Rechercher ou filtrer les données

Table des matières

Guy Verghote

Rechercher ou filtrer les données

Objectif de l'activité professionnelle

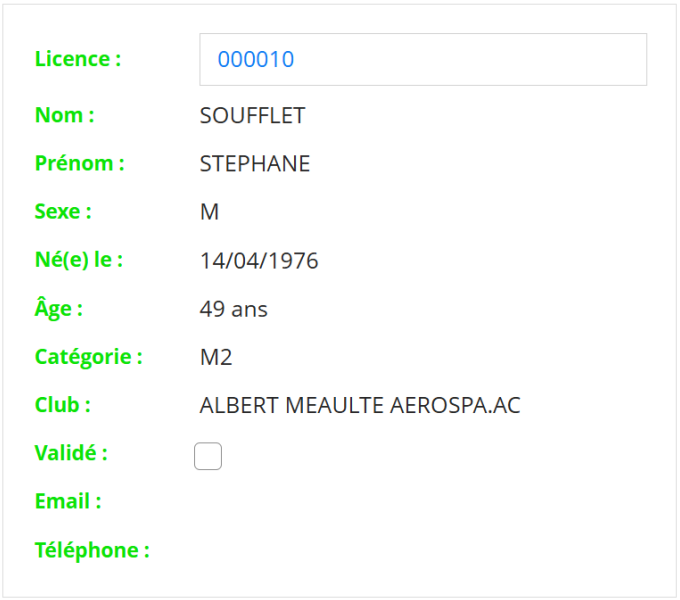
A la fin de cette activité, vous serez capable :

De mettre en place les différentes solutions pour rechercher et afficher un enregistrement De mettre en place les différentes solutions pour rechercher et afficher plusieurs enregistrements De mettre en place des filtres sur les données affichées

1. Recherche d'un licencié sur son numéro de licence

Répertoire recherche/getbyid

L'interface est composée de lignes découpées en 2 colonnes (une colonne fixe de 150 px et l'autre extensible)



Il comporte un champ de saisie 'licence'

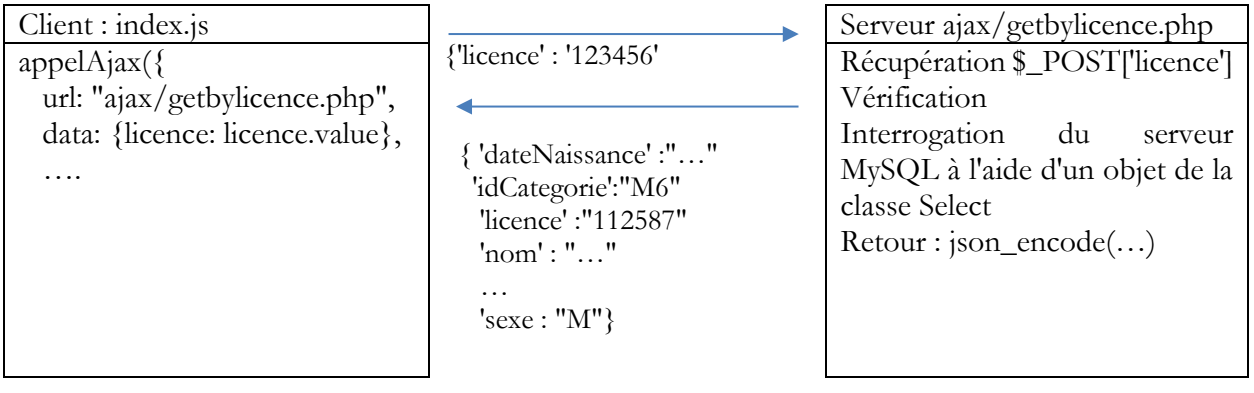
La mise en forme de l'interface utilise le mode 'grid' qui permet de répartir les éléments en ligne et en colonne

```
.fiche {
    display: grid;
    grid-template-columns: 150px 1fr;
    /* Label fixe, champ extensible */
    margin: 1rem auto;
    padding: 1rem;
    background: transparent;
    align-items: center;
    width: 100%;
    max-width: 600px;
    box-sizing: border-box;
    border: 1px solid #ddd;
}
```

L'âge est une donnée calculée à l'aide de la fonction getAge(date) contenu dans le composant fonction/date.js

La validation du champ de saisie par la touche Entrée déclenche un appel ajax vers le script ajax/getbylience.php en lui transmettant par la méthode POST le numéro de licence.

La réponse envoyée en cas de succès contient dans le format JSON les coordonnées du coureur



La méthode getByLicence($licence) de la classe Coureur permet de récupérer les coordonnées du licencié dont le numéro de licence est passé en paramètre :

```
Select licence, coureur.nom, prenom , sexe,
        date_format(dateNaissance, '%d/%m/%Y') as dateNaissanceFr, dateNaissance,
        idCategorie , club.nom AS nomClub, ffa, telephone, email, idClub
from coureur
      join club on coureur.idClub = club.id
 where licence = :licence
```

Elle utilise un objet Select et sa méthode getRow() pour récupérer l'enregistrement
La méthode getRow retourne la valeur nulle si le numéro de licence n'existe pas

1. Écrire la méthode getByLicence($licence) de la classe Coureur

1.1. Le script ajax/getbylicence.php

Le numéro de licence est transmis par la méthode POST, il est donc accessible depuis le tableau $_POST['licence']. Un premier contrôle doit vérifier que le numéro de licence est bien transmis.

```
if (!isset($_POST['licence'])) {
   Erreur::envoyerReponse("Numéro de licence non transmis", 'licence');
}
```

La valeur est alors stockée dans la variable $licence

```
$licence = $_POST['licence'];
```

Un second contrôle doit vérifier que sa valeur respecte le format attendu : 6 ou 7 chiffres
Pour cela nous utilisons une expression régulière : ^[0-9]{6,7}$

^ : Indique le début de la chaîne.
[0-9] : Correspond à n'importe quel chiffre de 0 à 9.
{6,7} : Spécifie qu'il doit y avoir exactement 6 ou 7 chiffres consécutifs.
$: Indique la fin de la chaîne.

```
if (!preg_match('/^[0-9]{6,7}$/', $licence)) {
   Erreur::envoyerReponse("Numéro de licence non conforme", 'licence');
}
```

Si le numéro de licence est conforme, le script doit appeler la méthode getByLicence de la classe Coureur afin de récupérer les coordonnées du client et les renvoyer dans le format JSON. Si la réponse renvoyée est false le numéro de licence n'a pas été trouvé. Il faut dans ce cas renvoyer un message d'erreur.

```
$ligne = Coureur::getByLicence($licence);
if (!$ligne) {
   Erreur::envoyerReponse("Numéro de licence inexistant", 'licence');
}
// envoi de la réponse
echo json_encode($ligne);
```

2. Compléter le script getbylicence.php

1.2. Le script index.php

Aucune donnée n'est à transmettre côté client, son contenu est totalement standard.

1.3. Le script index.js

Ce script prend en charge le contrôle de la valeur saisie dans le champ 'licence'.
Pour cela il utilise la fonction **filtrerLaSaisie**('licence', /[0-9]/) qui permet ici de n'accepter que la saisie de chiffres.
Cette fonction est contenue dans la bibliothèque de fonction 'fonction/controle.js'.
Derrière cette fonction se cache un événement keydown qui déclenche le traitement à chaque fois que l'utilisateur enfonce une touche dans le champ de saisie
L'appuie sur la touche Entrée dans le champ de saisie déclenche l'événement change

Le gestionnaire sur cet événement associé au champ de saisie doit déclencher l'appel de la fonction rechercher si le champ contient une valeur valide par rapport aux attributs de contrôle définis dans la balise <input>

required	Le champ ne peut pas être vide
pattern="[0-9]{6,7}"	Le champ doit comprendre 6 ou 7 chiffres
maxlength="7"	7 caractères au maximum
minlength="6"	6 caractères au minimum

En Javascript la méthode checkValidity() appliqué sur un objet HTMLInputElement retourne false si la valeur du champ ne respecte pas ces règles. La propriété validationMessage contient le message d'erreur correspondant ou une chaîne vide en cas d'absence d'erreur.

```
licence.onchange = function()  {
   if (this.checkValidity()) {
      rechercher(this.value);
   } else {
      afficherSousLeChamp('licence');
   }
};
```

La fonction afficherSousLeChamp permet d'afficher le message d'erreur (propriété validationMessage) sous le champ dont l'id est passé en paramètre.

La fonction rechercher(licence) réalise un appel Ajax du script getbycoureur.php en lui passant par la méthode POST le numéro de licence.
Cet appel est réalisé par la fonction **appelAjax** du composant fonction/ajax.js
La réponse en cas de succès (data) contient les coordonnées du coureur.

```
function rechercher(licence) {
   appelAjax({
      url: 'ajax/getbylicence.php',
      data: { licence: licence },
      success: (data) => afficher(data)
   });
}
```

La fonction afficher(data) afficher les coordonnées sur l'interface dans les balises correspondantes et fait perdre le focus au champ licence.

```
function afficher(coureur) {
   nom.innerText = coureur.nom;
   prenom.innerText = coureur.prenom;
   sexe.innerText = coureur.sexe;
   dateNaissance.innerText = coureur.dateNaissanceFr;
   nomClub.innerText = coureur.nomClub;
   idCategorie.innerText = coureur.idCategorie;
   age.innerText = getAge(coureur.dateNaissanceFr) + ' ans';
   licence.blur(); // retire le focus du champ de saisie
}
```

Lorsque le champ 'licence' reçoit le focus, il faut effacer les données :

```
licence.onfocus = function() {
   nom.innerText = '';
   …
   age.innerText = '';
   this.value = '';
}; }
```

3. Compléter le script index.js

2. Recherche d'un licencié sur son nom à l'aide d'un système d'autocomplétion

Répertoire recherche/getbyname

Il est difficile de connaitre le numéro de licence de tous les coureurs, en revanche on connait plus facilement leur nom et prénom. Proposer de sélectionner un coureur à partir d'une zone de liste n'est pas un bon choix car le nombre de coureurs est élevé.

En revanche, en mettant un place un système d'autocomplétion, il est possible de sélectionner un coureur sur son nom en saisissant simplement le début de son nom.

Nous allons utiliser le composant autoComplete.js (https://tarekraafat.github.io/autoComplete.js/#/) pour mettre en place le système d'autocomplétion.

Comme tous les composants ce dernier est stocké dans le répertoire composant du site afin d'éviter la nécessité d'avoir un accès internet en phase de production.

