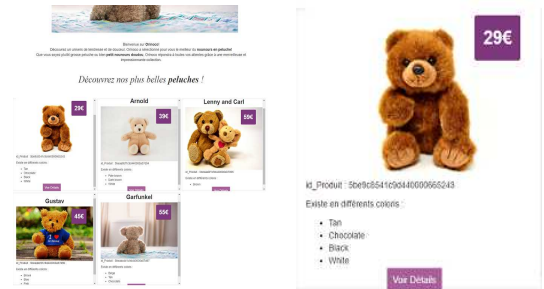
- on accède aux données de l'objet « product » et on veut afficher l'attribut « name » en contenu text (« textContent ») etc.....imageUrl , attribut source 'src'

```
41
42 const colors = product.colors;
43 for(let color of colors){
44     var listItem = document.createElement('li');
45     listItem.textContent = color;
46     myList.appendChild(listItem);
47 }
48
49 itemElement.appendChild(myH2);
50 itemElement.appendChild(myPara1);
51 itemElement.appendChild(myPara2);
52 itemElement.appendChild(myPara3);
53 itemElement.appendChild(myPara4);
54 itemElement.appendChild(myList);
55
56
```

#### Récupération des colors et affichage de la

liste des colors: Accès aux couleurs de l'objet product (product.colors) et création de l'élément liste HTML 'li' (document.createElement) pour afficher la liste des couleurs  
listItem est le nœud enfant qui sera placé dans myList.

Intégration des nœuds enfants dans itemElement (détails du produit)



## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

index.html <> productList.js

érents coloris :

rown

rown

Voir Détails

☑ le bouton "voir détails" renvoie bien vers la page détail produit

**btn.href = "details.html?id="+ product. id;** création du lien bouton pour page de destination détail produit.

➡ **Creation d'un lien en javascript :**  
**document.createElement("a").**

Pour activer le lien on accede aux donnees de l'objet "btn" en intégrant le lien "details.html?id" c est a dire en y associant "id" à l'adresse de la page destination "details.html" en rajoutant "?id" à l'adresse.

➡ Puis on rajoute un texte au bouton par **btn.textContent = "Voir Détails"**.

➡ **appendChild** : on ajoute un "noeud enfant à un noeud parent".

- Pour insérer le noeud enfant 'btn' (soit le bouton) dans la liste des items du product, on fait **itemElement.appendChild(btn)**.

- On intègre les produits à la liste Produits = le noeud lui même enfant 'itemElement' est intégré à list.Element grâce à **listElement.appendChild(itemElement)**.

### **Async function init()**

➡ toujours en asynchrone  
= fonctions qui se déclenchent en parallèle des autres fonctions :

**initialisation de l'affichage**  
la liste complète des produits avec **displayListProducts**

```
75
76  async function init(){
77      const products = await getProducts()
78      displayListProducts(products)
79
80
81  }
82
83
84  init()
85
86
```

## EVENEMENTS ET FONCTIONS

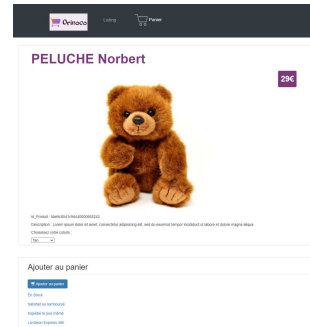
**PAGE 2**  
= DETAILS produit

details.html  
produitDetails.js

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

details.html <> productDetails.js

☑ L'utilisateur arrive sur la page détail produit - elle affiche le détail produit



```
1
2 function getProductId(){
3   const params = (new URL(document.location)).searchParams;
4   const id = params.get("id")
5   return id
6 }
7
8 async function getProduct(id){
9   const response = await fetch('http://localhost:3000/api/teddies/' + id)
10  return response.json()
11 }
12
13 async function loadProductDetails(id){
14   const productDetail = document.createElement('div')
15   for(let product of products){
16     displayItemProduct(product, listElement)
17   }
18
19   document.getElementById('product').appendChild(listElement)
20 }
```

### ⇒ Function getProductId()

new URL(document.location).searchParams / params.get("id") : on crée une fonction permettant d'extraire l'id de l'URL de la page "detail.html?id" Cf [developer.mozilla.org](https://developer.mozilla.org) : "La propriété en lecture seule searchParams de l'interface URL retourne un objet URLSearchParams permettant d'accéder aux arguments décodés de la requête GET contenu dans l'URL."

