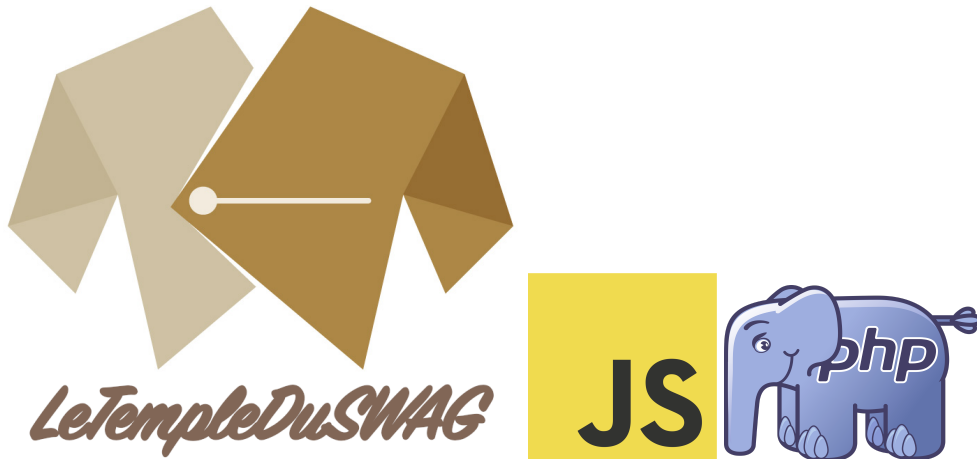


Web avancé – L2 MIASHS

TD 09 – LeTempleDuSwag – ajout d'un filtre



Précédemment dans LeTempleDuSWAG saison 1...

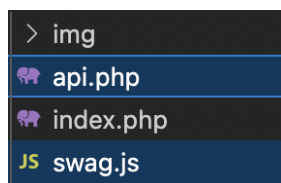
Je m'égare 😊.

Nous avons vu dans le TD précédent comment mettre en place une simple page montrant nos produits issus de la base de données. Nous y avons notamment des badges affichant les différents éléments du produit, dont le genre auquel il est destiné.

L'objectif de ce TD est de permettre de filtrer les produits par genre d'un simple clic sur le badge indiquant le genre. Cela peut paraître simple pour tout un TD, mais pour que ce type de fonctionnalité fonctionne bien sans transition, il est nécessaire d'utiliser... de l'AJAX ! 🤖

Pour cela nous ajouterons deux fichiers : un fichier PHP dédié aux requêtes de la base de

données et un fichier javascript.



Feuille de route du TempleDuSwag :

- ☒ ~~Mise en place du parcours des articles~~
- ☐ Ajout d'un filtre par genre en AJAX (ce TD ! 😊)
- ☐ Ajout des fonctionnalités de suppression et d'ajout avec formulaire dédié (TD suivant)
- ☐ Ajout d'une inscription et d'une connexion de l'utilisateur (semaine prochaine)

Objectif : mettre en place le parcours minimaliste du TempleDuSwag

Cible : les étudiants connaissant déjà Javascript, l'usage de l'AJAX et PHP

Durée : très variable, en fonction de votre aisance dans l'usage combiné des différentes notions. Rapide si vous êtes à l'aise avec Javascript 😊

Un conseil important : même si ce TD peut, à certains points, vous sembler facile, ne sautez pas d'étape !

Chaque exercice se distingue par ce logo  !

Dans ce TD, le code est intégré sous forme d'image afin que vous ne puissiez pas simplement le copier-coller.

Mise en place de l'API

Le terme API (Application Programming Interface) désigne une interface de programmation. Vous avez utilisé une API quand vous aviez fait une requête à mon serveur de films.

Dans cette section, et dans toute cette application, nous allons mettre en place une API toute simple qui ne respecte volontairement pas toutes les bonnes pratiques, afin de la rendre la plus simple possible. Toujours est-il que pour la mettre en place vous devez :

- Créer un fichier **api.php** qui contient uniquement du PHP (et donc commence directement par une balise PHP ouvrante).
- Mettre en place une connexion à la base de données comme vue lors du TD précédent (et oui, je ne vais pas faire de redite donc n'hésitez pas à vous référer aux TDs précédents 😊)

L'objectif est d'obtenir un fichier **api.php** globalement organisé comme ceci.

```
1  <?php
2
3  > function connect() { ...
24 }
25
26 > function parseResults($result) { ...
41 }
42
43 > function findAll($conn) { ...
47 }
48
49 > function findBy($conn, $filterColumn, $filterValue) { ...
53 }
54
55 $conn = connect();
56
57 > if (count($_GET)) { ...
61 > } else { ...
63 }
64
65 $conn->close();
66
67 ?>
```

L'usage de fonctions y est donc primordial !

Veuillez donc suivre les étapes suivantes :

- Créez une fonction de connexion **connect()** qui connecte la base de données et retourne l'objet instancié de la connexion (voir TD précédent).
- Créez une fonction **findAll()** qui accepte en argument l'objet de connexion et va exécuter la requête SQL suivante dans la base de données.

```
"SELECT * FROM vêtements"
```

Je vous laisse volontairement refaire cette étape par vous-même (nous en avons vu tous les éléments lors du précédent TD).

- Ajoutez-y un appel à une fonction **parseResults()** qui accepte en argument le résultat de la requête SQL. La fonction n'a donc pas besoin de retourner une valeur.
- Créez une fonction **parseResults()** qui prend en argument le résultat de la requête SQL effectuée sur la base de données. Il s'agit donc essentiellement de faire ce que nous avons fait lors du TD précédent à quelques changements près :
 - Commencez par ajouter le header (entête) à la réponse HTTP indiquant que le contenu est de type JSON comme suit :

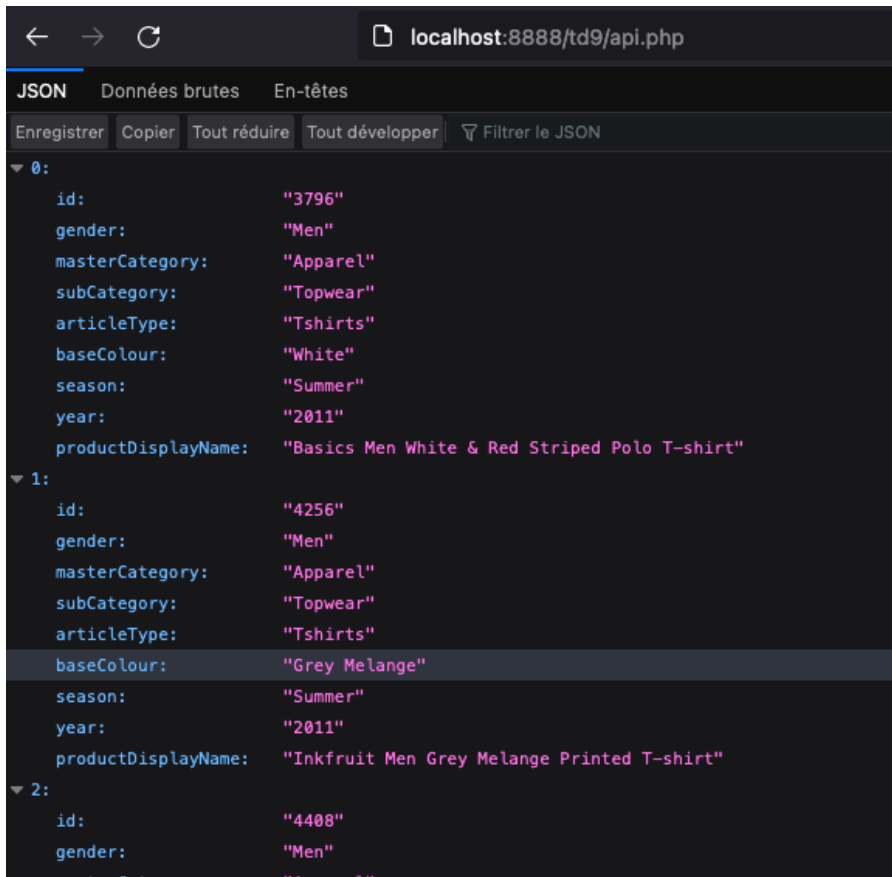
```
header("Content-Type: application/json");
```
 - Terminez la fonction par l'affichage dans le navigateur ([echo](#)) du tableau de tableaux associatifs au format JSON et la fin d'exécution du script. Pour ce faire, utilisez [json_encode\(\)](#) pour convertir ce tableau en JSON, et [exit\(\)](#) pour finir l'exécution du script.

