

Gestion des données

L'ajout

Table des matières

1. Méthode de développement	1
2. Présentation de la structure de données.....	2
3. Présentation de la structure de l'application	4
Le répertoire .config contient le fichier de configuration du menu horizontale	4
4. Présentation de l'ajout.....	5
4.1. La classe Table et les Classes Input.....	5
4.2. La script ajax/ajouter.php	5
5. Ajout d'une catégorie	7
5.1. Le rôle du déclencheur avantAjoutCategorie	7
5.2. La classe métier	8
5.3. Vérification des contrôles à l'aide de Postman	10
5.4. L'interface : /administration/categorie/ajout/index.html	11
5.5. Le script /administration/categorie/ajout/index.php	11
5.6. Le script /administration/categorie/ajout/index.js	12
6. Ajout d'un coureur	14
6.1. Le déclencheur associé.....	14
6.2. La classe métier Coureur.....	14
6.3. Vérification des contrôles à l'aide de Postman	15
6.4. L'interface : /administration/licencie/ajout/index.html	16
6.5. Le script /administration/licencie/ajout/index.php.....	17
6.6. Le script /administration/licencie/ajout/index.js	18
7. Ajout d'une annonce	20
7.1. La classe métier	20
7.2. L'interface : /administration/annonce/ajout/index.html	20
7.3. Le script /administration/annonce/ajout/index.php	21
7.4. Le script /administration/annonce/ajout/index.js	21
8. Ajout d'un projet et de ses compétences	23
8.1. La classe métier Projet (classemetier/projet.php)	24
8.2. Les déclencheurs	25
8.3. L'interface.....	26
8.4. Le script ajax/enregistrer.php	27
8.5. