La consultation des données

Table des matières

Guy Verghote

La consultation des données

Objectif de l'activité professionnelle

A la fin de cette activité, vous serez capable :

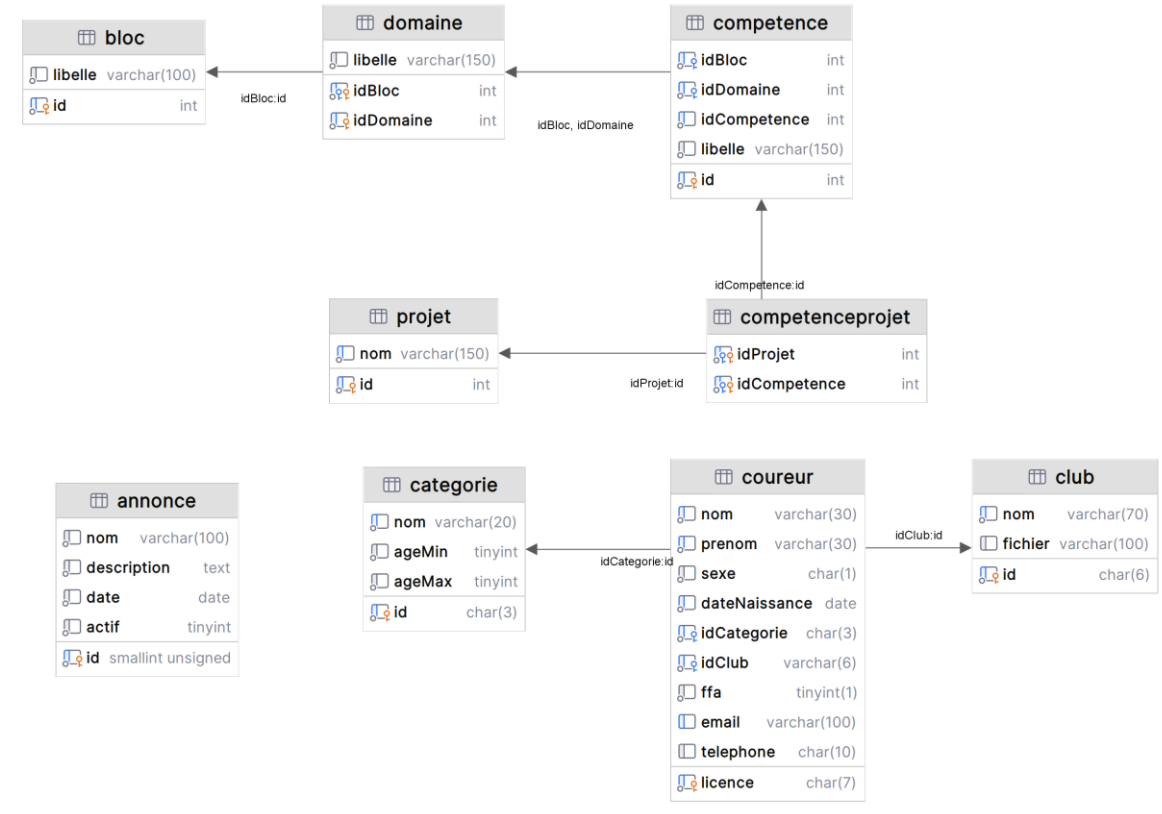
De comprendre comment on accède en consultation aux données d'une table D'écrire des méthodes dans une classe métier permettant de récupérer les données De récupérer et d'afficher les données côté client sous différentes mises en forme (tableau, carte)

1. Présentation de la structure de données

La structure de données doit être mise en place en exécutant les scripts SQL du répertoire '.sql' du projet

Exécuter les autres scripts SQL dans l'ordre suivant :
Coureur : create puis insert,
Projet : create puis insert,
Annonce : create puis insert

Schéma de la base de données gestion



2. La consultation des catégories

Répertoire : consultation/categorie
Objectif : afficher en générant une baliste <table> la liste des catégories

2.1. L'interface

Catégorie	Code	Âge entre	Né(e) entre
Minime	MI	14-15	2010-2011
Cadet	CA	16-17	2008-2009
Junior	JU	18-19	2006-2007
Espoir	ES	20-22	2003-2005
Senior	SE	23-34	1991-2002
Master 0	M0	35-39	1986-1990
Master 1	M1	40-44	1981-1985
Master 2	M2	45-49	1976-1980
Master 3	M3	50-54	1971-1975
Master 4	M4	55-59	1966-1970
Master 5	M5	60-64	1961-1965
Master 6	M6	65-69	1956-1960
Master 7	M7	70-74	1951-1955
Master 8	M8	75-79	1946-1950
Master 9	M9	80-84	1941-1945
Master 10	M10	85-99	1926-1940

2.2. La classe Categorie

Les classes métiers assurent le dialogue entre l'application et la base de données.
Elles contiennent des méthodes permettant d'interroger la base de données afin de récupérer les données qui alimenteront l'interface.