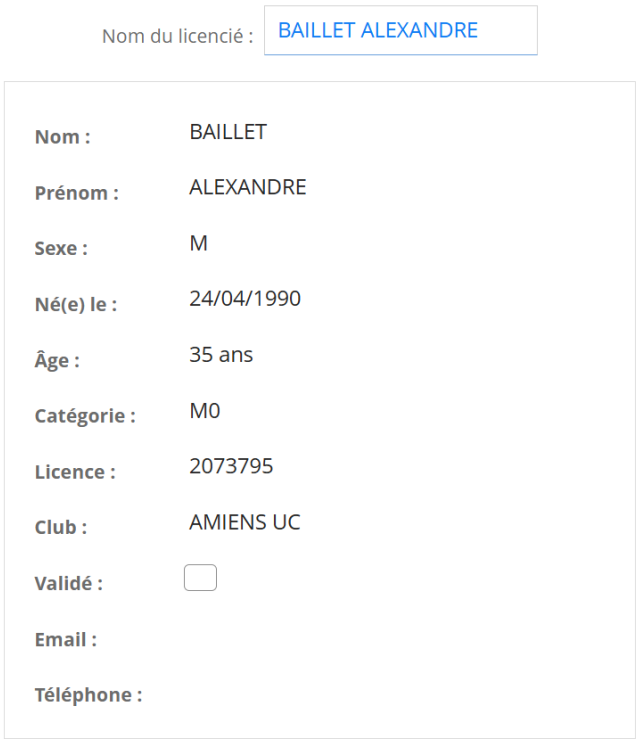
L'utilisation de ce composant peut être généralisé en utilisant le composant fonction/autocomplete.js

Documentation : autocomplete.pdf et https://tarekraafat.github.io/autoComplete.js/#/

Rechercher ou filtrer les données

L'interface comporte un champ 'nomR' permettant de saisir les premières lettres du nom.
Pour réussir à fixer la largeur des deux colonnes, le mode display 'grid' est utilisé comme dans l'interface précédente

De cette façon il suffit d'enchainer les balises label et input




```
<div class="d-flex flex-row justify-content-center">
   <div class="form-recherche">
      <label for="nomR">Nom du licencié :</label>
      <input id="nomR" type="text" pattern="^[a-zA-Z ]+$" placeholder="Saisir les premières
lettres"
            autocomplete="off">
   </div>
</div>
<div class="fiche">
   <label for="nom" >Nom : </label>
   <output id="nom" class="form-control"></output>
    …
   <label for="telephone" >Téléphone :</label>
   <output id="telephone" class="form-control"></output>
</div>
```

La méthode getByNomPrenom($nomPrenom) de la classe Coureur retourne les coureurs dont le nom-prénom contient la valeur passée en paramètre. Le tableau résultant sera utilisé comme source de la zone d'autocomplétion.

```
public static function getByNomPrenom(string $nomPrenom): array  {
      $sql = <<<SQL
           Select licence, coureur.nom, prenom,  concat(coureur.nom, ' ', prenom) AS nomPrenom,
sexe,
                  date_format(dateNaissance, '%d/%m/%Y') as dateNaissanceFr, dateNaissance,
                  idCategorie , club.nom AS nomClub, ffa, telephone, email, idClub
           from coureur
             join club on coureur.idClub = club.id
          where concat(coureur.nom, ' ', prenom) like :terme
          order by coureur.nom, prenom
          limit 10
SQL;
      $select = new Select();
      // on ajoute % pour la recherche
      $terme = "%$nomPrenom%";
      return $select->getRows($sql,['terme' => $terme]);
 }
```

Pour rechercher les enregistrements dont la valeur de nomPrenom contient $nomPrenom il faut utiliser l'opérateur 'like' associé au caractère spécial % qui signifie "n'importe quelle suite de caractères". Il faut placer ce caractère au début et à la fin de la valeur

4. Compléter la méthode de la classe Coureur

2.1. Le script ajax/getbynomprenom.php

Il reçoit la valeur recherchée par la méthode GET :
Il appelle la méthode Coureur:getByNomPrenom afin de retourner les coureurs correspondants.
Si la valeur recherchée n'est pas précisée ou si elle contient des caractères autres que les lettres non accentuées, le script retourne un tableau vide.

5. Réaliser les tests fonctionnels de ce script sous Postman

Exemple : http://gestion/recherche/getbyname/ajax/getbynomprenom.php?search=a

2.2. Le script index.php

Il charge aussi le composant autocomplete.js en l'ajoutant dans la variable $head

```
<script src="/composant/autocomplete/autocomplete.min.js"></script>
<link rel="stylesheet" href="/composant/autocomplete/autocomplete.css">
```

6. Compléter le script index.php

2.3. Le script index.js

Il doit mettre en œuvre le composant autoComplete.js qui permet d'ajouter une zone de complétion sous un champ de saisie.

Documentation à lire : **autoComplete.pdf**
Exemple à étudier : **http://exemple/autocompletion/**

Pour simplifier sa mise en œuvre, le composant fonction/autocomplete.js propose une méthode initAutocomplete qui assure l'initialisation de ce composant

```
import {initAutoComplete} from "/composant/fonction/autocomplete.js";
```

4 paramètres permettent de personnaliser la mise en place du composant

```
initAutoComplete({
   selector: "#nomR",
   fetchUrl: "ajax/getbynomprenom.php",
   searchKey: "nomPrenom",
   onSelection: (selection) => {
      nomR.value = selection.nomPrenom;
      afficher(selection);
   }
});
```

initAutoComplete({...}) : appel d'une fonction d'initialisation encapsulant la configuration et l'instanciation du composant autoComplete.js, utilisé ici pour fournir une auto-complétion asynchrone basée sur une requête HTTP.

selector: "#nomR" : le champ cible sur lequel l'autocomplétion est appliquée (<input id="nomR">). Ce sélecteur CSS est utilisé pour attacher dynamiquement le comportement d'auto-complétion.

fetchUrl: "ajax/getbynomprenom.php" : script backend qui retourne les suggestions en fonction de la chaîne tapée. Ce script retourne un tableau JSON d'objets.

searchKey: "nomPrenom" : indique quelle propriété des objets retournés doit être utilisée pour la recherche et pour l'affichage dans la liste déroulante.

onSelection: (...) => { ... } : callback exécuté lorsque l'utilisateur sélectionne une suggestion :

7. Compléter le script index.js

3. Recherche des licenciés appartenant à une catégorie

Répertoire rechercher/categorie

L'interface graphique comporte une zone de liste <select id='idCategorie'> et une balise <table>

L'usage de la classe 'masquer' permet de ne pas afficher certaines colonnes sur les 'petits écrans' (Cf. feuille de style)

Catégorie Minime Nombre de coureurs : **9**

Licence	Nom	Sexe	Né(e) le	Catégorie	Club
1959344	Carpentier Corentin	M	19/03/2010	MI	ALBERT MEAULTE AEROSPA.AC
2077084	Debusscher Jean	M	26/07/2010	MI	AMIENS UC
2087820	Dos Santos Mael	M	15/02/2011	MI	AMIENS UC
1993205	Dufour Louise	F	13/03/2010	MI	SPORTING CLUB ABBEVILLOIS ATHL
2080118	Kozak Dolphin Jean Edouard	M	07/04/2010	MI	AMIENS UC
1380403	Legris Clemence	F	05/05/2010	MI	SPORTING CLUB ABBEVILLOIS ATHL
2071812	Poquet Charlyne	F	13/05/2010	MI	AMICALE DU VAL DE SOMME
2037360	Poutrain Philippine	F	18/07/2010	MI	US CAMON
2012775	Vasseur Adele	F	17/02/2010	MI	ALBERT MEAULTE AEROSPA.AC

Nous souhaitons récupérer tous les licenciés appartenant à une catégorie. Une zone de liste est proposée afin de sélectionner la catégorie.

Concernant les données, nous avons besoin de :
- Récupérer la liste des catégories (id et libellé) afin d'alimenter la zone de liste.
- Récupérer les coureurs appartenant à la catégorie sélectionnée.

L'interface doit être responsive, les colonnes 'sexe' et 'date de naissance' ne seront pas affichées sur smartphone.

La méthode getListe de la classe Categorie retourne les catégories

```
select id, nom from categorie  order by ageMin;
```

La méthode getByCategorie retourne les licenciés appartenant à la catégorie passée en paramètre

```
Select licence, coureur.nom, prenom , concat(coureur.nom, ' ', prenom) as nomPrenom,  sexe,
        date_format(dateNaissance, '%d/%m/%Y') as dateNaissanceFr,
        club.nom AS nomClub, idCategorie
from coureur
      join club on coureur.idClub = club.id
where idCategorie = :idCategorie
order by coureur.nom, prenom;
```

8. Compléter les méthodes de la classe Categorie

3.1. Le script ajax/getlescoureurs.php

Le code de la catégorie est transmis par la méthode POST.
Un premier contrôle doit vérifier que ce code est bien transmis.

```
if (!isset($_POST['idCategorie'])) {
    Erreur::envoyerReponse("L'identifiant de la catégorie n'est pas transmis", 'global');
}
```

La valeur est alors stockée dans la variable $idCategorie

```
$idCategorie = $_POST['idCategorie'];
```

Un second contrôle doit vérifier que sa valeur respecte le format attendu : une lettre majuscule, suivie soit d'une autre lettre majuscule, soit d'un chiffre allant de 0 à 10 :

```
if (!preg_match("/^[A-Z](?:[A-Z]|\d|10)$/", $idCategorie)) {
    Erreur::envoyerReponse("L'identifiant de la catégorie n'est pas conforme", 'global');
}
```

Pour cela nous utilisons une expression régulière : ^[A-Z](?:[A-Z]|\d|10)$

^ : Indique le début de la chaîne.
[A-Z] : Correspond à une lettre majuscule (de A à Z).
(?: ...) : Crée un groupe non capturant, ce qui signifie que la correspondance à l'intérieur du groupe n'est pas enregistrée dans les résultats. Cela permet d'appliquer une logique mais de ne pas garder les correspondances des groupes.
[A-Z] : Correspond à une lettre majuscule (de A à Z).
|\d|10)}: Le symbole | signifie "ou". Ici, on dit soit qu'il peut y avoir une autre lettre majuscule ([A-Z]), soit un chiffre (\d), soit le chiffre 10.
$: Indique la fin de la chaîne.

Si le code de la catégorie est conforme, le script doit appeler la méthode getByCategorie($idCategorie) de la classe Coureur afin de récupérer les coordonnées des coureurs et les renvoyer dans le format JSON.

```
echo json_encode(Coureur::getByCategorie($idCategorie));
```

9. Compléter le script ajax/getlescoureurs.php

3.2. Le script index.php

Il doit transmettre côté client les données renvoyées par la méthode Categorie::getListe()

10.Compléter le script index.php

3.3. Le script index.js

Il faut commencer par alimenter la zone de liste des catégories à partir des données transmises par le script index.php et appeler la fonction getLesCoureurs sur la catégorie actuellement sélectionnée

```
for (const element of data) {
   idCategorie.add(new Option(element.nom, element.id));
}
getLesCoureurs(idCategorie.value);
```

La sélection d'une valeur dans la zone de liste doit aussi déclencher l'appel de la fonction getLesCoureurs

```
idCategorie.onchange = () => {
      getLesCoureurs(idCategorie.value);
};.
```

La fonction getLesCoureurs() appelle le script ajax/getlescoureurs.php en lui passant la valeur de la balise <select id='idCategorie'> (idCategorie.value).

```
function getLesCoureurs(idCategorie) {
   appelAjax({
      url: 'ajax/getlescoureurs.php',
      data: {idCategorie: idCategorie},
      success: afficher
   });
}
```