2) EN parallèle, on fait travailler 2 fonctions en asynchrone :

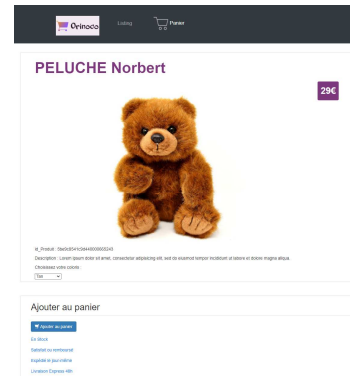
1. **async function getProduct(id)** avec await fetch  
localhost + id => requête fetch dans base localhost par l'"id"

2. **async function loadProductDetails(id)** => creation  
élément div affichant les items produit

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

details.html <> productDetails.js

☑ le sélecteur de couleurs fonctionne et permet de choisir parmi les différentes couleurs



```
const myPara5 = document.createElement('p');
myPara5.className = "colors";
```

```
//create select list colors//
const myList = document.createElement('select');
myList.id = 'selectedoption';
```

```
myH2.textContent = 'PELUCHE ' + product.name;
myPara1.src = product.imageUrl;
myPara2.textContent = 'id_Produit : ' + product._id;
myPara3.textContent = 'Description : ' + product.description;
myPara4.textContent = product.price/100 + '€';
myPara5.textContent = 'Choisissez votre coloris : ';
```

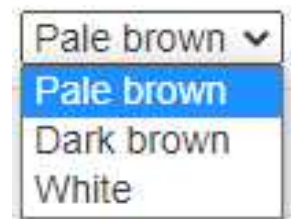
```
//create and append the colors options //
const colors = product.colors;
for(let color of colors){
  var listItem = document.createElement('option');

  listItem.textContent = color;
  myList.appendChild(listItem);
}
```

1

2


>> 1. création du sélecteur de couleur  
avec document.createElement('select')



>> 2. création et append des options de couleurs avec  
document.createElement('option') affichée sous forme de texte (text  
Content =color) puis on insère le noeud  
enfant listItem contenant les options de couleurs dans l'élément myList =  
le sélecteur "myList",

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

details.html <> productDetails.js

 Ajouter au panier

☑ Le bouton "ajouter au panier" fonctionne

### Function readStorage()

```
// requête GET pour récupérer l'élément "shop" dans le stockage local
function readStorage(){
  const storage = localStorage.getItem("shop")
  return JSON.parse(storage)
}
```


- ☞ "La méthode getItem() de l'interface Storage renvoie la valeur associée à la clé passée en paramètre."= **récupère l'élément "shop" du stockage local,**
- ☞ La méthode **JSON.parse()** analyse une chaîne de caractères JSON et construit la valeur JavaScript ou l'objet décrit par cette chaîne

*Notes : Les objets localStorage et sessionStorage vont nous fournir les propriétés et méthodes suivantes : setItem() : permet de stocker une paire clef/valeur. Prend une clef et une valeur en arguments ; getItem() : permet d'obtenir une valeur liée à une clef. cf,MDN*

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

---

details.html <> productDetails.js

 Ajouter au panier

☑ Le bouton "ajouter au panier" fonctionne

### Function addToStorage (id,option)

```
function addToStorage(id,option){  
  const storage = readStorage() || {}  
  storage[id] = option  
  localStorage.setItem("shop",JSON.stringify(storage))  
}
```

- ➡ on stocke avec **setItem** l'id produit et son option dans le **storage stockage local** *Notes : . SetItem, c'est une fonction du localStorage, qui demande à enregistrer une valeur pour l'item, passé en paramètre.*
- ➡ `|| {}`, c'est pour retourner {} (objet) au cas où ça n'existe pas en localStorage
- ➡ `JSON.stringify` = conversion en chaîne de caractères ( strings) de l'objet placé dans le local ! **si on appelle l'objet "shop" il ressort la valeur de storage soit id et option.**

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

details.html <> productDetails.js

- ✓ 1 alerte affiche que le produit a bien été pris en compte dans le panier.

