SCI6306 Bases de données documentaires (Automne 2023)

Christine Dufour, EBSI, UdeM

A2023 17 novembre 2023

Cours 10 : Formulaire pour saisie (2 de 2)

SCI6306

Christine Dufour, EBSI, UdeM

Table des matières

Formulaires Web pour saisir des données dans une BD relationnelle avec PHP (partie 2 de 2)

- + Au programme aujourd'hui
- Bases de données sur le Web : Saisie de données (partie 2 de 2)
    - B.A.-BA de la chasse aux erreurs en PHP
    - Préparation du code PHP
        - Structure générique pour enregistrement
        - Bloc 1 : Connexion avec la base de données
        - Bloc 2 : Requête SQL générique pour enregistrer de nouvelles données
        - Bloc 3 : Exécution de la requête et définition des variables
        - Blocs 4 et 5 : fermeture de la requête et fermeture de la base de données
    - Où placer le code pour l'enregistrement ?
        - Scénario 1 : Aller vers une autre page
            - Scénario 2 : Rester sur le formulaire
    - Exemple d'interface complexe
- Projet de session, volet C
- Ressources en lien avec le cours

1. + Au programme aujourd'hui

- **B.A.-BA de la chasse aux erreurs en PHP**
- **Intégration Web d'une base de données MySQL pour l'enregistrement de données**
    - Partie 2 : Préparation du code PHP pour enregistrer les données
- **Présentation du volet C**
- **Travail en laboratoire : Volet C (Partie A)**

2. Bases de données sur le Web : Saisie de données (partie 2 de 2)

2.1. B.A.-BA de la chasse aux erreurs en PHP

Certains **réflexes** sont utiles à développer lorsque vient le temps de **trouver des erreurs** dans du code PHP. Le système ne retourne en effet pas toujours des messages d'erreur très clairs, voire même peut ne rien retourner! Cela peut entre autres dépendre de la configuration du serveur. Pour des raisons de sécurité, il arrive que les messages d'erreur détaillés soient désactivés, comme c'est le cas sur *gin-ebsi*. C'est ce qui explique l'injection du fichier *erreurs.php* dans les fichiers du volet C du projet de session. On y retrouve des instructions permettant d'activer ponctuellement les messages d'erreur plus explicites. Mais même avec des messages plus explicites, des stratégies complémentaires sont utiles pour déboguer de manière plus efficace le code.

À moins de travailler dans Bloc-Notes, l'environnement utilisé pour développer des pages comportant du PHP effectue habituellement de la **coloration syntaxique**. Cette coloration permet de distinguer, par exemple, les balises HTML du contenu de ces dernières, les inclusions de PHP dans une page HTML, les commandes PHP des variables ou des chaînes de caractères fixes. Dans l'extrait ci-dessous, c'est la coloration syntaxique qui nous montre assez rapidement que certains caractères n'ont pas été compris comme il faut. La chaîne *les chiens* devrait être en gris pâle comme la chaîne *les chats*.

```
<p>Permettez-moi de vous présenter <strong><?php echo $_POST['prenom']; ?></strong> : <?php echo date('Y-m-d
H:i:s')-$_POST['naissance']; ?> ans, aimant <?php echo ($_POST['animal']=='chat'?'les chats':les chiens); ?>,
sa maxime préférée est <cite><?php echo $_POST['maxime']; ?></cite>!</p>
```

Exploitation de la coloration syntaxique pour détecter des erreurs

Un autre truc fort utile à exploiter dans Visual Studio Code est sa **mise en exergue des caractères d'encadrement** (parenthèses, balises, etc.) lorsque l'on clique sur le caractère d'ouverture ou celui de fermeture. Cela permet en effet de rapidement vérifier si tout est bien "balancé". Dans l'exemple ci-dessous, on voit très bien la parenthèse ouvrante et celle fermante associée.

```
<p>Permettez-moi de vous présenter <strong><?php echo $_POST['prenom']; ?></strong> : <?php echo date('Y-m-d
H:i:s')-$_POST['naissance']; ?> ans, aimant <?php echo ($_POST['animal']=='chat'?'les chats':'les chiens'); ?>,
sa maxime préférée est <cite><?php echo $_POST['maxime']; ?></cite>!</p>
```

Mise en exergue des caractères d'encadrement associés

Finalement, la possibilité de mettre des éléments en **commentaire** aide à cibler les morceaux de code problématiques. On peut ainsi systématiquement valider ce qui fonctionne ou non. Comme illustré ci-dessous, dans PHP, la syntaxe pour indiquer des commentaires consiste en deux barres obliques consécutives.

```
<p>Permettez-moi de vous présenter <strong><?php echo $_POST['prenom']; ?></strong> : <?php echo date('Y-m-d
H:i:s')-$_POST['naissance']; ?> ans, aimant <?php //echo ($_POST['animal']=='chat'?'les chats':'les chiens'); ?
>, sa maxime préférée est <cite><?php echo $_POST['maxime']; ?></cite>!</p>
```

Mise en commentaires de code PHP pour isoler un problème

2.2. Préparation du code PHP

L'approche utilisée dans le cadre du cours pour la programmation en PHP est une approche de **PHP pour non programmeur**. Il ne sera en effet pas nécessaire de comprendre tout le détail de la syntaxe. L'important sera de comprendre les différents "*blocs de construction*" présentés pour être en mesure d'en adapter le code. Dans les notes de cours, les différents blocs de code seront présentés avec l'identification des éléments à modifier **en gras**.

On retrouvera *5 blocs de base* pour **enregistrer des données** dans la table *X* de la base de données *Y* :

1. **Connexion** avec la base de données *Y*
2. **Préparation de la requête SQL générique** pour l'ajout des données dans la table *X*
3. **Exécution de la requête SQL** en précisant les **valeurs** à saisir
4. **Fermeture de la requête**
5. **Fermeture de la base de données**

Le code présenté dans les exemples par la suite provient de l'exemple accessible à l'URL https://cours.ebsi.umontreal.ca/sci6306/demo_c10/formulaire.php (les fichiers sont accessibles à l'URL https://cours.ebsi.umontreal.ca/sci6306/demo_c10/sci6306_demo_c10.zip). Il s'agit d'une légère modification de l'exemple du cours précédent, la différence étant que les données saisies dans le formulaire ne sont pas affichées dans une page Web, mais qu'elles sont **enregistrées** dans une base de données. Le code PHP pour sauvegarder les données se trouve au tout début du fichier où se trouve le formulaire (`formulaire.php`).

a) Structure générique pour enregistrement

Afin de bien **gérer les exceptions** (c'est à dire les cas où le code provoquerait des erreurs), les 5 blocs permettant d'enregistrer des données se retrouveront imbriqués dans la structure suivante :

```
try
{
[Blocs 1 à 5]
}
// Gestion des erreurs
catch(Exception $e)
{
exit('Erreur : '.$e->getMessage());
}
```

b) Bloc 1 : Connexion avec la base de données

Le code suivant permet d'ouvrir une connexion avec la base de données où l'on désire enregistrer des données. **Trois éléments** sont à personnaliser, mis en gras, soit (1) le **nom de la base de données** MySQL, (2) le **nom d'usager** sur le serveur MySQL et (3) le **mot de passe** sur le serveur MySQL :

```
$pdo_options[PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;
$bdd = new PDO('mysql:host=localhost;dbname=nom_de_la_BD', 'nom_d'usager_MySQL', 'mot_de_passe_MySQL', $pdo_options);
$bdd->exec("SET CHARACTER SET utf8");
```

Comme les informations d'identification sur le serveur sont passées en clair dans le code (notamment le mot de passe!), **ce bloc n'est habituellement pas inséré directement dans le code PHP des pages**, pour des raisons de sécurité. Cette information est placée dans un **fichier séparé** (nommé `ouverture_bd.php` dans l'exemple utilisé), fichier que l'on dépose dans un dossier `App_Data`, dossier qui fait l'objet d'une protection particulière sur le serveur.

Dans le fichier PHP qui sert à l'enregistrement des données, on **insère ce fichier** en l'appelant avec une commande **INCLUDE** de cette manière : `include 'App_Data/ouverture_bd.php';` (on y indique le **chemin relatif** pour se rendre jusqu'au fichier contenant le bloc d'instruction).

La commande INCLUDE peut aussi être utilisée pour insérer dans une page des blocs de commande qui sont communs à plusieurs pages (par exemple, une entête de page ou un pied de page). Cela évite d'avoir à les réécrire plusieurs fois et permet de plus efficacement les mettre à jour. Par exemple, dans le volet C du projet de session, c'est ainsi que le pied de page est inséré dans toutes les pages.

c) Bloc 2 : Requête SQL générique pour enregistrer de nouvelles données

Le deuxième bloc consiste à **déclarer** la **requête SQL qui servira à enregistrer les données**. À ce moment-ci, elle n'est pas exécutée, mais uniquement préparée (déclarée). Ceci s'explique du fait que l'on n'y mettra pas les valeurs à enregistrer comme telles, mais plutôt des *variables* que nous compléterons dans le troisième bloc. Dans notre exemple, voici en quoi consiste le deuxième bloc :

```
$requete_ajout = $bdd->prepare('INSERT INTO donnees (q1, q2_1, q2_2, q2_3, q3, q4, q5, q6, q6_autre, q7) VALUES (:q1, :q2_1, :q2_2, :q2_3, :q3, :q4, :q5, :q6, :q6_autre, :q7);');
```

Le premier élément (*$requete_ajout*) est le **nom de la variable** dans laquelle on met en mémoire la **requête** préparée. En **PHP**, un nom de **variable** est préfixé par un signe de dollar (**$**). La chaîne qui suit - `requete_ajout` - est un nom que vous choisissez et qui doit être représentatif du contenu de la variable.

Dans ce qui suit, ce qu'il faut **modifier** est la **requête SQL** comme telle pour **la faire correspondre à la structure de votre base de données** et **aux champs que vous voulez y ajouter**. Ici, nous retrouvons une requête **INSERT INTO** comme nous voulons ajouter un **nouvel enregistrement** dans la table. Dans certains contextes, un formulaire pourrait permettre de mettre à jour des données existantes; en ce cas, il s'agira plutôt d'une requête UPDATE.

Il faut donc **adapter la requête INSERT INTO** pour préciser, (1) le **nom de la table**, (2) les **champs que l'on veut enregistrer**, et (3) les **valeurs à enregistrer dans les champs** (voir au besoin les notes de cours sur les *requêtes Ajout (INSERT INTO)*). Les **valeurs à verser** dans la table (clause *VALUES*) **ne sont pas définies ici spécifiquement**. La requête étant générique, on les représente plutôt par des **variables**; la syntaxe d'une **variable** en **SQL** est de **préfixer le nom donné à la variable** (par exemple `q1`) d'un **deux points** (`:q1`). Comme on peut le remarquer ici, les noms des variables correspondent en tout point aux noms des champs où l'on enregistre les données; c'est en effet plus simple de s'y retrouver ainsi!

L'utilisation de variables permet de rendre la requête SQL plus **lisible**.

d) Bloc 3 : Exécution de la requête et définition des variables

Le troisième bloc permet d'**exécuter la requête SQL** préparée précédemment en y **injectant, dans les variables, les données présentes dans le formulaire**. Dans l'exemple, on retrouve cette ligne pour le bloc trois :

```
$requete_ajout->execute(array('q1' => $_POST['q1'], 'q2_1' => $_POST['q2_1'], 'q2_2' => $_POST['q2_2'], 'q2_3' => $_POST['q2_3'], 'q3' => $_POST['q3'], 'q4' => $_POST['q4'], 'q5' => $_POST['q5'], 'q6' => $_POST['q6'], 'q6_autre' => (@$_POST['q6_autre']?$_POST['q6_autre']:NULL), 'q7' => $_POST['q7']));
```

La commande débute en rappelant le nom donné à la variable *$requete_ajout* contenant la requête (bloc 2).

L'exécution se fait par la suite avec la commande `execute`. Dans le cas d'une requête SQL comportant des variables, comme c'est le cas ici, la commande `execute` aura comme **argument/paramètre** la précision des **valeurs** pour chacune des **variables**. Cela se fera par le biais d'un **vecteur** (`array`), un vecteur étant une liste d'éléments, ici les variables de la requête SQL avec leurs valeurs. La **syntaxe générique d'un vecteur** est la suivante : `array('nom_var1'=>valeur1, 'nom_var2'=>valeur2`. On y retrouve précisée chacune des variables avec leurs valeurs, ces binômes `nom=>valeur` étant séparés par des virgules. On devra ainsi y retrouver **autant de binômes qu'il y a de variables** dans la requête SQL.

Comme mentionné au cours 9, la syntaxe suivante permet de **récupérer les valeurs saisies** dans des **contrôles** d'un formulaire soumis avec la méthode **POST** : `$_POST['nom_controle']`. C'est de cette manière que les différentes variables de la requête SQL sont associées aux contrôles correspondants, à une exception près. On remarque en effet un traitement différent pour le contrôle de formulaire `q6_autre`.

On verse en effet dans cette variable, non pas directement la valeur du contrôle correspondant (`$_POST['q6_autre']`), mais plutôt ceci : `(@$_POST['q6_autre']? $_POST['q6_autre']:NULL)`. Nous avons déjà croisé cette syntaxe, qui est celle pour une **condition**. Le premier morceau correspond au critère, le deuxième à ce qu'il faut faire si c'est vrai et le troisième à ce qu'il faut faire si c'est faux. Ainsi, en français, cela revient à vérifier si on retrouve quelque chose dans ce contrôle. Si c'est le cas, on placera dans la variable la valeur saisie. Si ce n'est pas le cas, on y placera plutôt la valeur *NULL*. Ce traitement est **nécessaire** pour les **boîtes de saisie textuelles pour les champs facultatifs**. Si on ne le fait pas, ce sera non pas une valeur *NULL* qui sera saisie, mais plutôt une **chaîne vide**.

e) Blocs 4 et 5 : fermeture de la requête et fermeture de la base de données

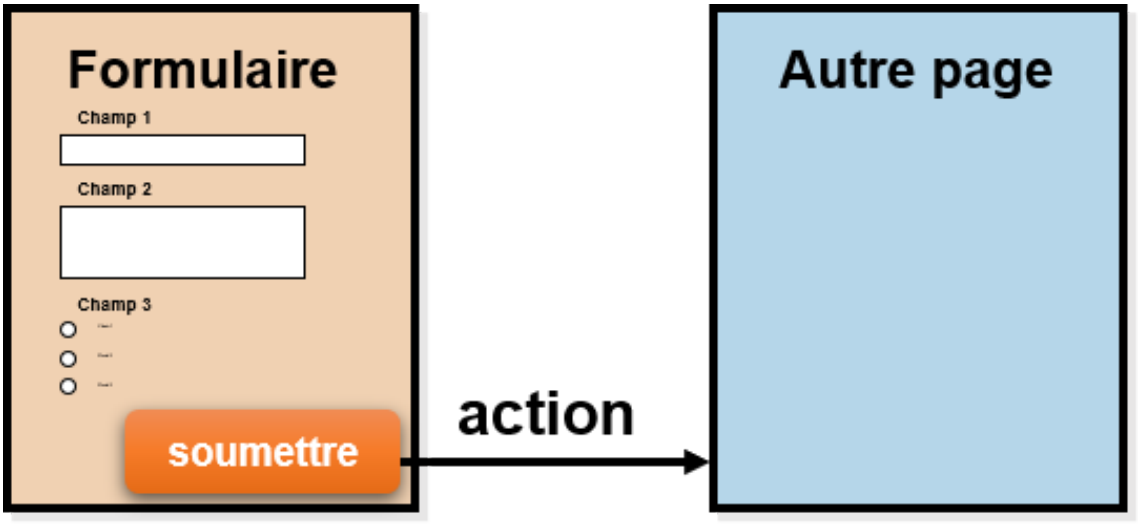
Il est bien important de s'assurer de **fermer la requête** ainsi que la **base de données** une fois l'enregistrement terminé. Sans ça, vous laissez plein de **portes ouvertes** qui pourraient, à la longue, créer une certaine **instabilité** sur le serveur. Ici il n'y a rien à modifier (à moins que vous ayez nommé différemment la variable où vous avez saisi la requête SQL).

```
// Bloc 4 : fermeture de la requête
$requete_ajout->closeCursor();
```

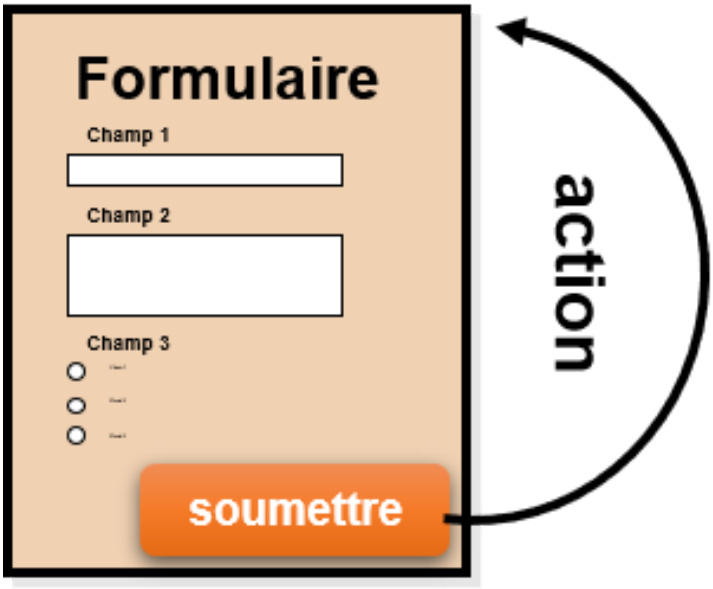
```
// Bloc 5 : fermeture de la base de données
$bdd=null;
```

2.3. Où placer le code pour l'enregistrement ?

Dans l'exemple précédent, le code PHP pour enregistrer les données se trouvait à même la page du formulaire. En fait, il y a deux scénarios possibles (décrits dans les sections suivantes) en fonction de ce que l'on souhaite comme comportement après la soumission des données saisies :



Scénario 1 : On veut, après avoir soumis le formulaire, aller à une autre page



Scénario 2 : On veut permettre la saisie en continu de plusieurs réponses au formulaire (donc on veut rester sur le formulaire)

a) Scénario 1 : Aller vers une autre page

Lorsque l'on veut, après avoir soumis un formulaire, **aller à une autre page**, le code PHP pour enregistrer les données **se retrouvera sur cette autre page, au tout début** de cette dernière comme illustré ci-dessous :

```php
<?php

// Enregistrement des données

try
{ ···
}

// Gestion des erreurs
catch(Exception $e)
{
    exit('Erreur : '.$e->getMessage());
}

?>
<!DOCTYPE html>
<html>

<html lang="fr-ca">
  <head>
    <meta charset="utf-8" />
```

Code pour enregistrer les données au début de la page de destination (scénario 1)

i) Scénario 2 : Rester sur le formulaire

Si, une fois que vous avez soumis le formulaire, **vous désirez demeurer sur la page du formulaire** pour continuer la saisie, le code pour procéder à l'enregistrement **se retrouvera à même la page du formulaire, généralement au tout début**. Il faut par contre se demander ce qui arrive lorsque l'on visite la page pour la toute première fois... Comme il n'y aura pas encore eu de saisie, on ne voudra pas que ces instructions pour l'enregistrement soient exécutées! En fait, on veut qu'elles **s'exécutent uniquement après avoir cliqué sur le bouton "Enregistrer"**. Vous êtes ici dans une logique d'une condition, la condition étant "si j'ai cliqué sur Enregistrer" avec comme action si c'est vrai d'exécuter les instructions et, si c'est faux, de ne pas les exécuter. La syntaxe pour y arriver est la suivante :

Fichier cours.php

Particularité **: Le code pour l'enregistrement est inséré dans une condition IF pour qu'il ne s'exécute que si la page a été rechargée après avoir cliqué sur le bouton de soumission.**

Ici, le formulaire avait pour action de recharger la page.

Le bouton avait pour nom « action » et sa valeur, lorsque cliqué, est « Enregistrer ».

```
if (@$_POST['action']=="Enregistrer")
{


    try
        {...
        }

        // Gestion des erreurs
        catch(Exception $e)
        {
        die('Erreur : '.$e->getMessage());
        }


<form class="form-horizontal" id="formulaire_cours" name="formulaire_cours" method="post" action="cours.php">


<input class="btn btn-success navbar-btn" name="action" id="Enregistrer" type="submit"
value="Enregistrer" />
```

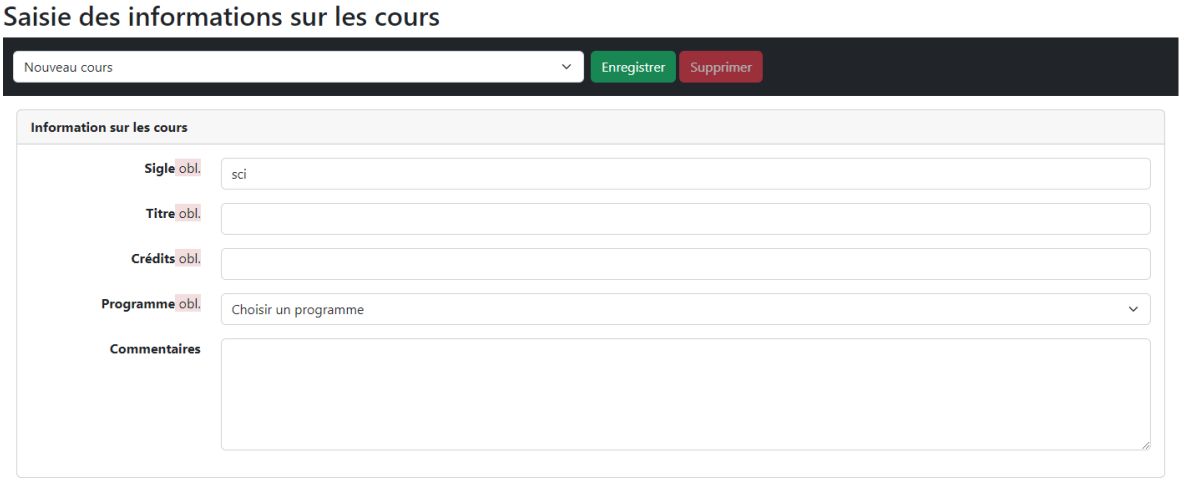
Exemple du code pour enregistrer les données tout en demeurant sur le formulaire (scénario 2)

Dans cet exemple, la condition "si j'ai cliqué sur Enregistrer" correspond à la première ligne : `if ($_POST['action']=="Enregistrer")`. Dans cette ligne, `$_POST['action']` correspond à la valeur du contrôle nommé `action` dans le formulaire, qui est ici le bouton de soumission. Lorsque l'on clique sur un bouton de soumission, on y met ce qui se trouve dans son attribut `value`, soit ici `Enregistrer`. La condition consiste donc à vérifier si on retrouve cette valeur dans ce contrôle, ce qui ne sera le cas qu'après l'avoir cliqué !

Si cette condition est vraie, ce qui suit et qui se retrouve entre accolades - soit les blocs pour enregistrer les données - sera exécuté. Si elle est fausse, comme il n'y a pas de clause `else` qui suit, rien n'est exécuté.

2.4. Exemple d'interface complexe

Dans un système plus complexe, on retrouvera plusieurs contrôles pour la saisie ainsi que plusieurs boutons permettant différentes actions. C'est le cas, par exemple, de l'extrait de l'interface du système utilisé pour le volet C du travail pour la saisie des informations sur les cours ci-dessous :



Interface pour la saisie des informations sur les cohortes

On retrouve dans la barre de navigation du formulaire un **menu déroulant** généré automatiquement à partir de la base de données et présentant la **liste des cours**. Lorsque l'on choisit un des cours, cela permet d'insérer dans les différents champs du formulaire les données présentent dans la base de données pour ce dernier. Il est alors possible de les modifier.

On retrouve aussi un **bouton Enregistrer**. Il permet d'exécuter une requête UPDATE pour mettre les données à jour.

Finalement, on retrouve un deuxième **bouton**, **Supprimer**, qui permet d'effacer dans la base de données les informations sur le cours choisi en exécutant une requête DELETE. Ce bouton est activé uniquement lorsque l'on a choisi un cours.

Une manière d'inclure plusieurs boutons dans un formulaire est d'insérer les lignes de commande qui leur sont associées dans une condition afin de ne les exécuter que lorsque le bouton a été cliqué. Par exemple, dans l'exemple ci-dessus, les instructions pour l'enregistrement sont à l'intérieur d'une condition `if (@$_POST['action']=='Enregistrer')`. Les instructions pour la suppression sont à l'intérieur d'une condition `if (@$_POST['action']=='Supprimer')`.

3. Projet de session, volet C

L'objectif du dernier volet du projet de session est de **développer des pages Web** pour faire de la **saisie** de données ainsi que de la **visualisation** de données.

Vous travaillerez à partir d'un **système Web existant** (disponible sur StudiUM) qu'il vous faudra **compléter** pour répondre aux besoins présentés dans le protocole. Le **visuel est prédéfini** à partir de la bibliothèque CSS **Bootstrap 5**.

Le **livrable** est l'**ensemble des fichiers constituant le système Web développé**. Il n'y a pas de journal de bord en parallèle à préparer pour justifier les choix; au besoin, des justifications pourront être faites à même le code HTML ou PHP en utilisant leurs balises respectives pour inclure des **commentaires**.

Pour ce travail, vous exploiterez le **compte d'équipe sur MySQL** ainsi que l'**espace d'équipe sur GIN-EBSI**.

✎ *Remarque* : Bibliothèque CSS Bootstrap 5

Bootstrap est une **bibliothèque CSS libre** qui offre un **visuel ergonomique et adaptatif** aux différents environnements de consultation (mobile, etc.) :

« Powerful, extensible, and feature-packed frontend toolkit. Build and customize with Sass, utilize prebuilt grid system and components, and bring projects to life with powerful JavaScript plugins. » (https://getbootstrap.com/[1])

Des **exemples documentés** des principaux styles utiles pour ce travail sont disponibles à l'URL https://cours.ebsi.umontreal.ca/sci6306/exemples_bootstrap.htm. Pour information, voici d'autres sources complémentaires qui pourraient se révéler utiles :

- https://www.w3schools.com/bootstrap5/index.php
- https://openclassrooms.com/fr/courses/7542506-creez-des-sites-web-responsives-avec-bootstrap-5
- https://getbootstrap.com/docs/5.2/getting-started/introduction/

4. Ressources en lien avec le cours

Matériel de cours

- *Notes de cours [cf. sci6306_cours10_notes.pdf]*

[1] http://getbootstrap.com/