La réponse du serveur est automatiquement transmise à la fonction afficher.
Pour une meilleure compréhension on peut écrire : **success : (data) => afficher(data)**

La fonction afficher(lesCoureurs) alimente la balise et les lignes de la balise <tbody id='lesLignes'

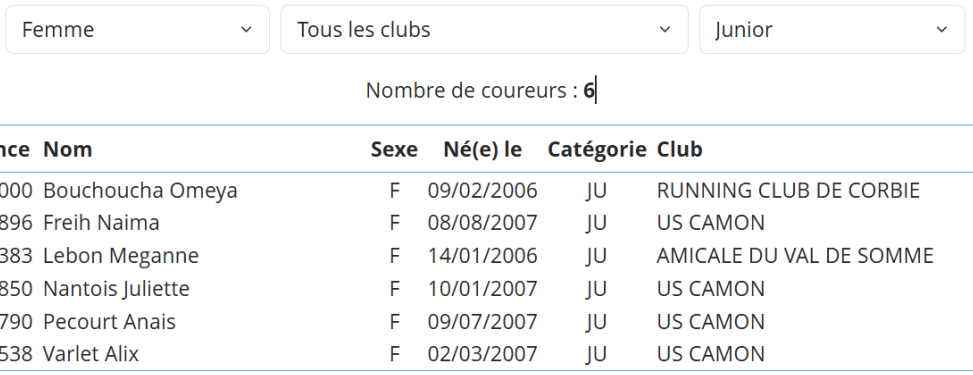
```
function afficher(lesCoureurs) {
   lesLignes.innerHTML = '';
   nb.innerText = lesCoureurs.length;

   for (const coureur of lesCoureurs) {
      const tr = lesLignes.insertRow();
      tr.style.verticalAlign = 'middle';
      tr.id = coureur.id;

      …

      // Colonne : Club
      td = tr.insertCell();
      td.innerText = coureur.nomClub;
      td.style.textAlign = 'left';
   }
}
```

11. Compléter le script index.js

4. Recherche sur la catégorie, le sexe et le club

Répertoire rechercher/recherchemultiple

L'interface graphique comporte trois zones de liste et une balise <table>



Licence	Nom	Sexe	Né(e) le	Catégorie	Club
1432000	Bouchoucha Omeya	F	09/02/2006	JU	RUNNING CLUB DE CORBIE
1265896	Freih Naima	F	08/08/2007	JU	US CAMON
1862383	Lebon Meganne	F	14/01/2006	JU	AMICALE DU VAL DE SOMME
1633850	Nantois Juliette	F	10/01/2007	JU	US CAMON
2016790	Pecourt Anais	F	09/07/2007	JU	US CAMON
1974538	Varlet Alix	F	02/03/2007	JU	US CAMON

Concernant les données, nous avons besoin de :

- Récupérer la liste des catégories (id et libellé) afin d'alimenter la zone de liste.
- Récupérer la liste des clubs (id, nom) afin d'alimenter la zone de liste ➔ Club::getListe()
- Récupérer les coureurs répondant aux critères ➔ méthode Coureur::getBySexeClubCategorie().

12. Écrire la méthode getListe() de la classe Club qui retourne les 'id' et 'nom' des clubs ordonnées sur le nom.

La méthode geBySexeClubCategorie($sexe = '*', $idClub = '*', $idCategorie = '*') de la classe Coureur doit retourner dans un tableau les coureurs vérifiant les critères dont la valeur n'est pas '*'

La requête doit donc être généré dynamiquement afin d'y ajouter des clauses 'and' pour chaque critères 'renseigné'.

Dans ce cas il est intéressant d'utiliser l'ancienne syntaxe de la jointure car elle garantit la présence de la clause where dans la partie fixe de la requête.

En effet si nous utilisons la notation 'join' il n'y a pas de clause where or pour ajouter une clause 'and' il faut une clause where.

Une astuce simple suffit pour contourner le problème : on ajoute une condition where toujours vrai (where 1 = 1)

```
public static function getBySexeClubCategorie($sexe = '*', $idClub = '*', $idCategorie = '*'): array
{
        $lesParametres = [];
        $sql = <<<EOD
            Select licence, coureur.nom, prenom , sexe,
                    date_format(dateNaissance, '%d/%m/%Y') as dateNaissanceFr,
                    club.nom AS nomClub, idCategorie
            FROM coureur , club
            where coureur.idClub = club.id
EOD;
```

```
        if ($sexe !== '*') {
            $sql .= " and sexe = :sexe";
            $lesParametres['sexe'] = $sexe;
        }
        if ($idClub !== '*') {
            $sql .= " and idClub = :idClub";
            $lesParametres['idClub'] = $idClub;
        }
        if ($idCategorie !== '*') {
            $sql .= " and idCategorie = :idCategorie";
            $lesParametres['idCategorie'] = $idCategorie;
        }
        $sql .= " ORDER BY coureur.nom, prenom;";
        $select = new Select();
        return $select->getRows($sql, $lesParametres);
    }
```

13. Écrire la méthode getBySexeClubCategorie de la classe Coureur

4.1. Le script index.php

Nous avons besoin de récupérer 2 jeux d'enregistrements pour alimenter les données de l'interface :

```
$lesCategories = json_encode(Categorie::getListe());
$lesClubs = json_encode(Club::getListe());

$head = <<<EOD
<script>
    let lesCategories = $lesCategories;
    let lesClubs = $lesClubs;
</script>
EOD;
```

14.Compléter le script index.php

4.2. Le script ajax/getlescoureurs.php

Un premier contrôle doit vérifier que les trois paramètres sont bien transmis.