### Async function init()

```
async function init(){
  const id = getProductId();
  const product = await getProduct(id)
  console.log(product)
  displayItemProduct(product, document.getElementById('product'))
  document.getElementById("submitbutton").addEventListener("click", function(){
    console.log('')
    const option = document.getElementById("selectedoption").value;
    addToStorage(id, option)
    alert('votre produit a été ajouté au panier')
  })
}

init()
```

b%20Developpeur/1%20Projets/orinoco/details.html?id=5beaa8bf1c9d44

Cette page indique  
votre produit a été ajouté au panier

OK

- ➡ Initialisation de la mise en panier avec **addEventListener (\*)** : événement **CLICK** bouton
- ➡ Intégration de la valeur de l'option choisie avec `const option = document...('selectedoption').value`
- ➡ puis initialisation `addToStorage (id, option)`

(\*) Source MDN : La méthode `addEventListener()` d'[EventTarget](#) installe une fonction à appeler chaque fois que l'événement spécifié est envoyé à la cible

- ➡ Puis création de l'alerte "Votre produit a été ajouté au panier"



# EVENEMENTS ET FONCTIONS

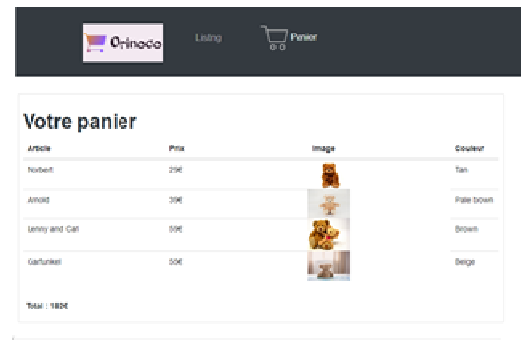
## **PAGE 3** = panier





panier.html  
panier.js

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

panier.html <> panier.js

- ☑ le panier s'affiche avec un tableau récapitulatif avec nom/image/prix/couleur



Article	Prix	Image	Couleur
Noddy	39€		Tan
Arnold	39€		Pale brown
Lenny and Cat	39€		Brown
Garfunkel	39€		Beige
Total : 156€			

### Function readStorage()

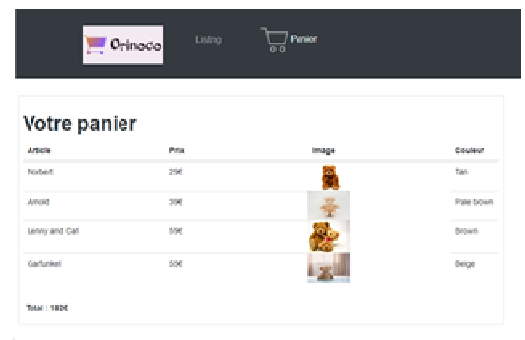
```
function readStorage(){  
  const storage = localStorage.getItem("shop")  
  return JSON.parse(storage)  
}  
  
async function getProducts(){  
  const response = await fetch('http://localhost:3000/api/teddies')  
  return response.json()  
}
```





- ➡ **Récupération de l'élément "shop« (panier)** (méthode get : renvoie la valeur associée à la clé placée en paramètre)
- ➡ la méthode `JSON.parse` analyse la chaîne de caractères JSON pour construire l'objet décrit par cette chaîne de caractère
- ➡ **Méthode FETCH** qui renvoie une promesse qui va se résoudre avec un objet `Response` = la connexion à l'API Teddies pour rechercher les produits