Vous voyez ce que vous venez de faire ? Vous venez de mettre en place des fonctions qui interagissent parfois entre elles mais qui ne sont actuellement jamais exécutées : vous créez un code plus propre et plus organisé, plus modulaire. La fonction **parseResults()** sera ainsi utilisée de nouveau lors du filtre 😊 !

Il ne nous reste maintenant qu'à appeler la fonction de connexion **connect()** afin d'obtenir l'objet de connexion instancié, puis d'appeler la fonction **findAll()** pour récupérer et afficher tous les produits au format JSON.

N'oubliez pas de fermer la connexion en fin de script.

Maintenant rendez-vous dans votre navigateur à l'adresse de votre script pour afficher un beau contenu JSON (par exemple : <http://localhost:8888/td9/api.php>).



```

{
  "0": {
    "id": "3796",
    "gender": "Men",
    "masterCategory": "Apparel",
    "subCategory": "Topwear",
    "articleType": "Tshirts",
    "baseColour": "White",
    "season": "Summer",
    "year": "2011",
    "productDisplayName": "Basics Men White & Red Striped Polo T-shirt"
  },
  "1": {
    "id": "4256",
    "gender": "Men",
    "masterCategory": "Apparel",
    "subCategory": "Topwear",
    "articleType": "Tshirts",
    "baseColour": "Grey Melange",
    "season": "Summer",
    "year": "2011",
    "productDisplayName": "Inkfruit Men Grey Melange Printed T-shirt"
  },
  "2": {
    "id": "4408",
    "gender": "Men",
    "masterCategory": "Apparel",
    "subCategory": "Topwear",
    "articleType": "Tshirts",
    "baseColour": "White",
    "season": "Summer",
    "year": "2011",
    "productDisplayName": "Basics Men White & Red Striped Polo T-shirt"
  }
}
  
```

Pas très glamour n'est-ce pas ? Certes, mais c'est très efficace et c'est exactement ce dont nous avons besoin pour le traiter en Javascript plus tard 😊. N'hésitez pas à regarder l'onglet "Données brutes" pour voir le véritable JSON donné par votre script PHP.

Fonction de filtre

Passons désormais à l'ajout du filtre. Vous avez désormais tout ce qu'il faut pour le faire. Il ne manque qu'une requête SQL différente dans une nouvelle fonction dédiée lancée à partir des éléments fournis dans la requête GET du protocole de transfert hypertexte (HTTP).

Pour cela :

- Créez une fonction **findBy()** qui va permettre n'importe quel filtre par valeur sur n'importe quelle colonne de votre table. Cette fonction accepte donc trois arguments : l'objet de connexion, le nom de la colonne sur laquelle filtrer et la valeur à filtrer.
 - Cette fonction est ensuite similaire à **findAll()** à l'exception de la requête SQL qui devient `"SELECT * FROM vêtements WHERE gender = Men ;"`. À vous de la rendre dynamique, [notamment à l'aide de la préparation des statements](#) (comme vu en CM). N'oubliez pas d'utiliser la méthode `->get_result()` pour obtenir les résultats.
 - N'oubliez pas d'utiliser **parseResults()** qui va analyser et renvoyer les résultats.