Chaque méthode doit :

- se connecter à la base,
- définir la requête SQL à envoyer au serveur MySQL (variable $sql),
- envoyer la requête SQL au serveur (différentes méthodes de la classe PDO sont possibles),
- retourner la réponse du serveur

Important : **Seules les méthodes des classes métier peuvent contenir des instructions SQL**

Pour réaliser cette interface, nous avons besoin de récupérer les données de la table categorie(id, nom, ageMin, ageMax)

Afin de faciliter l'affichage des données, la requête va retourner des données concaténées :

concat(ageMin, '-', ageMax) as age, concat($annee - ageMax, '-' ,$annee - ageMin) as annee

La requête doit aussi retourner le nombre de coureurs appartenant à chaque catégorie. Cette information servira plus tard à détecter si une catégorie peut être supprimée.
Les changements de catégories sont réalisés au premier septembre.
La variable $année doit contenir l'année en cours si l'on se situe avant le premier septembre
La variable $année doit contenir l'année prochaine si l'on se situe après le 31 août.

```
$annee = date('Y');
$mois = intval(Date('m'));
if ($mois >= 9) {
    $annee++;
}
```

La méthode getAll() doit permettre de récupérer dans un tableau numérique les enregistrements contenus dans la table 'categorie'.

Chaque enregistrement est stocké dans un tableau associatif (équivalent d'un dictionnaire) dont la clé correspond au nom de la colonne dans la table catégorie.

Afin de limiter les ligne de code à écrire, la classe technique Select offre trois méthodes pour lire les données

getRows() pour retourner un tableau indexé numériquement contenant des tableaux associatifs, un pour chaque ligne de résultat SQL.

getRow() pour retourner un seul enregistrement dans un tableau associatif

getValue() pour retourner

En PDO	Avec la classe Select
$sql = <<<SQL Select… where id = :id; SQL; $db = **Database::getInstance();** $cmd **= $db->prepare**($sql); $cmd->**bindValue**('id', $id); $cmd->**execute**(); $lesLignes = $cmd->fetchAll(PDO::FETCH_ASSOC); $cmd->closeCursor(); return $ligne;	$sql = <<<SQL Select … where id = :id; SQL; **$select = new Select();** **return $select->getRow($sql, ['id' => $id]);**

En PDO (requête sans paramètre)	Avec la classe Select
$sql = "Select…" $db = **Database::getInstance();** $cmd **= $db->query**($sql); $lesLignes = $cmd->fetchAll(PDO::FETCH_ASSOC); …	$sql = "Select…" **$select = new Select();** **return $select->getRows($sql);**

La méthode getAll prend alors la forme suivante :

```
public static function getAll(): array  {
        $annee = date('Y');
        $mois = intval(Date('m'));
        if ($mois >= 9) {
           $annee++;
        }
        $sql = <<<SQL
           select id, nom, ageMin, ageMax, concat(ageMin, '-', ageMax) as age,
                    concat($annee - ageMax, '-' ,$annee - ageMin) as annee,
                    (select count(*) from coureur where coureur.idCategorie = categorie.id) as nb
           from categorie
           order by ageMin;
SQL;
        $select = new Select();
        return $select->getRows($sql);
}
```

1. Compléter la méthode getAll().

2.3. Le script index.html

Il décrit la partie statique de la page, l'utilisation du composant html2pdf qui génère une version PDF de la page, nécessite d'encapsuler la partie de l'interface à générer au format PDF dans une balise

Il contient principalement la balise <table> avec la définition de l'entête du tableau et la balise tbody, identifiée par l'id 'lesLignes' afin de pouvoir la remplir dynamiquement.

```
<table style="margin: 0 auto;" >
<thead>
<tr>
   <th style="width: 200px; text-align: left">Catégorie</th>
   <th style="width: 100px; text-align: left" >Code</th>
   <th style="width: 150px; text-align: center">Âge entre</th>
   <th style="width: 150px; text-align: center">Né(e) entre</th>
</tr>
</thead>
<tbody id="lesLignes"></tbody>
</table>
```

Si on ne définit pas l'attribut style, les colonnes seront ajustées à leur contenu avec un cadrage à gauche par défaut.

2.4. Le script index.php

C'est le point d'entrée (Endpoint) de la fonctionnalité demandée.

Il doit :

- récupérer les données de la table 'categorie' en appelant la méthode Categorie::getAll(),
- transmettre ses données côté client.

Nous savons que pour transmettre des données entre le serveur Web (Apache avec PHP) et le client (navigateur) il faut utiliser le format JSON.

Par conséquent, il est nécessaire de convertir en JSON le résultat de la méthode PHP getAll() :

```
$lesCategories = json_encode(Categorie::getAll());
```

Les données sont ensuite transmises côté client en utilisant la variable $head qui contient une balise script permettant la déclaration de la constante JavaScript recevant le contenu de la variable PHP.

Ici nous utilisons le composant html2pdf pour obtenir la conversion de la page HTML en fichier PDF. Ce composant est pratique lorsque la page HTML tient sur une seule page PDF. Il faut donc ajouter le chargement de ce composant dans la variable $head

```
$head = <<<HTML
  <script src="/composant/html2pdf/html2pdf.bundle.min.js"></script>
  <script>
     const lesCategories = $lesCategories;
  </script>
HTML;
```

En PHP, <<< est utilisé pour délimiter une chaîne de caractères multilignes, dans laquelle les variables PHP sont interprétées automatiquement. Ce délimiteur doit être suivi d'un mot-clé arbitraire (par exemple EOD, HTML, SQL).
La chaîne se termine par ce même mot-clé, placé au début d'une nouvelle ligne sans indentation.
L'outil d'anlyse de code JsHint signale une erreur sur l'utilisation de const car il ne peut pas comprendre que $lesCategories contient une chaine JSON représentant un tableau [….]. Il faut désactiver ce signalement.

2. Compléter le script index.php

2.5. Le script index.js

Le script index.js doit générer dynamiquement le contenu de la balise <tbody id='lesLignes'>
Il s'agit de parcourir chaque ligne du tableau 'lesCategories' afin de générer une balise tr contenant 4 balises td.
Indication : la feuille de style style.css définit la mise en forme des balises table, tr et td.

```
for (const categorie of lesCategories) {
   const tr = lesLignes.insertRow();
   tr.style.verticalAlign = 'middle';
   tr.id = categorie.id;

   // Colonne : Catégorie
   tr.insertCell().innerText = categorie.nom;

   // Colonne : Code
   tr.insertCell().innerText =  categorie.id;

   // Colonne : Âge entre (centré)
   let td = tr.insertCell();
   td.innerText = categorie.age;
   td.style.textAlign = 'center';

   // Colonne : Né(e) entre (centré)
   td = tr.insertCell();
   td.innerText = categorie.annee;
   td.style.textAlign = 'center';
}
```

La génération de la version PDF se déclenche sur le clic du bouton btnGenerer :

```
btnPdf.addEventListener("click", () => {
   const element = document.getElementById("pdfContent");

   const opt = {
      margin:       0.5,
      filename:     'categories.pdf',
      image:        { type: 'jpeg', quality: 0.98 },
      html2canvas:  { scale: 2 },
      jsPDF:        { unit: 'in', format: 'a4', orientation: 'portrait' }
   };

   html2pdf().set(opt).from(element).save();
});
```

Remarque : JsHint n'est pas capable de connaitre les données qui ont été envoyées côté client depuis le script index.php. Pour éviter qu'il signale des erreurs sur ces données en les soulignant, il faut ajouter une ligne en commentaire : */*global lesCategories html2pdf */*

3. Compléter le script index.js.

3. Les coureurs

Répertoire : consultation/coureur

3.1. L'interface

Rechercher : licence, nom ou club...

Nombre de coureurs : **627**

Licence ▲	Nom prénom	Sexe	Né(e) le	Catégorie	Club
1291517	Abdesmed Mohamed	M	21/04/1961	M5	Amicale Du Val De Somme
2054055	Aboubi Maurad	M	19/12/1983	M1	Us Camon
1650412	Allonneau Jean Marie	M	09/02/1958	M6	Amicale Du Val De Somme
1217206	Andre Stephane	M	16/02/1979	M2	Amicale Du Val De Somme
2072538	Ansard Alexis	M	08/06/1995	SE	Amiens Uc

3.2. La classe Coureur

Pour réaliser cette interface, nous avons besoin de récupérer les données de la table 'coureur' : la licence, le nom, le prénom, le sexe, la date de naissance dans le format français (dateNaissanceFr) l'id de la catégorie, le nom du club

La méthode getAll() doit permettre de récupérer les enregistrements contenus dans la table 'coureur'.

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

4. Compléter la méthode getAll()

3.3. Le script index.php

Il charge les données à afficher pour les transmettre côté client.
L'interrogation de la base de données s'effectue toujours par l'intermédiaire d'une classe métier, ici la méthode getAll() de la classe Coureur

```
$lesCoureurs= json_encode(Coureur::getAll());
$head = <<<HTML
   <script>
      let lesCoureurs= $lesCoureurs;
   </script>
HTML;
```

5. Compléter le script index.php

3.4. Le script index.js

Le script index.js doit afficher les données en prenant éventuellement en prenant en compte la valeur recherchée (search) afin d'appliquer un filtrer
Le tableau sera responsif.
Les données doivent pouvoir être triées sur toutes les colonnes du tableau.

La fonction ucWord du composant /composant/fonction/format.js est utilisée afin d'afficher le nom et prénom avec uniquement la première lettre en majuscule (dans la table 'coureur', les noms et prénoms sont stockés en majuscule)

La fonction activerTri du composant /composant/fonction/tableau.js. va permettre de mettre en place les tris demandés.

La fonction afficher(lesCoureurs) va générer pour chaque ligne du tableau lesCoureurs une balise <tr> contenant des balises <td>

```
function afficher(lesCoureurs) {
   const valeur = search.value.toLowerCase();

   lesLignes.innerHTML = '';
   let count = 0;

   for (const coureur of lesCoureurs) {
      // Filtrage direct, sans créer un nouveau tableau
      if (
         valeur &&
         !coureur.licence.toLowerCase().includes(valeur) &&
         !coureur.nomPrenom.toLowerCase().includes(valeur) &&
         !coureur.nomClub.toLowerCase().includes(valeur)
      ) {
         continue; // on passe au suivant
      }
      count++;
      const tr = lesLignes.insertRow();
      tr.style.verticalAlign = 'middle';
      tr.id = coureur.id;

      let td;

      td = tr.insertCell();
      td.innerText = coureur.licence;
      td.style.textAlign = 'center';

      td = tr.insertCell();
      td.innerText = ucWord(coureur.nomPrenom);
      td.style.textAlign = 'left';

      td = tr.insertCell();
      td.innerText = coureur.sexe;
      td.style.textAlign = 'center';
```

```
        td = tr.insertCell();
        td.innerText = coureur.dateNaissanceFr;
        td.style.textAlign = 'center';

        td = tr.insertCell();
        td.innerText = coureur.idCategorie;
        td.style.textAlign = 'center';

        td = tr.insertCell();
        td.innerText = ucWord(coureur.nomClub);
        td.style.textAlign = 'left';
    }
    nb.innerText = count;
}
```

Pour mettre en place les tri il suffit d'appeler la fonction activerTri en lui passant dans un objet, l'ensemble des paramètres attendus :

```
activerTri({
    idTable: "leTableau",
    getData: () => lesCoureurs,
    afficher: afficher,
    triInitial: {
        colonne: 'licence',
        ordre: "asc"
    }
});
```

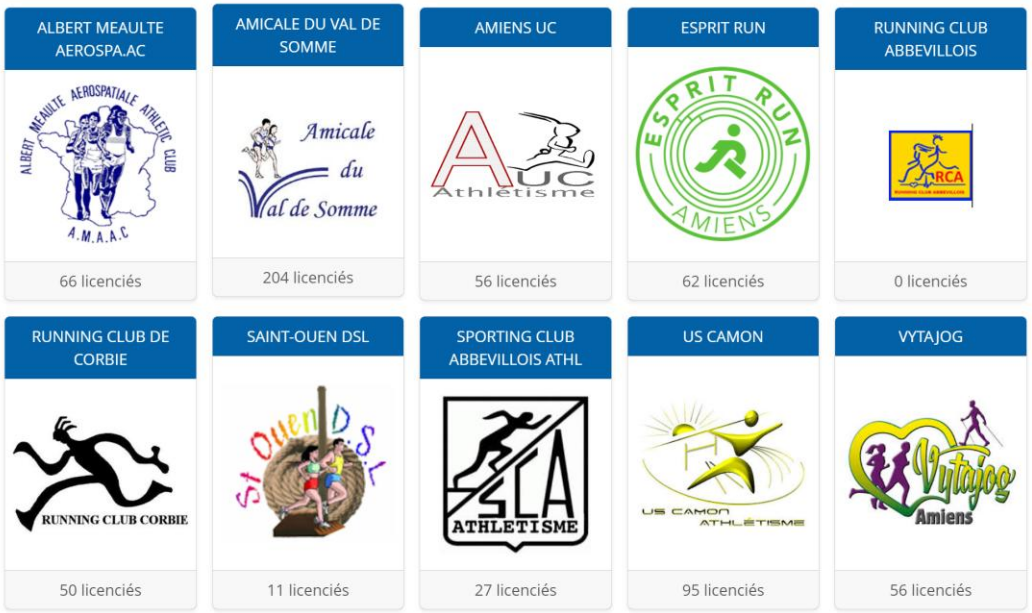
6. Compléter le script index.js

4. Les clubs

Répertoire : consultation/club

Il est parfois préférable d'afficher les données sous forme de carte à l'aide des classes Bootstrap 'card'. C'est le cas lorsque les données sont peu nombreuses ou comportent des images. Nous allons afficher les clubs de cette façon.

4.1. L'interface



Le champ fichier de la table club contient le nom du fichier image associé au club (son logo).
Les logos sont stockés dans le répertoire /data/club

4.2. La classe Club

Nous avons besoin de récupérer dans la base de données les colonnes id, nom, fichier et le nombre de licenciés de chaque club. Tous les clubs doivent apparaitre.

Pour obtenir le nombre de licenciés de chaque club, nous pouvons utiliser une sous requête dans la clause 'select'

```
select id, nom, fichier, (select count(*) from coureur where coureur.idClub = club.id) as nb
from club
order by nom;
```

Il est souhaitable de réaliser un contrôle supplémentaire permettant de s'assurer que la colonne fichier contient une URI (Uniform Resource Identifier ou Adresse d'une ressource physique) valide. En d'autres termes, s'assurer que le logo est bien présent dans le répertoire data/club afin d'éviter des erreurs lors de l'affichage côté client.

Pour cela il faut parcourir le résultat de la requête et lui ajouter une colonne 'present' de type booléen qui indiquera si la ressource existe bien.

La colonne fichier de la table club accepte la valeur null (tous les clubs n'ont pas forcément de logo)

```
public static function getAll(): array   {
        $sql = <<<SQL
           select id, nom, fichier, (select count(*) from coureur where coureur.idClub = club.id) as nb
           from club
           order by nom;
SQL;
        $select = new Select();
        $lesLignes = $select->getRows($sql);

        // ajout d'une colonne permettant de vérifier l'existence du logo
        foreach ($lesLignes as &$ligne) {
            $ligne['present'] = isset($ligne['fichier']) && is_file(self::DIR . $ligne['fichier']);
        }
        return $lesLignes;
}
```

Self::DIR permet d'obtenir le chemin complet côté serveur pour atteindre le fichier image.

7. Compléter la méthode getAll().

4.3. Le script index.html

L'interface statique se limite ici à une simple div identifiée 'lesCartes' sur laquelle les cartes générées dynamiquement viendront s'insérer.

```
<div id='lesCartes'></div>
```

4.4. Le script index.php

Point d'entrée de la fonctionnalité, il doit récupérer les clubs pour les transmettre côté client dans une variable JavaScript.

```
// récupération des clubs
$lesClubs = json_encode(Club::getAll());

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

8. Compléter le script index.php

4.5. Le script index.js

Il affiche les données sous la forme de carte. Une carte est un ensemble de balise div avec des classes Bootstrap : card, card-header, card-body et card_footer.

Documentation : **Bootstrap 5 Notions de base.pdf**

Le modèle est le suivant :

```
<div class="card carte-club shadow-sm" style="flex: 0 1 300px; max-width: 100%;">
    <div class="card-header text-center" style="height: 80px; background-color: rgb(7, 144, 228);
color: white;">
      ALBERT MEAULTE AEROSPA.AC
    </div>
    <div class="card-body" style="height: 250px;">
        <img src="/data/club/amaac.png" alt="ALBERT MEAULTE AEROSPA.AC logo"
            style="max-height: 100%;
                max-width: 100%;
                object-fit: contain;
                display: block;
                margin: 0px auto;">
   </div>
    <div class="card-footer text-muted text-center">
        64 licenciés
    </div>
</div>
```

flex: 0 1 300px : La carte ne grandit pas au-delà de sa base, Elle peut se rétrécir si l'espace est limité
object-fit: contain : Assure que l'image garde ses proportions, même si elle est plus petite, sans être coupée

La fonction creerCarte(element) assure cette génération dynamique en JavaScript.

```
function creerCarte(element) {
    // Création de la carte principale
    const carte = document.createElement('div');
    carte.classList.add('card', 'carte-club', 'shadow-sm');

    // Entête de la carte
    const entete = document.createElement('div');
    entete.classList.add('card-header', 'text-center');
    entete.style.height = '80px';

    // Couleurs dynamiques depuis les variables CSS
    entete.style.backgroundColor =
getComputedStyle(document.documentElement).getPropertyValue('--background-color-
header').trim();
    entete.style.color = getComputedStyle(document.documentElement).getPropertyValue('--text-
color-header').trim();
    entete.innerText = element.nom;
```

```
    // Corps
    const corps = document.createElement('div');
    corps.classList.add('card-body');
    corps.style.height = '250px';

    if (element.present) {
        const img = document.createElement('img');
        img.src = '/data/club/' + element.fichier;
        img.alt = `${element.nom} logo`;
        img.style.maxHeight = '100%';
        img.style.maxWidth = '100%';
        img.style.objectFit = 'contain';
        img.style.display = 'block';
        img.style.margin = '0 auto'; // centrer horizontalement
        corps.appendChild(img);
    }

    // Pied de la carte
    const pied = document.createElement('div');
    pied.classList.add('card-footer', 'text-muted', 'text-center');
    pied.innerText = `${element.nb} licenciés`;

    carte.appendChild(entete);
    carte.appendChild(corps);
    carte.appendChild(pied);

    carte.style.flexGrow = '0';          // La carte ne grandit pas au-delà de sa base
    carte.style.flexShrink = '1';        // Elle peut se rétrécir si l'espace est limité
    carte.style.flexBasis = '300px';     // Largeur de base (taille cible)
    carte.style.maxWidth = '100%';       // Elle ne dépasse jamais la largeur de son conteneur

    return carte;
}
```

Le programme principal va parcourir le tableau lesClubs pour demander la génération des cartes/
Pour obtenir un résultat responsif , le style du conteneur est mis en place.

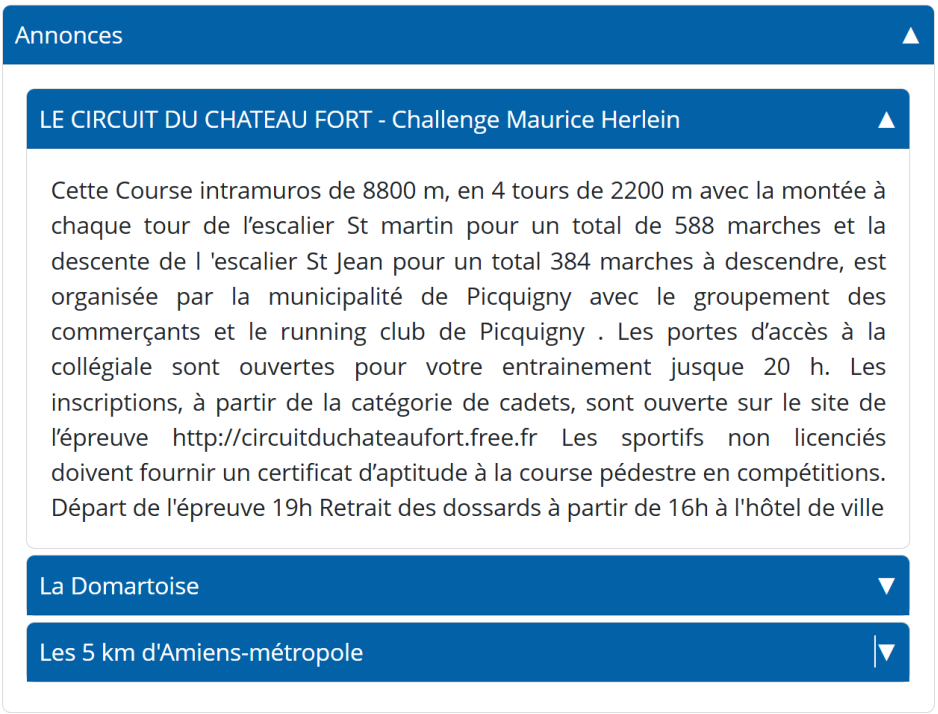
```
//  Mise en forme du conteneur flex
lesCartes.style.display = 'flex';              // Flexbox activé
lesCartes.style.flexWrap = 'wrap';             // Retour à la ligne si besoin
lesCartes.style.gap = '1rem';                  // Espacement entre les cartes
lesCartes.style.justifyContent = 'flex-start'; // Alignement des cartes à gauche

// Génération des cartes à partir des données
for (const element of lesClubs) {
    const carte = creerCarte(element);
    lesCartes.appendChild(carte);
}
```

9. Compléter le script index.js.

5. Les annonces

Répertoire : consultation/annonce

5.1. L'interface



La particularité réside ici dans le fait de n'afficher que les annonces 'actives' dont la date n'est pas dépassée.
L'affichage s'effectue dans des 'cartes' avec la mise en place d'un système d'ouverture/fermeture à l'aide du composant fonction/openclose.js

5.2. La classe Annonce

Nous avons besoin de récupérer les annonces 'actives' dont la date n'est pas dépassée.

```
public static function getLesAnnoncesActives() {
        $sql = <<<SQL
          Select id, nom, description, date, actif, date_format(date, '%d/%m/%Y') as dateFr
          from annonce
          where actif = 1
          and date >= curdate()
          order by date;
SQL;
        $select = new Select();
        return $select->getRows($sql);
}
```

10. Compléter la méthode getLesAnnoncesActives().

5.3. Le script index.html

Il comporte uniquement le conteneur des annonces qui est lui-même une 'carte' sur laquelle s'applique aussi le système d'ouverture/fermeture

```
<div class="d-flex mt-1" style="max-width: 600px">
   <div class="card w-100">
      <div class="card-header entete">Annonces</div>
      <div id="contenuCadreAnnonce" class="m-1" style="display: none;"></div>
   </div>
</div>
```

w-100 permet de prendre toute la largeur disponible dans le conteneur parent sinon la largeur serait celle nécessaire pour afficher le mot 'Annonces' seul élément visible au départ.

5.4. Le script index.js

Il doit parcourir le tableau 'lesAnnonces' et afficher chaque annonce sous forme de carte.
Il faut décomposer le problème en créant une fonction qui génère une carte et un programme principal qui parcourt le tableau et appelle cette fonction.

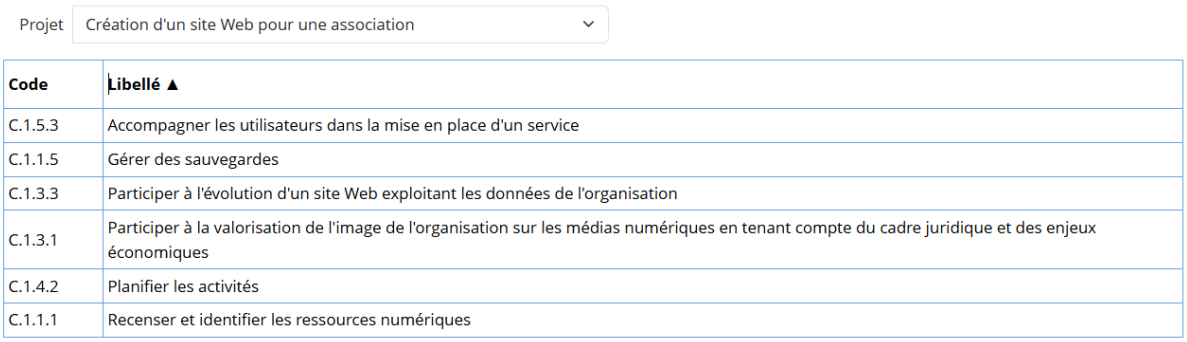
```
function creerCarte(element) {
   const carte = document.createElement('div');
   carte.classList.add("card", "mb-1"); // mb-1 pour espace vertical entre cartes
   // Entête
   const entete = document.createElement('div');
   entete.classList.add("card-header", "entete");
   entete.appendChild(document.createTextNode(element.nom));
   carte.appendChild(entete);
   // Corps
   const corps = document.createElement('div');
   corps.classList.add("card-body", "text-break");
   corps.style.textAlign = "justify";
   corps.innerHTML = element.description;
   carte.appendChild(corps);
   return carte;
}

// Programme principal
let nbInformation = lesAnnonces.length;
if (nbInformation > 0) {
   for (const element of lesAnnonces) {
      const carte = creerCarte(element);
      contenuCadreAnnonce.appendChild(carte);
   }
} else {
   contenuCadreAnnonce.innerHTML = genererMessage("Aucune annonce pour le moment.",
"orange");
}
```

11. Compléter le script index.js

6. Les compétences associées à un projet

Répertoire : consultation/projet

6.1. L'interface



Code	Libellé ▲
C.1.5.3	Accompagner les utilisateurs dans la mise en place d'un service
C.1.1.5	Gérer des sauvegardes
C.1.3.3	Participer à l'évolution d'un site Web exploitant les données de l'organisation
C.1.3.1	Participer à la valorisation de l'image de l'organisation sur les médias numériques en tenant compte du cadre juridique et des enjeux économiques
C.1.4.2	Planifier les activités
C.1.1.1	Recenser et identifier les ressources numériques

6.2. La classe Projet

La classe Projet regroupent les méthodes permettant de gérer à la fois la table 'projet' et la table 'competenceprojet'. Elle utilise aussi la table 'competence' afin d'obtenir le libellé des compétences.

Au niveau de la consultation, nous avons besoin de 3 méthodes pour alimenter l'interface :

getAll() pour récupérer l'ensemble des projets : select id, nom from projet order by nom;

getLesCompetences($idProjet) pour retourner les compétences associées au projet dont l'id est passé en paramètre (utilisée dans le script ajax/getlescompetences.php)

```
public static function getLesCompetences(int $idProjet): array  {
   $sql = <<<SQL
    select competence.id, concat('C.', idBloc, '.', idDomaine, '.', competence.idCompetence) as code,
           libelle
    from competenceprojet
         join competence on competence.id = competenceprojet.idCompetence
    where idProjet = :idProjet
    order by libelle;
SQL;
    $select = new Select();
    return $select->getRows($sql, ['idProjet' => $idProjet]);
 }
```

getById($id) pour vérifier l'existence du projet (utilisée dans le script ajax/getlescompetences.php)

```
public static function getById(int $id): mixed  {
       $sql = "select id, nom from projet where id = :id;";
       $select = new Select();
       return $select->getRow($sql, ['id' => $id]);
}
```

12. Compléter les 3 méthodes de la classe Projet.

6.3. Le script index.html

Il se compose d'une zone de liste qui sera remplie dynamiquement et de l'entête du tableau qui affichera dynamiquement les compétences du projet sélectionné.

```
<div class="d-flex align-items-center gap-2" style="max-width: 600px; width: 100%;">
   <label for="idProjet" class="form-label m-0" style="white-space: nowrap;">Projet</label>
   <select class="form-select flex-grow-1" id="idProjet" aria-label="Projet"></select>
</div>
<div id="msg"></div>
<table id='leTableau' style="margin: auto;" class="table table-sm table-bordered">
   <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>
```

Les attributs "data" **data-champ** et **data_type** vont permettre d'utiliser la fonction activerTri() du composant /composant/fonction.tableau.js pour trier le tableau en cliquant sur l'entête de la colonne.
data-champ permet d'indiquer la propriété des éléments du tableau Javascript sur lequel le tri sera effectué.
data-type permet de définir le type des données à trier (text par défaut, date ou number)

6.4. Le script ajax/getlescompetences.php

Lorsque l'utilisateur sélectionne un autre projet, il faut de nouveau interroger le serveur pour récupérer les compétences associées au projet sélectionné.

Le script ajax/getlescompetences.php permet de répondre à cette demande. La valeur du l'identifiant du projet (idProjet) lui est transmise par la méthode POST.

Nous commençons toujours par vérifier que le paramètre attendu a bien été transmis :

```
if (!isset($_POST['idProjet'])) {
   Erreur::envoyerReponse("Paramètre manquant.", 'global');
}
```

La méthode **envoyerReponse** arrête le script et renvoie le message d'erreur dans une reponse au format JSON qui sera traitée automatiquement par la fonction appelAjax() appelée dans la fonction chargerLesCompetencesDuProjet(idProjet) du script index.js.

Le second paramètre ('global') permet d'indiquer le type d'erreur qui sera pris en compte pour choisir le mode d'affichage de l'erreur. Pour une erreur de type global, l'affichage se fera par défaut dans la balise si elle existe sinon dans une fenêtre modale.

Si l'id du projet est bien transmis, nous stockons sa valeur dans une variable de même nom

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

Il faut ensuite vérifier son format avant d'éviter toute tentative d'injection. Plusieurs solutions s'offrent à nous : l'utilisation de fonction PHP comme filter_var() ou filter_input() sont possible mais elles ne s'adaptent pas toujours au format attendu.

Ex : $idProjet = filter_var($_POST['idProjet'], FILTER_VALIDATE_INT);
Ex : $idProjet = filter_input(INPUT_POST, 'idProjet', FILTER_VALIDATE_INT);
Filtre possible : FLOAT EMAIL URL IP REGEXP

La solution qui s'adapte le mieux reste l'utilisation d'une expression régulière.

Documentation : les expressions régulières.pdf et PHP Les expressions régulières.pdf

Nous utilisons l'expression régulière suivante ^\d+$

^ : Indique le début de la chaîne.
\d: Correspond à n'importe quel chiffre de 0 à 9 (equivalent à [0-9]).
+ : Spécifie qu'il doit y avoir au moins 1 chiffre.
$: Indique la fin de la chaîne.

La fonction **preg_match** en PHP permet de rechercher une correspondance entre une expression régulière et une chaîne de caractères. Elle renvoie un résultat indiquant si l'expression régulière correspond à une partie ou à l'intégralité de la chaîne.

Il est nécessaire de mettre les délimiteurs (souvent des barres obliques /) autour de l'expression régulière

```
if (!preg_match('/^\d+$/', $idProjet)) {
    Erreur::envoyerReponse("L'identifiant du projet n'est pas valide.", 'global');
}
```

Vérifions maintenant que le projet existe en appelant la méthode getById de la classe Projet

```
if (!Projet::getById($idProjet)) {
    Erreur::envoyerReponse("Ce projet n'existe pas.", 'global');
}
```

Nous pouvons maintenant récupérer les compétences du projet en appelant la méthode Projet::getLesCompetences($idProjet) et les envoyer dans le format JSON/

```
echo json_encode(Projet::getLesCompetences($idProjet));
```

13. Compléter le script ajax/getlescompetence.php

6.5. Le script index.php

Il est chargé de récupérer les données de la table projet qui serviront à alimenter la zone de liste pour les envoyer côté client.

Il faut ici envisager le cas où la table projet est vide.
Toutes les erreurs détectées dans un projet peuvent être traitées à l'aide de la classe Erreur
La méthode afficherReponse(message, type) redirige le visiteur vers la page d'erreur charger d'afficher l'erreur.

Le message qui sera afficher dépend du type de l'erreur

'global' : le message d'erreur sera affiché sur la page dédiée aux erreurs (erreur/index.php)

'system' : L'erreur est une erreur inattendue interceptée par un try catch. Le message sera enregistré dans le fichier /backoffice/.log/erreur.log et l'interface affichera le message standard "Une erreur inattendue est survenue"

Documentation : La gestion des erreurs.pdf

```
// récupération de tous les projets
$lesProjets = Projet::getAll();

// Si aucun projet n'est enregistré dans la base de données, on envoie une erreur
if (count($lesProjets) === 0) {
  Erreur::afficherReponse("Aucun projet n'est enregistré dans la base de données.", 'global');
}

$lesProjets = json_encode($lesProjets);
$head = <<<HTML
   <script>
      let lesProjets = $lesProjets;
   </script>
HTML;
```

14. Compléter le script index.php

6.6. Le script index.js

Il doit générer les balises <option> de la balise <select> en parcourant les lignes du tableau lesProjets

```
for (const projet of lesProjets) {
   idProjet.add(new Option(projet.nom, projet.id));
}
```

Lorsque l'utilisateur sélectionne un autre projet, il faut de nouveau interroger le serveur pour récupérer les compétences associées au projet sélectionné.

Comment déclencher le traitement ? En définissant un gestionnaire d'événement attaché à la zone de liste qui traite l'événement change.

Documentation : **JavaScript Les événements.pdf – JavaScript Les modules.pdf**

```
idProjet.onchange = () => {
      chargerLesCompetencesDuProjet(idProjet.value);
};
```

La fonction chargerLesCompetencesDuProjet(idProjet) doit interroger le serveur tout en restant sur la page. Pour cela nous utilisons la technologie Ajax qui permet côté client (donc en JavaScript) d'interroger le serveur sans avoir besoin de quitter la page.

Nous utiliserons ici la fonction appelAjax du composant fonction/ajax.js pour réaliser l'ensemble des appel Ajax.

```
import {appelAjax} from "/composant/fonction/ajax.js";
```

Cette fonction utilise l'API fetch du langage JavaScript pour réaliser cet appel. Elle offre l'avantage de gérer les erreurs internes et de les afficher sur l'interface en ayant recours aux fonctions d'affichage du composant fonction/afficher.js

```
appelAjax({url,  data = null,  method = 'POST',  success = null,  error = null,  dataType = 'json' })
```

url : script PHP à interroger (toujours placé dans le sous répertoire ajax du module)
data : objet Json ou objet formData contenant les données passées en paramètre
method : POST ou GET
success : fonction de rappel qui sera automatiquement exécuté en cas de réussite
error : fonction de rappel qui sera automatiquement exécuté en cas d'erreur (en plus de l'erreur déjà affichée par la fonction elle-même)
dataType : le type de réponse attendu : JSON, TEXT ou blob

L'appel Ajax s'effectue dans la fonction chargerLesCompetencesDuProjet

```
function chargerLesCompetencesDuProjet(idProjet) {
   msg.innerHTML = "";
   appelAjax({
      url: 'ajax/getlescompetences.php',
      data: { idProjet: idProjet },
      success: data => {
           lesCompetences = data;
            afficherLesCompetences(lesCompetences);
         }
   });
}
```

Les propiétés 'method' et 'dataType' n'étant pas précisées, vont prendre leur valeur par défaut ('POST' et 'JSON')

Pour permettre d'activer le tri sur les colonnes affichées, il reste à appeler la fonction activerTri du composant fonction.tri.js

```
activerTri({
   idTable: "leTableau",
   getData: () => lesCompetences,
   afficher: afficherLesCompetences,
   triInitial: { colonne: 'libelle',  ordre: "asc" }
});
```

idTable : valeur de l'attribut id de la balise <table>
getData : fonction qui retourne la source de données (permet ainsi à la source d'évoluer au cours du traitement)
afficher : fonction qui sera automatiquement appelée après le tri de la source. Il s'agit toujours de la fonction qui permet d'afficher les données sur l'interface. Le paramètre data est implicitement transmis.
triInitial : objet JSON indiquant l'ordre initial des données du tableau, afin d'afficher l'icône représentant le tri initial dans la bonne colonne.

15. Compléter le script index.js.