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### panier.html <> panier.js

- ☑ le panier s'affiche avec un tableau récapitulatif avec nom/image/prix/couleur – 1 -



Article	Prix	Image	Couleur
Nutsell	25€		Tan
Amande	20€		Pale brown
Lenny and Cat	30€		Brown
Catfunest	30€		Beige
Total : 105€			

## Function displaySelectedProducts

```
function displaySelectedProducts(filteredProducts,selectedProducts){  
  const tableElement = document.getElementById('cart-tablebody mb-4')  
  for (let product of filteredProducts){  
    product.option = selectedProducts[product._id]  
  
    const lineElement = document.createElement('tr')  
    const articleElement = document.createElement('td')  
    articleElement.textContent = product.name;  
    const priceElement = document.createElement('td')  
    priceElement.textContent = product.price/100 + '€';  
    const imageElement = document.createElement('img')  
    imageElement.src = product.imageUrl;  
    imageElement.className= 'imagepanier';  
    const colorElement = document.createElement('td')  
    colorElement.textContent = product.option;  
  
    lineElement.appendChild(articleElement);  
    lineElement.appendChild(priceElement);  
    lineElement.appendChild(imageElement);  
    lineElement.appendChild(colorElement)  
    tableElement.appendChild(lineElement);  
  }  
}
```

➡ création de la **fonction displaySelectedProducts** pour afficher les produits sélectionnés dans le panier dans un tableau avec arguments(filteredProducts , SelectedProducts) issus de methode products.filter et de la fonction selectedProducts ligne63 asyncfunc init

➡ création d'un tableau avec **const tableElement = document.getElementById('cart...')** qui prend les produits filtrés et sélectionnés avec leu propriété 'option' couleur. (" for ....product.option)

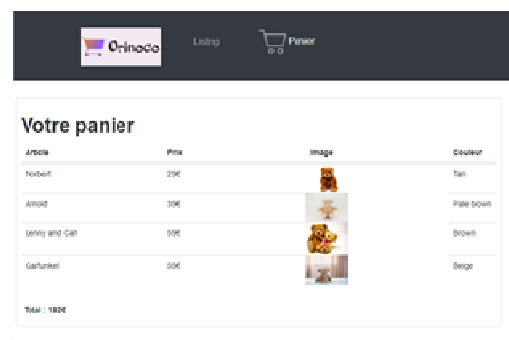
➡ const lineElement >> création de l'élément "ligne" dans le tableau





➡ on affiche les items produit sélectionné avec appendChild line Element puis on affiche la ligne dans le tableau avec appendChild tableElement

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### panier.html <> panier.js

☑ le panier s'affiche avec un tableau récapitulatif avec nom/image/prix/couleur – 2 -



Article	Prix	Image	Couleur
Notebook	25€		Tan
Armoire	30€		Pale brown
Lenny and Cat	30€		Brown
Garfunkel	30€		Beige
Total : 115€			

### Async Function init()

```
1  async function init(){
2    const selectedProducts = await readStorage()
3    console.log(selectedProducts)
    const products = await getProducts()
    console.log(products)
    const filteredProducts = products.filter(
      (product)=>selectedProducts[product._id])
    console.log(filteredProducts)
    displaySelectedProducts(filteredProducts,selectedProducts)
    displayTotal(filteredProducts)
```

>> fonction asynchrone function init déclenchée en parallèle :

1.appel des produits sélectionnés avec selected products = await readStorage, appel des produits du panier stocké dans localStorage

2. appel des produits avec await getProducts

3.méthode filter: liste retournée par l'API triée des produits sélectionnés dans le panier ( recoupe avec les produits sélectionnés dans le panier )

**source MDN :** La méthode **filter()** crée un nouveau tableau **qui** contient l'ensemble des éléments **qui** remplissent une condition fournie par la fonction de test passée en argument

**La méthode filter() crée et retourne un nouveau tableau contenant tous les éléments du tableau d'origine qui remplissent une condition déterminée par la fonction callback..**

exemple : `const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];const result = words.filter(word => word.length > 6);console.log(result);`

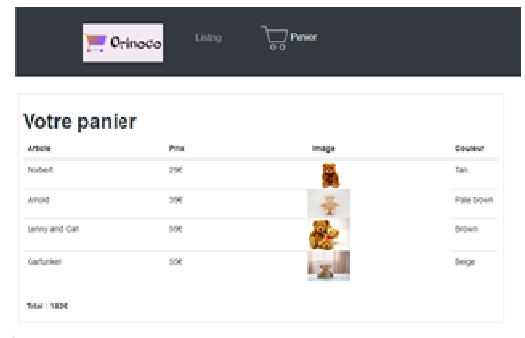
donc ici : `const filteredProducts = products.filter( (product)=>selectedProducts[product._id])>>> la méthode filter crée et retourne un nouveau tableau contenant tous les produits qui ont été sélectionnés dans le panier avec la cle de l'"id" produit`





>> **NB** filteredProducts, c'est la liste retournée par l'api, triée des produits non ajouté au panier/ products est un tableau, filter est une méthodes des tableaux qui te génère un nouveau tableau en tenant compte de ton filtre

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### panier.html <> panier.js

- ☑ le panier s'affiche avec un tableau qui affiche le total commandé en €
- 3 -



Article	Prix	Image	Couleur
Nutsell	25€		Tan
Almond	20€		Pale brown
Lenny and Cat	30€		Brown
Catfunest	30€		Beige
Total : 105€			

```
function recalculPrice(filteredProducts){
  const newprice = filteredProducts.reduce((acc,cur)=> acc + cur.price,0)
  document.getElementById('prixrecalcule').textContent = newprice/100;
}

function calculateTotal(filteredProducts) {
  return filteredProducts.reduce((acc,cur)=> acc + cur.price,0)
}

function displayTotal(filteredProducts){
  const total = filteredProducts.reduce((acc,cur)=> acc + cur.price,0)
  document.getElementById('total').textContent = total/100;
}
```

⇒ fonction recalculPrice pour diviser le prix par 100

⇒ fonction calculateTotal pour faire la somme des prix des filteredproducts du tableau

⇒ fonction displayTotal avec document.getElementById ('total') pour afficher ce total dans le tableau , prix divisé par 100, résultat affiché en text (text.content).

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### panier.html <> panier.js

☑ Le formulaire de validation de commande se complète avec indications des erreurs de saisie : nom/prénom inférieur à 3 lettres >> ALERTE "Insérer un nom / Prénom valide" & contrôle automatique de saisie valide de l'adresse email >> apparition d'un champ "Veuillez inclure @ dans votre adresse" etc....

Validez votre commande

Vous y êtes presque!  
Remplissez le questionnaire et validez votre commande

Vos Coordonnées

Nom :

Prénom :

Mail :

Confirmer mail

Votre Adresse

Adresse

Ville

```
async function init(){
  const selectedProducts = await readStorage()
  console.log(selectedProducts)
  const products = await getProducts()
  console.log(products)
  const filteredProducts = products.filter(
    (product) => selectedProducts[product._id])
  console.log(filteredProducts)
  displaySelectedProducts(filteredProducts, selectedProducts)
  displayTotal(filteredProducts)
}
```

```
const formElement = document.getElementById('myForm')
formElement.addEventListener('submit', async (evt) => {
  evt.preventDefault()
  console.log(evt.target)
  if (isFormValid(evt.target)) {
    await sendOrder(evt.target, filteredProducts)
  }
})
```

```
async function sendOrder(form, filteredProducts) {
  const lastName = form.nom.value
  const firstName = form.prenom.value
  const email = form.courriel.value
  const address = form.adress.value
  const city = form.ville.value
  const products = filteredProducts.map((product) => product._id)
  const contact = {
    lastName,
    firstName,
    email,
    address,
    city,
  }
}
```

⇒ const formElement, getElementById('myForm'): Création de l'événement 'submit' avec addEventListener. La fonction associée à addEventListener est une callback appelée dès que l'utilisateur submit le formulaire: 'submit' correspond à l'élément html <input type="submit" value="Envoyer">. Si la **fonction isFormValid** return une valeur **TRUE** alors elle valide/active la fonction sendOrder

⇒ async function sendOrder : les arguments passés à la fonction sont 'form' et 'filteredProducts' car elle utilise les valeurs du formulaire 'form' et des 'filteredProduct'

⇒ création de 2 constantes :

**const contact** avec valeurs last name/firstname etc.....

**const products** avec l'objet Map pour créer une carte de valeurs des filteredProducts

>> L'objet **Map** représente un dictionnaire, autrement dit une carte de clés/valeurs. N'importe quelle valeur valable en JavaScript (que ce soit les objets ou les valeurs de types primitifs) peut être utilisée comme clé ou comme valeur. Un objet Map permet de retrouver ses éléments dans leur ordre d'insertion.

>> *map est une méthode des tableaux qui crée un tableau à partir du tien, et dans lequel, chaque item aura été modifié comme souhaité, par la fonction passée en paramètre*

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### panier.html <> panier.js

☑ Le formulaire de validation de commande se complète avec indications des erreurs de saisie :  
nom/prénom inférieur à 3 lettres  
>> ALERTE "Insérer un nom / Prénom valide"  
& contrôle automatique de saisie valide de l'adresse email >> apparition d'un champ "Veuillez inclure @ dans votre adresse" etc....

```
var nomElt = document.getElementById("nom");
nomElt.value = "MonNom";

function isValid(form){
  const nomElement = form.nom;
  const prenomElement = form.prenom;
  const emailElement = form.courriel;
  const adressElement = form.adress;
  const villeElement = form.ville;

  const regexCourriel = /.+@.+\./;
  const valideCourriel = "";
  if (!regexCourriel.test(emailElement.value)) {
    alert('insérer un email valide');
    return false
  }

  if (nomElement.value.length<3){
    alert ('insérer un nom valide');
    return false
  }

  if (prenomElement.value.length<3){
    alert ('insérer un prénom valide');
    return false
  }

  if (adressElement.value.length<3){
    alert ('insérer une adresse valide');
    return false
  }
}
```

Validez votre commande

Vous y êtes presque!

Remplissez le questionnaire et validez votre commande

Vos Coordonnées

Nom : GRUNER

Prénom : AUDREY

Mail : gruner.audrey@gmail.com

Confirmer mail gruner.audrey@gmail.com

Votre Adresse

Adresse 5 rue Louis Pasteur

Ville Lampertheim

Envoyer Annuler

### ➔ Function isValid(form)

➔ création des constantes avec propriétés "nom" "prenom" indiquées dans le html dans < label for="nom"....>

➔ création du contrôle de validité Email avec const regexCourriel

➔ contrôle validité saisie du nom /prénom etc... avec if (value.length<3) return false / alert ('insérer un nom valide')

## EVENEMENTS ET FONCTIONS

### **PAGE 4**

= confirmation  
commande

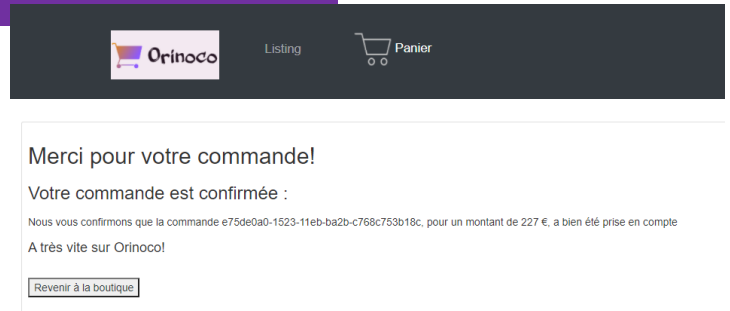
confirmation.html  
confirmation.js



## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### confirmation.html <> confirmation.js

- ☑ le formulaire est envoyé et s'ouvre une landing page de confirmation de sa commande affichant le numéro de la commande "order\_id" et le prix total "total" de la commande



### Dans panier.js

```
}  
const orderRequest = await fetch('http://localhost:3000/api/teddies/order', {  
  method: 'POST',  
  mode: 'cors',  
  credentials: 'same-origin',  
  referrerPolicy: 'no-referrer',  
  headers: {  
    'Content-Type': 'application/json'  
    // 'Content-Type': 'application/x-www-form-urlencoded',  
  },  
  body: JSON.stringify({  
    contact,  
    products,  
  })  
})
```

### >> on POSTE la commande "order" à l'API localhost TEDDIES avec la Methode POST

>> Le « **Cross-origin resource sharing** » (CORS) ou « partage des ressources entre origines multiples » (en français, moins usité) est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent **utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant**. Un agent utilisateur réalise une requête HTTP **multi-origine (cross-origin)** lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.

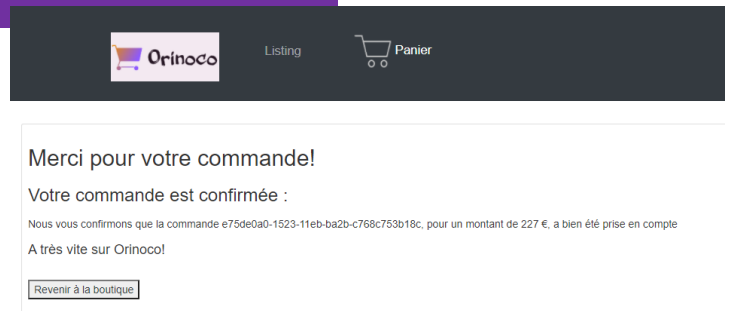
>> **Methode POST** : Côté client, un formulaire HTML n'est rien d'autre qu'un moyen commode et convivial de configurer une requête HTTP pour envoyer des données à un serveur  
. L'utilisateur peut ainsi adresser des informations à joindre à la requête HTTP.  
La méthode POST est un peu différente. C'est la méthode que le navigateur utilise **pour demander au serveur une réponse prenant en compte les données contenues dans le corps de la requête HTTP** : « Hé serveur ! vois ces données et renvoie-moi le résultat approprié »

- > création de la constante orderId par envoi d'une requête JSON à localStorage.
- > Méthode **setItem** : **méthode permettant de stocker des valeurs dans l'objet localStorage**. Elle prend 2 paramètres : une clé et une valeur. Ici la clé est 'total\_order' et sa valeur est le résultat de la fonction calculateTotal(filteredProducts).
- > document.location.href etc....= la réponse de la requête se fait sur une landing page qui affiche l'orderId (console log)

## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### confirmation.html <> confirmation.js

- ☑ le formulaire est envoyé et s'ouvre une landing page de confirmation de sa commande affichant le numéro de la commande "order\_id" et le prix total "total" de la commande



### Dans panier.js

```
//  
const {orderId} = await orderRequest.json()  
console.log(orderId)  
localStorage.setItem('total_order', calculateTotal(filteredProducts))  
document.location.href = 'commande.html?order_id='+orderId  
}
```

➡ création de la constante orderId par envoi d'une requête JSON à localStorage.

➡ Méthode **setItem** : **méthode permettant de stocker des valeurs dans l'objet localStorage**. Elle prend 2 paramètres : une clé et une valeur. Ici la clé est 'total\_order' et sa valeur est le résultat de la fonction calculateTotal(filteredProducts).

➡ document.location.href etc....= la réponse de la requête se fait sur une landing page qui affiche l'orderId (console log)

> création de la constante orderId par envoi d'une requête JSON à localStorage.

> Méthode **setItem** : **méthode permettant de stocker des valeurs dans l'objet localStorage**. Elle prend 2 paramètres : une clé et une valeur. Ici la clé est 'total\_order' et sa valeur est le résultat de la fonction calculateTotal(filteredProducts).

> document.location.href etc....= la réponse de la requête se fait sur une landing page qui affiche l'orderId (console log)

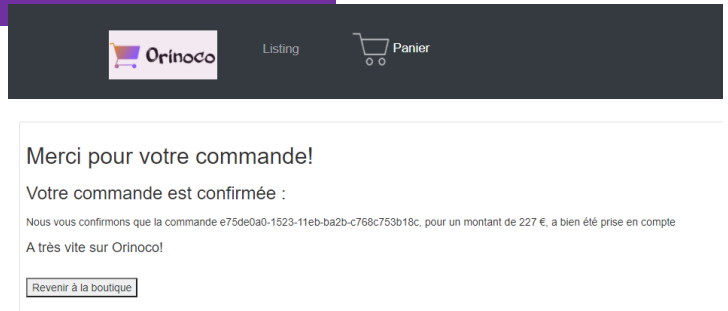
## PARTIE III \_EVENEMENTS ET FONCTIONS JAVASCRIPT

### confirmation.html <> confirmation.js

- ☑ le formulaire est envoyé et s'ouvre une landing page de confirmation de sa commande affichant le numéro de la commande "order\_id" et le prix total "total" de la commande

#### Dans confirmation.js

```
function displayConfirmation(orderId, total) {  
  document.getElementById('confirmation').textContent = `Nous vous confirmons que la commande ${orderId},  
  pour un montant de ${parseInt(total)/100} €, a bien été prise en compte`;  
  localStorage.clear()  
}  
  
function init() {  
  
  const params = (new URL(document.location)).searchParams;  
  const orderId = params.get("order_id")  
  
  if(!orderId) document.location.href = "index.html"  
  const total = localStorage.getItem('total_order')  
  displayConfirmation(orderId, total)  
}  
  
init()
```



⇒ **création de la fonction displayConfirmation (orderId, total)** : intégration dans l'html à l'#id confirmation de la phrase 'Nous vous ...' et intégration des valeurs "orderId" (générée et renvoyée par localStorage) et de la variable "total" convertie en nombre par la fonction parseInt .

⇒ **parseInt(chaine, [base])** - Fonction JS **qui** convertit une chaîne de caractères en nombre

⇒ remise à zéro du panier localStorage. avec localStorage.clear() ( = fonction)

#### ⇒ **function init :**

⇒ on récupère l'**order id** par params.get "order (id) "qui est crée automatiquement par l'api searchParams = pour extraire le paramètre "order\_id" de l'URL de index.html

⇒ on récupère **avec getItem le montant de total\_order** dans le **panier localStorage**

⇒ on initialise la fonction displayConfirmation(orderId, total)

P5\_orinoco\_gruner\_audrey

## PARTIE IV

### Plan de Tests



P5\_orinoco\_gruner\_audrey

## PARTIE V

### Compétences validées



# Compétences évaluées

## •Valider des données issues de sources externes :

- ⇒ Recherche des items products dans l'API
- ⇒ Recherche de l'order\_id stocké dans l'API
- ⇒ Recherche des options couleurs colors

## •Gérer des événements JavaScript :

- ⇒ Bouton « voir détails »
- ⇒ Bouton « ajout panier »
- ⇒ Alerte « Produit ajouté dans panier »
- ⇒ Sélection des options « colors »
- ⇒ Compilation du tableau « Panier » et « total panier »
- ⇒ Envoi du formulaire
- ⇒ Génération d'une page html « confirmation »
- ⇒ contenant l'order\_id et le montant « total » de la commande

## •Créer un plan de test pour une application

=> voir fichier excel; plan de tests fonctionnels

## •Interagir avec un web service avec JavaScript

=> Connexion localhost 3000 API avec node.js /npm



**MERCI  
POUR  
VOTRE  
ATTENTION**