Les valeurs seront alors stockées dans les 3 variables $sexe, $idClub et $idCategorie

Un second contrôle doit vérifier que les valeurs respectent leur format attendu :
Catégorie : lettre majuscule, suivie soit d'une autre lettre majuscule, soit d'un chiffre allant de 0 à 10 (ou *)
Club : 6 chiffres commençant par 080 ou '*'
Sexe : 'M', 'F' ou '*'

Tous les messages seront de type 'global'. Ils seront affichés dans la balise

Si tout est conforme, le script doit appeler la méthode getBySexeClubCategorie($sexe, $idClub,$idCategorie) de la classe Coureur afin de récupérer les coordonnées des coureurs et les renvoyer dans le format JSON.

15. Compléter le script ajax/getlescoureurs.php

4.3. Le script index.js

Il doit alimenter les zones de liste et ajouter sur chacune d'elle un gestionnaire sur l'événement change qui appelle la fonction filtrer().

La fonction filtrer doit faire un appel Ajax du script ajax/getlescoureurs.php en passant en paramètre le sexe, le club et la catégorie.

```
function filtrer() {
   appelAjax({
      url: 'ajax/getlescoureurs.php',
      data: {
         idCategorie: idCategorie.value,
         idClub: idClub.value,
         sexe: sexe.value
      },
      success: afficher
   });
}
```

En cas de succès la fonction afficher() est appelée (version identique à la précédente fonctionnalité)

16.Compléter le script index.js

5. Filtrer sur la catégorie, le sexe et le club

Répertoire rechercher/filtragemultiple

Il s'agit de la **même interface** mais cette fois ci tous les coureurs sont initialement chargés dans la variable javascript data qui sera alors filtrée côté client pour réponde aux critères de sélection.
Le tableau data pourrait être filtré en lui appliquant la méthode filter

```
function filtrer() {
   const categorie = idCategorie.value;
   const club = idClub.value;
   const sexeValue = sexe.value;
   const lesCoureurs = data.filter(coureur => {
      const matchCategorie = categorie === '*' || coureur.idCategorie === categorie;
      const matchClub = club === '*' || coureur.idClub === club;
      const matchSexe = sexeValue === '*' || coureur.sexe === sexeValue;

      return matchCategorie && matchClub && matchSexe;
   });
   afficher(lesCoureurs);
}
```

C'est fort pratique mais très peu performant car cela crée à chaque fois un nouveau tableau. Il faut donc éviter de recourir à cette méthode.

5.1. Le script index.php

Nous avons besoin de récupérer 3 jeux d'enregistrements pour alimenter les données de l'interface.

```
$lesCategories = json_encode(Categorie::getListe());
$lesClubs = json_encode(Club::getListe());
$lesCoureurs = json_encode(Coureur::getAll());

$head = <<<HTML
<script>
      const lesCategories = $lesCategories;
      const lesClubs = $lesClubs;
       const lesCoureurs = $lesCoureurs;
</script>
HTML;
```

17.Compléter le script index.php

5.2. Le script index.js

Il doit alimenter les zones de liste et ajouter sur chacune d'elle un gestionnaire sur l'événement change qui appelle la fonction afficher

La fonction afficher() affiche les enregistrements du tableau global 'lesCoureurs' qui vérifient les conditions sélectionnées dans les zone de liste

```
function afficher() {
   lesLignes.innerHTML = '';
   let compteur = 0;

   for (const coureur of lesCoureurs) {
      if (!filtrer(coureur)) {
         continue;
      }
      compteur++;

      const tr = lesLignes.insertRow();
      tr.style.verticalAlign = 'middle';
      tr.id = coureur.id;

      // Colonne : licence
      …
   }
   nb.innerText = compteur;
}
```

La fonction filtrer() vérifie si l'enregistrement passé en paramètre vérifie la condition exprimée par les valeurs sélectionnées dans les zones de liste

```
function filtrer(coureur) {
   return (idCategorie.value === '*' || coureur.idCategorie === idCategorie.value) &&
          (idClub.value === '*' || coureur.idClub === idClub.value) &&
          (sexe.value === '*' || coureur.sexe === sexe.value);
}
```

Le licencié doit être affiché si la valeur de la zone de liste vaut "*" ou si elle correspond à la propriété correspondant dans l'objet coureur.

18. Compléter le script index.js

6. Filtrer à l'aide d'un champ de recherche applicable sur toutes les colonnes

Répertoire rechercher/filtrer

On souhaite afficher les coureurs mais pouvoir les filtrer à partir d'un champ de recherche qui recherche dans les colonnes 'licence', 'nom', 'dateNaissanceFr' ou 'club' si la valeur de la colonne contient la valeur saisie dans le champ de recherche.

L'interface graphique comporte un champ de recherche 'search' de classe 'recherche', le reste est identique aux modules précédents.

Nombre de coureurs : **6**

Licence	Nom	Sexe	Né(e) le	Catégorie	Club
1558815	Dubois Frederic	M	18/11/1968	M4	VYTAJOG
1115977	Dubreucq Antoine	M	01/11/1983	M1	AMICALE DU VAL DE SOMME
1115979	Dubreucq Ludovic	M	22/06/1982	M1	AMICALE DU VAL DE SOMME
1184435	Dubuc Francis	M	09/09/1965	M5	AMICALE DU VAL DE SOMME
2044892	Dubuc Marie	F	22/02/1977	M2	AMICALE DU VAL DE SOMME
1388149	Saulduboi Keller Anne Lise	F	25/09/1978	M2	ESPRIT RUN

Concernant les données, nous avons besoin de :

- Récupérer la liste des coureurs ➔ Coureur.getAll().

Le script index.php

Il appelle la méthode Coureur::getAll() et transfère le résultat (data) côté client

Le script index.js

La recherche sera lancée sur l'événement input du champ 'search' de manière à se lancer à chaque fois que l'utilisateur saisit un caractère.

```
search.oninput = afficher;
```

La fonction afficher est identique à la version précédente, il faut adapter la fonction filtrer

```
function filtrer(coureur) {
   if (search.value === '') {
      return true; // Si le champ de recherche est vide, on affiche tous les coureurs
   }
   return coureur.nomPrenom.toLowerCase().includes(search.value.toLowerCase()) ||
         coureur.licence.includes(search.value) ||
         coureur.dateNaissanceFr.includes(search.value) ||
         coureur.nomClub.toLowerCase().includes(search.value.toLowerCase());
}
```

.toLowerCase() assure que la comparaison ne soit pas sensible à la casse.
.includes() vérifie si la valeur de la ligne contient la chaîne de recherche.

19. Compléter le script index.js

7. Recherche sur des critères imbriqués

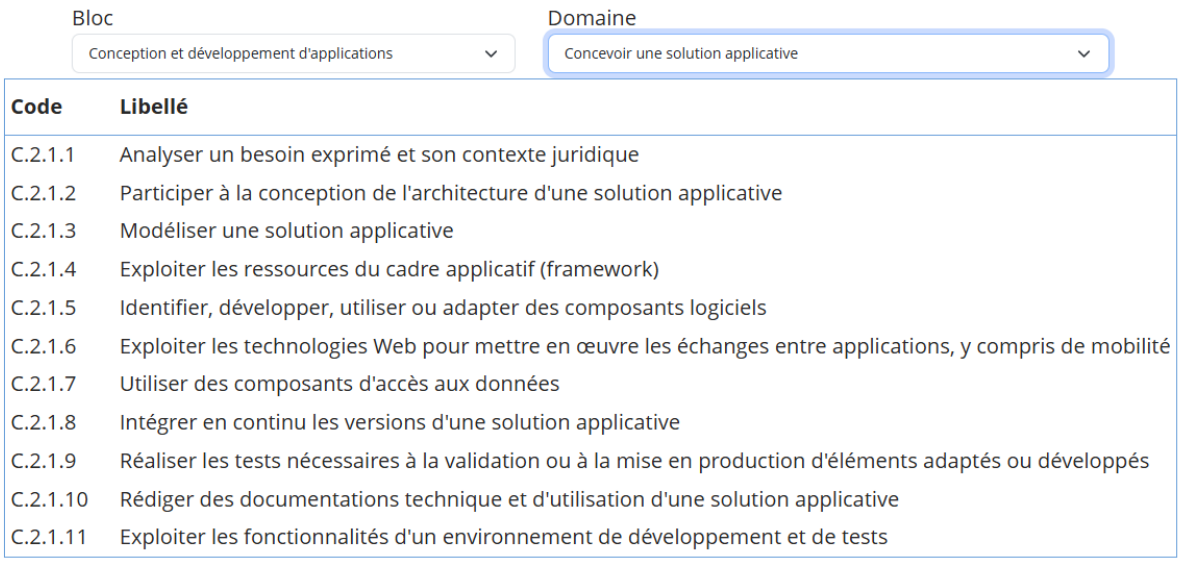
Répertoire recherche/competence

Certains critères de recherche influent sur d'autres critères. On parle alors de critères imbriqués. C'est le cas notamment dans le référentiel SIO où les compétences sont organisées en domaines d'activité qui appartiennent à un bloc du référentiel.

Autrement dit :
À un bloc correspond une liste de domaines.
À un domaine correspond une liste de compétences.

On souhaite afficher les compétences du référentiel pour un bloc, ou un domaine dans un bloc.
La première zone de liste doit afficher les blocs et la seconde doit afficher les domaines correspondant au bloc sélectionné.
Le remplissage de la seconde zone de liste doit donc être dynamique : La sélection d'une valeur dans la première zone modifie les valeurs de la seconde.

L'interface graphique



L'interface positionne deux balises <select id ='idBloc'> et <select id='idDomaine'> sans leur option.
L'entête du tableau est défini en y ajoutant les attributs 'data' permettant la mise en place du tri des colonnes

```
<table id='leTableau' style="margin: auto; max-width: 1200px">
   <thead class="">
   <tr>
      <th data-champ="code" data-type="text" style="width: 100px" >Code</th>
      <th data-champ="libelle" data-type="text">Libellé</th>
   </tr>
   </thead>
   <tbody id="lesLignes"></tbody>
</table>.
```

7.1. Les méthodes de la classe Competence (classemetier/competence.php)

Pour alimenter cette interface nous avons besoin des trois méthodes suivantes de la classe Competence :

- getLesBlocs() pour récupérer les blocs

```
select id, libelle from bloc order by libelle;
```

- getLesDomaines($idBloc) pour récupérer les domaines associés à un bloc

```
select idDomaine, libelle from domaine where idBloc = :idBloc  order by libelle;
```

- getLesCompetences($idBloc, $idDomaine = '*') pour récupérer les compétences associées à un bloc ou à un domaine d'un bloc

```
public static function getLesCompetences(string $idBloc, string $idDomaine = '*'): array  {
      $lesParametres['idBloc'] = $idBloc;
      $sql = <<<SQL
         select id, concat('C.', idBloc, '.', idDomaine, '.', idCompetence) as code, libelle
         from competence
         where idBloc = :idBloc
SQL;
      // prise en compte du domaine si l'idDomaine n'a pas la valeur par défaut
      if ($idDomaine !== '*') {
         $sql .= " and  idDomaine = :idDomaine";
         $lesParametres['idDomaine'] = $idDomaine;
      }
      $select = new Select();
      return $select->getRows($sql, $lesParametres);
   }
```

La condition sur le domaine est facultative ($idDomaine === '*'). Il faut donc générer une partie de la requête dynamiquement en lui ajoutant une clause "and"

20.Compléter les méthodes de la classe Competence

7.2. Le script ajax/getlesdomaines.php

Le script ajax/getlesdomaines.php récupère les domaines correspondant au bloc transmis en paramètre par la méthode POST.

Le code du bloc est envoyé via la méthode POST. Un premier contrôle doit vérifier que le paramètre est bien présent.

```
if (!isset($_POST['idBloc'])) {
    Erreur::envoyerReponse("L'identifiant du bloc n'a pas été transmis.", 'global');
}
```

Si le paramètre est valide, sa valeur sera stockée dans la variable $idBloc

```
$idBloc = $_POST['idBloc'];
```

Un second contrôle doit vérifier que la valeur respecte le format attendu : un nombre entier.

```
if (!preg_match("/[0-9]+/", $idBloc)) {
    Erreur::envoyerReponse("L'identifiant du bloc n'est pas valide.", 'global');
}
```

Si toutes les vérifications sont conformes, le script appelle la méthode Competence::getLesDomaines($idBloc) de la classe Competence afin de récupérer les compétences et de les renvoyer au format JSON.

```
echo json_encode(Competence::getLesDomaines($idBloc));
```

21.Compléter le script ajax/getlesdomaines.php

7.3. Le script ajax/getlescompetences.php

Le script ajax/getlescompetences.php récupère les compétences correspondant au bloc et au domaine transmis en paramètre par la méthode POST.

Après les vérifications, il appelle la méthode Competence::getLesCompetences($idBloc, $idDomaine) et renvoie la réponse au format JSON.

Les vérifications concernant le bloc sont identiques à celles du script précédent.
Le domaine n'est pas obligatoire ; s'il n'est pas transmis, la valeur '*' sera utilisée :

```
$idDomaine = isset($_POST['idDomaine']) ? $_POST['idDomaine'] : '*';
```

Toutefois, dans tous les cas, une vérification de la conformité de sa valeur doit être effectuée : elle doit être soit un chiffre, soit '*'.

Si toutes les vérifications sont conformes, le script doit appeler la méthode getLesCompetences($idBloc, $idDomaine) de la classe Competence pour récupérer les compétences et les renvoyer au format JSON.

22.Compléter le script ajax/getlescompetences.php

7.4. Le script index.php

Il récupère les blocs par l'appel de la méthode Competence::getLesBlocs() pour les transmettre côté client afin de remplir la zone de liste correspondante.

7.5. Le script index.js

Il doit :

- Alimenter la zone de liste des blocs

```
for (const bloc of data) {
   idBloc.appendChild(new Option(bloc.libelle, bloc.id));
}
```

- Lancer la fonction getLesdomaines

```
getLesDomaines(idBloc.value);
```

- Relancer la fonction getLesdomaines lorsqu'un autre bloc est sélectionné

```
idBloc.onchange = function() {
   getLesDomaines(this.value);
};
```

- Lancer la fonction getLesCompetences lorsqu'un domaine est sélectionné

```
idDomaine.onchange = function() {
   getLesCompetences(idBloc.value, this.value);
};
```

La fonction getLesDomaines doit récupérer par un appel Ajax les domaines du bloc sélectionné afin d'alimenter la zone de liste des domaines et lancer la fonction getLesCompetences.

```
function getLesDomaines(idBloc) {
   appelAjax({
      url: 'ajax/getlesdomaines.php',
      data: {
         idBloc: idBloc,
      },
      success: (data) => {
         // réinitialisation de la liste des domaines
         idDomaine.innerHTML = '';
         // ajout de l'option 'Tous les domaines'
         idDomaine.add(new Option('Tous les domaines', '*'), 0);
         // ajout des domaines contenu dans la réponse data
         for (const domaine of data) {
            idDomaine.add(new Option(domaine.libelle, domaine.idDomaine));
         }
         // récupération des compétences du bloc et du domaine sélectionnés
         getLesCompetences(idBloc, idDomaine.value);
      }
   });
}
```

La fonction getLesCompetences doit récupérer par un appel Ajax les compétences en passant en argument le bloc et le domaine puis les afficher sur l'interface

```
function getLesCompetences(idBloc, idDomaine) {
    appelAjax({
        url: 'ajax/getlescompetences.php',
        data: {
            idBloc: idBloc,
            idDomaine: idDomaine,
        },
        success: data => {
            // réinitialisation de la liste des compétences
            lesCompetences = data;
            afficher(); }
    });
}
```

La variable globale lesCompetences permet de conserver les compétences à afficher mais aussi à définir la source d'information à trier (Cf. activerTri).

La fonction afficher() affiche les compétences sur l'interface

```
function afficher() {
    lesLignes.innerHTML = '';
    for (const competence of lesCompetences) {
        const tr = lesLignes.insertRow();
        tr.style.verticalAlign = 'middle';

        // Colonne : code
        tr.insertCell().innerText = competence.code;

        // Colonne Libellé
        tr.insertCell().innerText = competence.libelle;
    }
}
```

Le tri est possible en cliquant sur les entêtes de colonne :

```
activerTri({
    idTable: "leTableau",
    getData: () => lesCompetences,
    afficher: afficher
});
```

23. Compléter le script index.js