- Conditionnez votre usage des fonctions **findAll()** et **findBy()** à la présence ou non de query dans la requête GET (queries HTTP sont égales à `http://monapi.php?query=value`). Suivez la logique suivante :
 - Si l'objet `$_GET` a une longueur (`count()`), alors je vérifie si les clés "filterby" et "filtervalue" sont bien présentes à l'aide de la fonction `isset()`. Si c'est bien le cas, j'appelle **findBy()** en lui fournissant les bons arguments.
 - Si l'objet `$_GET` n'a pas de query, alors j'appelle **findAll()**

Et voilà ! 🙌 Votre filtre est mis en place. Vérifiez dans le navigateur à l'aide de l'URL, par exemple comme ceci : <http://localhost:8888/td9/api.php?filterby=gender&filtervalue=women>

Pas mal hein ? 😊

Attention, désormais nous devons mettre à jour la page HTML à l'aide de Javascript.

Requête Asynchrone et mise à jour de la liste des produits affichés

Il nous faut désormais adapter le HTML légèrement et, surtout, créer le script Javascript qui va nous permettre d'exploiter au mieux notre API faite en PHP.

Pour cette section, vous aurez besoin de vos connaissances en Javascript et notamment de la manipulation du DOM et de l'utilisation de requêtes asynchrones (AJAX). Je ne vais pas tout détailler de nouveau, veuillez donc vous référer au TD du tableau de films ainsi qu'aux deux vidéos dédiées à ce sujet.

Nous devons donc commencer par modifier légèrement le code HTML généré par notre fichier **index.php**.

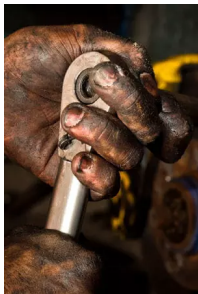
- Créez un fichier **swag.js** et liez le à **index.php** (avec la balise `script` en HTML)
- Pour chaque carte de produit (card), remplacez le badge indiquant le genre par un bouton (`button`) ayant pour classes "btn", "btn-secondary" et "btn-sm". Conservez l'icône dans le bouton.
 - Ajoutez à ce bouton un attribut "onclick" ayant pour valeur l'appel de la fonction **filterByGender()** avec donné en argument, le genre de l'article actuel.
- Il sera nécessaire de savoir quel élément récupérer pour modifier la section de l'affichage des produits. Pour ce faire, avant la boucle for en PHP, modifiez la balise **div** ayant les classes "row" et "gx-5" pour y ajouter un identifiant "produitsAffichage".

Bien entendu, cette fonction **filterByGender()** doit venir du Javascript. Mais pour le moment ce fichier ne contient... rien... nada... le néant 😞 Nous nous devons d'y remédier ! 😞

Mise en place du Javascript

ENFIN ! Vous l'attendiez ! Vous l'adooooorez ! Le langage tant apprécié, tant redouté, mais jamais, Ô grand jamais, dompté ! Le.... JAVASCRIPT 🤖🤖🤖🤖🤖🤖

Javascript va nous permettre de moderniser un peu l'apparence et le comportement de notre site. Un clic sur le bouton indiquant le genre et *SWISH!* la vue est filtrée ! bon, ça c'est d'un point de vue utilisateur. À nous de mettre les mains dans le cambouis.



Pour mettre en place ce JS (abréviation de Javascript) vous devez donc :

- une fonction qui va construire l'URL à l'aide des paramètres fournis en requête GET
- une fonction qui va traiter le contenu récupéré
- une fonction qui va envoyer une requête au serveur et traiter le contenu à l'aide de la fonction précédente
- une fonction qui va insérer le nouveau contenu dans une balise cible à l'aide de son identifiant
- une fonction qui va déclencher le filtre, en l'occurrence la fameuse fonction **filterByGender()** qui cache derrière elle tout ce travail 😊

Cela ne vous rappelle rien ? Même pas un petit peu, presque EXACTEMENT le code que nous avons mis en place pour la gestion du tableau des films ? 🤖 Regardons l'organisation du code attendu :

```

1
2 > function buildUrl(filterby, filterval){ ...
10 }
11
12 > async function loadJSONDoc(filterby, filterval){ ...
39 }
40
41 /**
42  * fonction qui parcourt le JSON analysé et met à jour le t
43  * @param {*} resp paramètre prenant le contenu de la répo
44  */
45 > function processContent(resp){ ...
53 }
54
55 > function insert(contenu, cible){ ...
12 }
13
14 > function filterByGender(value) { ...
16 }
  
```

Et oui dans ce code nous ne créons que des fonctions qui ne seront utilisées que si elles sont appelées lors de l'événement click (mis précédemment avec l'attribut "onclick"). Ce n'est pas pour rien que ce principe se nomme "Programmation événementielle" 😊

Voyons donc ce que doit faire chaque fonction. Implémentez chacune de ces fonctions en vous servant de ces indications ainsi que du TD sur les films et des deux vidéos si nécessaires.

buildUrl()

- construit l'adresse (URL) avec ses paramètres (?query=value) à partir des deux arguments.
- retourne l'URL construite

loadJSONDoc()

- fonction asynchrone
- utilise les deux arguments pour construire l'URL avec la fonction dédiée
- va utiliser [fetch\(\)](#) et [await](#) pour requêter un serveur distant à l'aide l'URL construite. La réponse est alors sauvegardée une fois récupérée.
- transforme la réponse sauvegardée en format JSON à l'aide de la méthode [.json\(\)](#)
- utilise la fonction pour traiter le contenu JSON si le statut de la réponse est "OK"; sinon une erreur est affichée

processContent()

- accepte la réponse au format JSON en argument
- va récupérer la balise cible (celle qui possède l'identifiant "produitsAffichage") et la met dans une variable
- vide le contenu HTML de cette balise (ainsi toutes les cartes de produits disparaissent)
- utilise la fonction dédiée à l'insertion des nouveaux éléments

insert()

La fonction qui vous rappellera les fameuses citrouilles et le jeu du serpent 😊

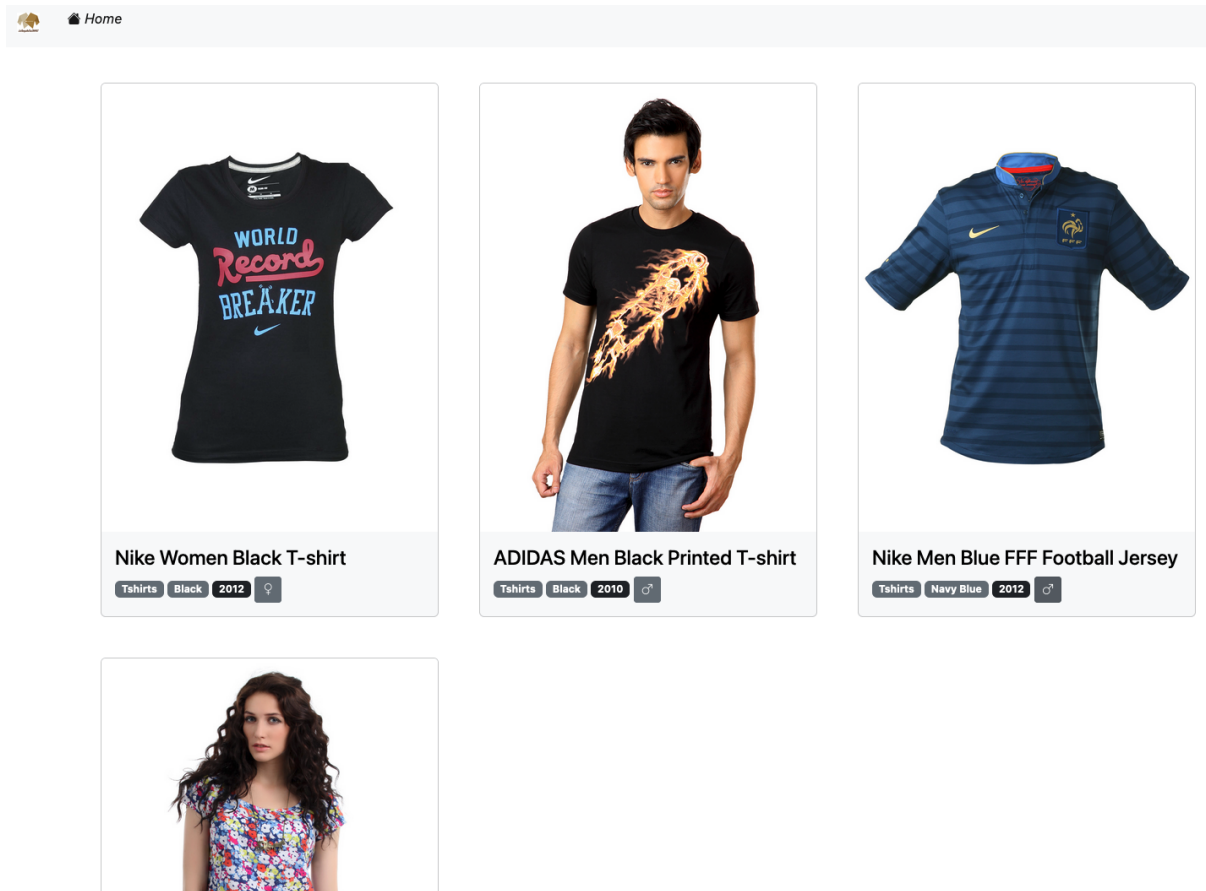
- accepte en arguments le contenu et la cible (la balise HTML en mémoire)
- pour chaque rangée dans le contenu, créer toute la hiérarchie HTML représentant la carte de ce contenu (l'équivalent de ce qui était fait en PHP mais cette fois-ci en JS)
- ajoute ce contenu à la cible pour qu'il apparaisse dans le navigateur

filterByGender()

- fonction dédiée à simplifier l'appel de toutes ces fonctionnalités
- accepte en argument la valeur du filtre (qui sera donc appliqué sur la colonne "gender")
- lance la fonction **loadJSONDoc()**

En suivant toutes ces consignes vous devriez avoir réussi à mettre en place le filtre par requête AJAX. Cela a-t-il été trop facile ? Trop difficile ? 😊

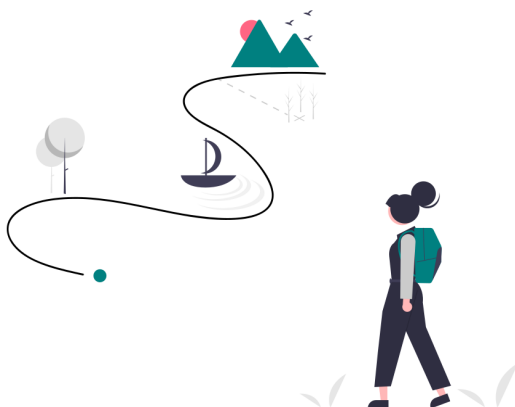
Vous devriez désormais pouvoir simplement cliquer sur le bouton ayant l'icône représentant le genre ciblé par le produit comme suit :



Et vous devriez pouvoir revenir à l'état initial en cliquant sur HOME ou les logos du super temple du SWAG 😊

Nous avons certes conditionné l'affichage et mis à jour ce dernier d'une belle manière avec une requête asynchrone. Cependant, qu'en est-il de réellement manipuler les données ? Ajouter un nouveau produit ? Supprimer un produit ?

N'ayez crainte ! Prochain cap : la suppression et l'ajout de produits !



C'est bon pour ce TD ! Youhou ! Vous en êtes venu à bout !



Liens utiles

- <https://www.php.net/manual/fr/>
- <https://www.w3schools.com/php/default.asp>
- <https://emojipedia.org/>
- <https://getbootstrap.com/docs/5.2/examples/>