

Présentation du site

* Composé de 4 pages html :

- l'accueil ou l'index,
- « la fiche produit »,
- la commande avec le panier et le formulaire,
- la page de confirmation,

* ressources servies par une API avec 3 url ou routes définies :

- url = "<http://localhost:3000/api/products>";
=> tous les produits
- url = "<http://localhost:3000/api/products/>" + product_id;
=> un produit avec l'_id »
- url1 = "<http://localhost:3000/api/products/order>"
qui après vérif, retourne un json dont un orderCommand
qui est une série d'hexa,

* principes

- La page index est située un cran au dessus du folder pages,
- Par choix le dossier image a été remonté au plus haut niveau (ceci permet d'avoir des backup « autonomes avec html ,js, css de 25 koctet)
- Chaque page html (les 4) appelle un script qui reprend son "nom" et est logé dans le dossier js.
(+ si besoin appel à commun.js qui regroupe qq, fonctions communes, qui est dans ce dossier).
- Chacun de ces scripts appelle des scripts modules (mjs) dans le folder module

Se placer dans « front »

NB : pour lancer le site :

- cd api ;
- node server,
- lancer serveur et par exemple <http://127.0.0.1:5500/front>

En Préambule possibilité d'avoir un panier pré-rempli (basique)

Dans la page d'accueil vous pouvez cocher la case :

```
function jeuPanier() {  
  if (document.getElementById("jeuEssai").checked) {  
    let jsonlePanier = {  
      "055743915a544fde83cfd9c904935ee7": { 2: 4, 0: 2, 1: 5 },  
      "77711f0e466b4ddf953f677d30b0efc9": { 0: 1, 1: 4 },  
      "a6ec5b49bd164d7f9be10f37b6363f9fb": { 2: 1, 3: 1 },  
    };  
    localStorage.setItem("panier", JSON.stringify(jsonlePanier));  
  }  
}
```

Ce qui permet de présenter la structure
LePanier = {id123{0:2,2,5},
 {id124{1:12,3,1}};

4 articles de la couleur d'indice 2
2 articles de la couleur d'indice 0
5 articles de la couleur d'indice 1

Header et Footer

- À titre anecdotique : le header et footer sont « dynamiques » pour 2 raisons :
 - on a besoin de 2 ou 3 paths pour les `img src` et les `a href` =.. '..'
 - pour éviter de répéter les choses (Do Not Repeat)

On importe une ressource depuis les *.js

```
import * as moduleEntete from "../module/entete.mjs";  
.....  
moduleEntete.ecrireHeaderFooter();
```

On appelle la fonction

```
const adresse = {  
  nom: "kanap",  
  tel: "01 23 45 67 89",  
  mail: "support@name.com"etc ....}
```

Un objet pour les paramètres

```
Function ecrireHeaderFooter(){....  
if (|Url == "/" + leFolder + "/" || |Url == "/" + leFolder + "/index.html") {  
  cheminIndex = "./";  
  chemin2 = "./pages/";  
}... return du html .}
```

Ici la fonction avec un if sur les paths

On l'exporte ici pour pouvoir l'importer ailleurs !

```
export { ecrireHeaderFooter };
```

l'accueil ou l'index 1: api

```
function initIndex() {  
  fetch(url, { method: "GET" })  
    .then((data) => {  
      console.log(data.ok + ", " + data.status);  
      return data.json();  
    })  
    .then((products) => {  
      const fragment = moduleEdit.ecrireListe(products);  
      document.getElementById("items").appendChild(fragment);  
    })  
    .catch(function (error) {  
      console.log("erreur : " + error);  
    });.....}
```

On vérifie la promesse et le cas échéant on la convertit en json..

En cas de retour valide on appelle une méthode :
Ici un template

En cas d'échec ... on indique l'erreur

Dans la littérature :

« La méthode `fetch()` retourne une promesse.
Si la promesse renvoyée est resolve,
cela signifie que la fonction dans la méthode `then()` est bien exécutée.
Cette fonction contient le code qui permet de traiter les données
reçues à partir de l'API. »

l'accueil ou l'index 2: template

- en cas de succès : afficheProduct appelle le module : moduleEdit from "../module/edition.mjs";
- et sa méthode ecrireListe() , celle ci dans une boucle appelle une sous méthode pour chaque produit.

```
const fragment = moduleEdit.ecrireListe(products);  
document.getElementById("items").appendChild(fragment);
```

Le retour modifie le « DOM » de la page

```
function ecrireListe(listeProduit) {  
  let fragmentSom = new DocumentFragment();  
  for (let element of listeProduit) {  
    fragmentSom.appendChild(ecrireUnProduit(element));  
  }  
  return fragmentSom;  
}
```

Boucle avec des fragments

```
function ecrireUnProduit(element) {  
  let fragment1 = ..  
  const clone = document.importNode(templateProduit.content, true);  
  const image = clone.querySelector("img");  
  image.src = element.imageUrl;  
  image.alt = element.altTxt;  
  ....  
  return..  
}
```

On prend un template, que l'on clone
puis on isole un élément de celui ci que l'on modifie :
Ici on renseigne 2 attributs de img

le clone ainsi modifié est retourné :
document.getElementById("").appendChild(clone)

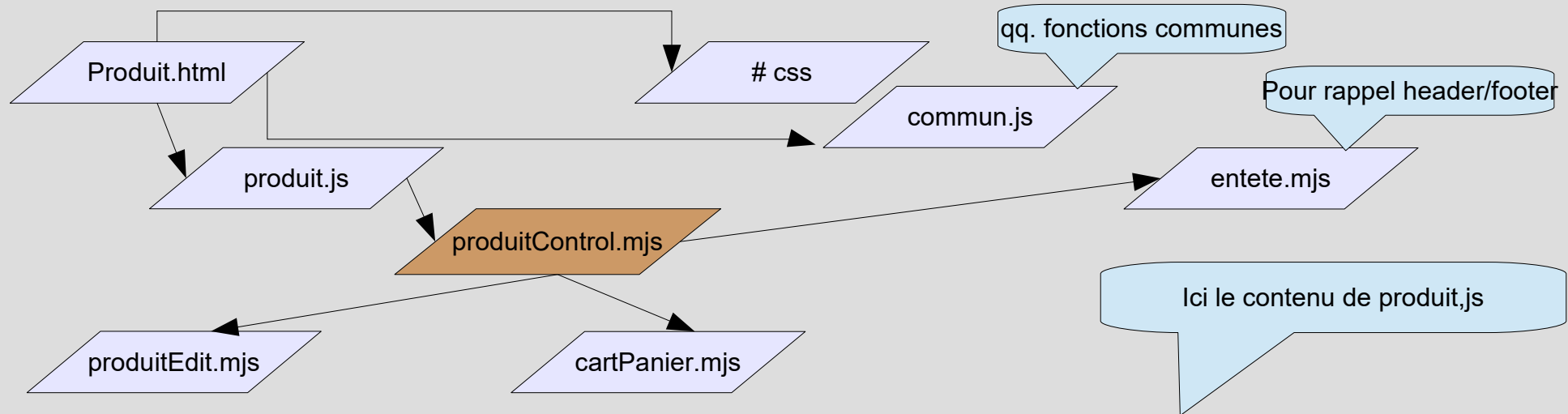
Page produit (templating +structure)

```
for (let i = 0; i < leProduit.colors.length; i++) {  
  let choix = document.createElement("option");  
  choix.textContent = leProduit.colors[i];  
  choix.value = i;  
  select.appendChild(choix);  
}
```

Ici template avec aussi sur une boucle la création d'elts du DOM

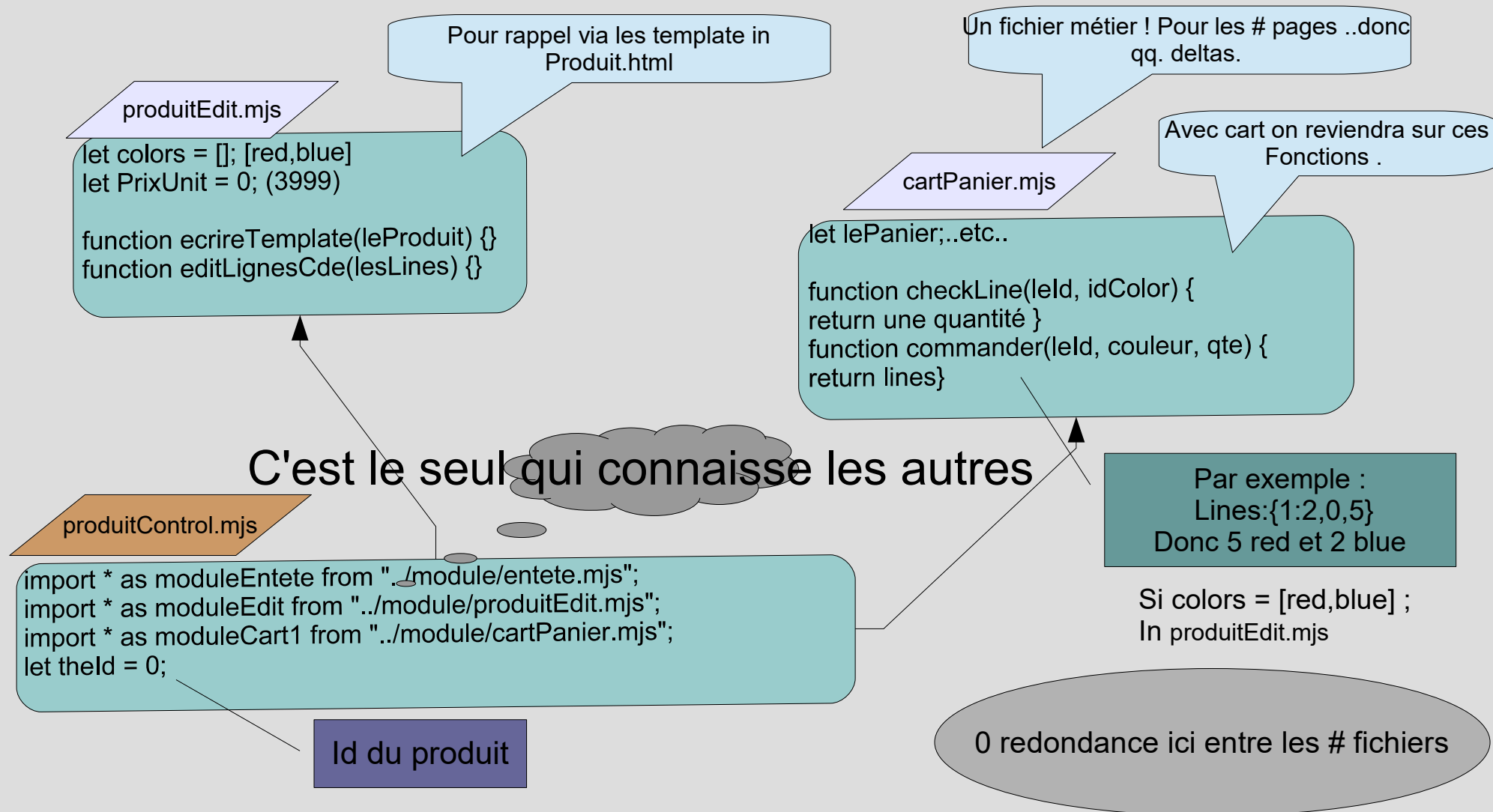
Là encore un Template
mais pour présenter un petit delta/
/ Template index

Cette page est d'abord statique mais est amenée à être « modifiée » par le user.. d'où **un contrôleur**



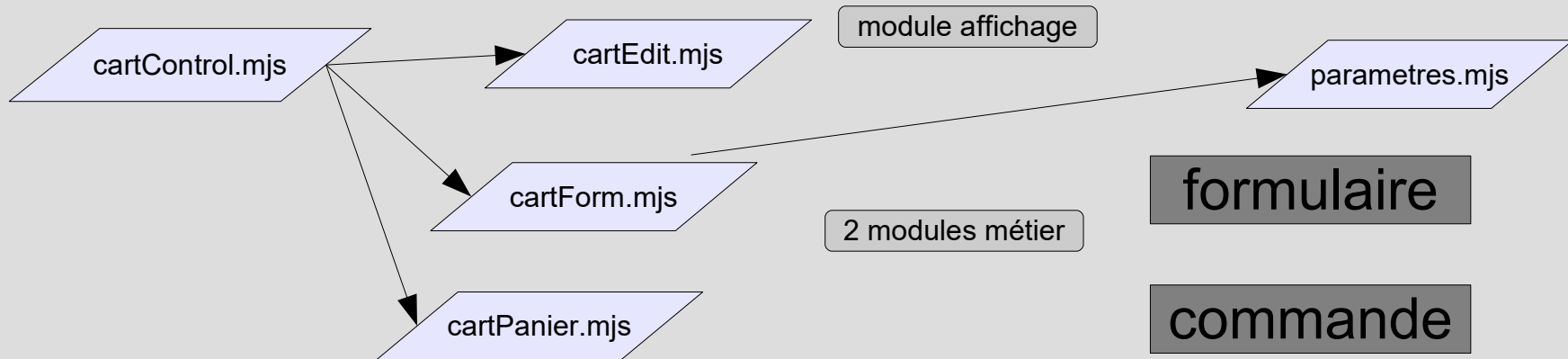
```
import * as moduleControl from "../module/produitControl.mjs";  
moduleControl.initProduit();
```

Page produit (fichiers)



Page Cart - introduction

- La structure est assez semblable à product... (en simplifiant) :



formulaire

L'Api impose x champs de formulaire, qui sont repris par le html/css fourni..(et écrit en dur), au moins 2 solutions :

1 - laisser le html en l'état et ne faire que les vérifications (regex et messages d'erreur)

2 - effacer le html (ou pas) et créer un objet qui va nous permettre de stocker toutes les spécificités souhaitées. (regex, messages d'erreur, placeholder, etc...)

Solution retenue

Avantage : par ex,
Si on a les coord.client en base de données on peut pré-remplir dynamiquement le formulaire

Page Cart (formulaire 1)

L'Api impose x champs de formulaire, repris par le html/css fourni..

Ce qui permet de définir un objet unClient composé d'objets avec par ex : city

```
const unClient = {  
  firstName: {},  
  City{},  
  Email {...},  
  etc..  
};
```



parametres.mjs

```
city: {  
  type: "number",  
  entete: "Votre code postal (5 chiffres exactement!)",  
  pholder: "Ça alors ! vous avez un code postal SVP ?",  
  leRegex: /^\\d{5}$/,  
  UnMessage: "Le code postal doit être composé de 5 chiffres exactement !",  
  exemple: 75019,  
},.
```

édition

```
function ecrireFormulaire(preRemplir, unClient) {  
  let text = "";  
  for (let key in unClient) {  
    if (unClient.hasOwnProperty(key)) {  
      let valeur = unClient[key];  
      objetVerif[key] = [valeur.leRegex, valeur.UnMessage];  
      let lexemple = preRemplir == 1 ? "value=" + valeur.exemple + "" : "";  
      text += `<div class="cart__order__form__question">  
        <label for="${key}">${valeur.entete}: </label>  
        <input type="${valeur.type}" id="${key}" name="${key}" placeholder="${valeur.pholder}" ${lexemple} >  
        <p id="${key}ErrorMsg"></p>`;   
    }  
  }  
  return text;  
}
```

on obtient un objet
objetVerif utile pour valider le form

Si preRemplir==1
On pré-remplit le questionnaire

Page Cart (formulaire 2 REGEX)

Vérification regex :

```
if (estValide(valeur, regle))
```

```
function estValide(value, regle) {  
    return regle.test(value);  
}
```

qq. exemples de regex
mis en place :

```
firstName: {  
    leRegex: /^[a-zA-Z]{1}[A-Za-z'àâãäåçèéêëìíîïðòóôõöùúûüý -._\s]*$/,  
    UnMessage: "Votre nom ne peut pas contenir de chiffres ou de signes tel que (, ° +} etc ...",},  
address: {  
    leRegex: /^[0-9]{1}[A-Za-z-0-9'àâãäåçèéêëìíîïðòóôõöùúûüý -._\s-]*$/,  
    UnMessage: "Votre adresse ... doit commencer par un chiffre",},  
city: {  
    leRegex: /^\d{5}$/,  
    UnMessage: "le code postal doit être composé de 5 chiffres exactement !",},  
email: {  
    leRegex: /^[w-\.]+\@([\w-]+\.)+[\w-]{2,4}$/,  
    UnMessage: "cet adresse mail n'est pas valide",...}
```

vérification

Page Cart (formulaire 3)

on utilise objetVerif pour inspecter et valider le form

```
function verifForm() {  
  let cptErreur = 0; let unContact = {};  
  for (const [key, arrayVerif] of Object.entries(objetVerif)) {  
    const regle = arrayVerif[0];  
    const inner0 = document.getElementById(key);  
    const valeur = inner0.value;  
    const inner1 = document.getElementById(key + "ErrorMsg");  
  
    if (estValide(valeur, regle)) {  
      ....  
    } else {  
      cptErreur++;  
      inner1.innerHTML = arrayVerif[1];  
      inner0.style.backgroundColor = "red";  
    }  
    unContact[key] = valeur;  
  }  
  console.log("nbre erreur" + cptErreur);  
  if (cptErreur == 0) {  
    return unContact;  
  } else {  
    return {};  
  }  
}
```

On parcours l'objet « converti en tableau

On crée un objet unContact

```
const unContact = {  
  firstName: « titi»,  
  City : « Lyon»,  
  Email :,  
  etc..  
};
```

La vérification est appelée par testOrder() qui lance la fonction verifForm() ,
si c'est ok et que panier #0 alors on reformate celui ci en newPanier=[id122,id123,...]
(cad au format attendu par l'api) et on envoie le json de l'ensemble à l'api.
Si celle ci retourne une rép avec un orderId=" alors
window.location = `./confirmation.html?idCommande=\${res.orderId}`;

Page Cart (commande 1)

commande

Ecrire le html : en ouverture de page : template (idem) juste une imbrication de 2 templates : article et ligne de Cde.

On note le besoin de rappeler l'API pour récupérer les caractéristiques des produits

cartControl.mjs

```
import * as myParam from "../module/parametres.mjs";  
import * as moduleEdit from "../module/cartEdit.mjs";  
import * as moduleCart from "../module/cartPanier.mjs";  
import * as moduleForm from "../module/cartForm.mjs";
```

Idem le contrôleur ..est seul à « dialoguer »

On note juste 2 variables avec un scope « page » :
lePanier = {id123{0:2,2,5},{id124{1:12,3,1}}};
lesPrix = {id123:4999,id124:999} ;
Nécessaires et suffisants pour actualiser le panier !

cartEdit.mjs

```
import * as moduleControl from "../module/cartControl.mjs";
```

cartPanier.mjs

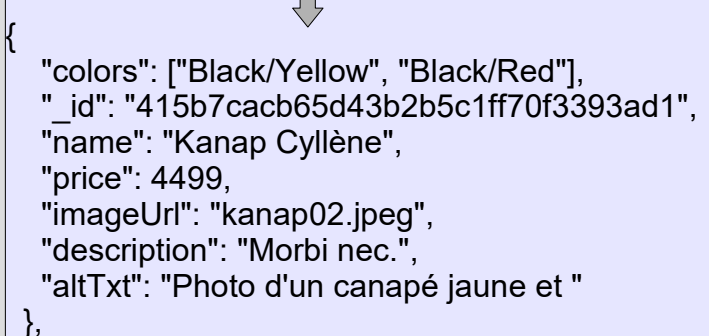
```
import * as moduleControl from "../module/cartControl.mjs";  
let lePanier = JSON.parse(localStorage.getItem("panier")) ?? {};  
let lesPrix = {};
```

Cart - Cde : formatage/édition du html du panier

L'idée était d'avoir un editCart.js le plus « basique possible » !

```
function preparePanier(listeProduit) {  
  /*  
  on renvoi au controleur un array de 2 array  
  A le premier : arrayCart  
  * =[[leCanapé,pu,qtetotal,prixtotal],  
  [[idcolor1,textcolor1,qt1,prix lignecolor1]],[idcolor2,textcolor2,qt2,lignecolor2]]]  
  c'est à dire :  
  - array[0] :avec les caractéristiques de l'article et son prix total  
  - array[1] : array de x couleurs et avec pour chacune les caractéristiques de la ligne  
  
  B le deuxieme : arrayTotal  
  avec juste qteTotal et prixTotal;  
  return [arrayCart, arrayTotal];  
}
```

LeCanapé (le n° : i) ou arrayCart[i][0][0] :



```
{  
  "colors": ["Black/Yellow", "Black/Red"],  
  "_id": "415b7cacb65d43b2b5c1ff70f3393ad1",  
  "name": "Kanap Cyllène",  
  "price": 4499,  
  "imageUrl": "kanap02.jpeg",  
  "description": "Morbi nec.",  
  "altTxt": "Photo d'un canapé jaune et "  
},
```

À l'intérieur d'une double boucle on a cela :

```
const arrayLigne = [idcolor, textColor, qte, formatPrix(qte * lePrix)];  
arrayProduit.push(arrayLigne);
```

Qui est réutilisé par editCart.js

```
function ecrireUneLigne(unId, arrayLigne) {  
  const indiceColor = arrayLigne[0];  
  const laColor = arrayLigne[1];  
  const qty = arrayLigne[2];  
  const prixLigne = arrayLigne[3];
```

Cela semble cohérent ?

mais peu être too much ? ?

```
/* console.log(arrayCart[0][0][1]);  
console.log("color " + arrayCart[0][1][1]);  
console.log(arrayCart[0][0][0].name); */
```



Bouton + ou -

```
moduleControl.ajouterUn(unId, indiceColor, -1 ou 1);
```

Supprimer la ligne

```
moduleControl.deleteLigne(unId, indiceColor);
```

Page Cart (commande modifiée 1)

Donnée du user
Donc celle la
à vérifier !

Change in area

```
moduleControl.checkModifQty(unId, indiceColor, this.value);
```

Supprimer tout

```
moduleControl.deleteArticle(l_Id);
```

4/5 fonctions *entrantes* ou interactions

→
contrôleur

Une fonction sortante :
modifQty()

```
function ajouterUn(unId, color, sens) {
  if.... moduleCart.modifQty(unId, color, newQte);....
}
function checkModifQty(unId, indicecolor, qte) {
  if moduleCart.modifQty(unId, indicecolor, qteVerif);...}

function deleteArticle(unId) {moduleCart.modifQty(unId, -1, 0);}
function deleteLigne(unId, uneColor) { moduleCart.modifQty(unId, uneColor, 0);}
```

On teste >= à un max(100) et >=0 !

Idem + on teste si c'est un entier (et >=0 etc..)

L'indice couleur à -1 astuce
pour transmettre
deleteArticle.

Page Cart (commande 3 modification du panier)

La solution simple et robuste : réécrire la commande
Sinon s'appuyer sur du innerHTML pour modifier only ce qui doit l'être

```
function modifQty(id, color, qteVerif) {  
  const color2 = modifPanier(id, color, qteVerif);  
  actuStorage();  
  actuEcran(id, color2, qteVerif);  
}
```

Actualiser le panier

Actualiser le localStorage

Actualiser l'écran

```
function modifPanier(id, color, qteVerif) {  
  if (qteVerif > 0) {  
    lePanier[id][color] = qteVerif;  
  } else {  
    delete lePanier[id][color];  
    if (Object.keys(lePanier[id]).length == 0 || color == -1) {  
      delete lePanier[id];  
      delete lesPrix[id];  
      color = -1;  
    }  
    if (lePanier == 0) {  
      lePanier = {};  
    }  
  }  
  return color;  
}
```

Purgé des lignes
de commentaires

Basique :
if (qteVerif > 0) {on modifie}
else {on delete ...}

|| color == -1) : test pour intercepter « supprimer article »
action color = -1 ... pour pouvoir actualiser le html

Ici, si on ne fait d'innerHTML,
on peut directement réécrire
tout le panier

Page Cart (commande 4 modification du HTML ...)

Actualiser l'écran

```
function actuEcran(id, idColor, newQty) {  
  ...  
  for (const [unId, lignes] of Object.entries(lePanier)) {  
    ...  
    for (const [color, qte] of Object.entries(lignes)) {  
      qtArticle += qte;  
      ...  
      if (id == unId) {  
        if (newQty > 0) {  
          moduleControl.modifArticle(unId, qtArticle, formatPrix(prixArticle)); ★  
          moduleControl.modifLigne(unId, idColor, newQty, formatPrix(prixLigne)); ★  
        }  
        if (newQty == 0 && qtArticle != 0) {  
          moduleControl.modifArticle(unId, qtArticle, prixArticle); ★  
          moduleControl.razLigne(unId, idColor); ★  
        }  
      }  
    }  
    ...  
  }  
  if (idColor == -1) {  
    /*ici en effet id n est plus présent dans le panier donc... */  
    moduleControl.razArticle(id); ★  
  }  
  /*dans tous les cas de figure...on actualise le total */  
  moduleControl.modifTotal(qteP, formatPrix(prixP)); ★  
}
```

Purgé des lignes
de calcul métier et des
commentaires

Fonctions appelées ici (dans le for) car qtArticle
spécifique à chaque article est accessible ici

★
Fonctions directement re-routées
par le contrôleur à l'éditeur

Par exemple :
function modifArticle(unId, qtArticle, prixArticle) {
 moduleEdit.modifArticle(unId, qtArticle, prixArticle);
}

En filant l'exemple :
function modifArticle(id, qtArticle, prixArticle) {
 document.getElementById("qte_" + id).innerHTML = qtArticle;
 document.getElementById("prix_" + id).innerHTML = prixArticle;
}

Unifier le panier/cart.html et /produit.html

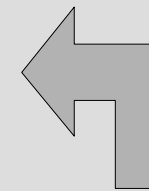
Dans la page cart avec **modifPanier**

```
function modifPanier(id, color, qteVerif) {  
  lePanier[id][color] = qteVerif;  
} else {  
  delete lePanier[id][color];  
}}
```

Soit on modifie le panier

Soit on delete le panier : lePanier[id][color]; ou lePanier[id];

Or sur « produit.htm » le produit **n'est pas forcément** dans le panier...
Donc on déclenche moduleCart1.commander(theld, couleur, qteVerif);



```
function commander(leld, couleur, qte) {  
  if (lePanier[leld]) {  
    if (lePanier[leld][couleur]) {  
      const quiSertPas = modifPanier(leld, couleur, qte);  
    }  
    lePanier[leld][couleur] = qte;  
  }  
  lePanier[leld] = {};  
  lePanier[leld][couleur] = qte;  
}  
..  
if (lePanier[leld]) {  
  .... return lePanier[leld]  
} else {  
  return {};  
}
```

On réutilise la fonction...

On crée une nouvelle ligne de la couleur choisie...

On crée une nouvelle ligne de l'article courant.

LocalStorage +
Le cas échéant on renvoie les ligneS de l'article courant.

Formatage des prix :

Intro : au départ bcq, de problèmes d'arrondi.
donc solution retenue de calculer avec des entiers

Dans communs.js : 2 fonctions possibles
(appelée au moment de l'affichage):

Fonction standard

```
function formatPrix(prix) {  
  prix = new Intl.NumberFormat("fr-FR", {  
    style: "currency",  
    currency: "EUR",  
  }).format(prix / 100);  
  return prix;  
}
```

Fonction perso

```
function formatPrix(prix) {  
  const millierSep = " ";const decimalSep = ",";  
  let entier = parseInt(prix / 100);  
  let decimal = prix % 100;  
  let lesCent = decimalSep + decimal;  
  if (decimal < 10) {  
    lesCent = decimalSep + "0" + decimal;  
  }  
  decimal = decimal == 0 ? "" : decimalSep + decimal;  
  
  entier = entier.toString();  
  const nbrChiffre = entier.length - 3;  
  if (nbrChiffre > 0) {  
    entier = entier.slice(0, nbrChiffre) + millierSep + entier.slice(nbrChiffre);  
  }  
  return entier + lesCent + " €";  
}
```

ici limité à 1 million

Page de confirmation

Pour rappel :

```
window.location = `./confirmation.html?idCommande=${res.orderId}`
```

Page statique très simple... on doit juste parser l'url ..pour obtenir Numcde

```
const urlParams = new URLSearchParams(window.location.search);  
const Numcde = urlParams.get("idCommande");
```

```
/**a l'ancienne mais à éviter !  
const Numcde = document.URL.split("idCommande=")[1];  
*/
```

Puis on teste : ici que c'est une suite d'hexadécimaux séparés par « - » ??

```
if (Numcde) {  
  const regle = /^[0-9a-fA-F-]*$/;  
  const testHexa = estValide(Numcde, regle);  
  
  if (testHexa) {  
    ....«OK»  
  }else{  
    « erreur »  
  }  
}
```

Voilà...
Si vous avez des questions !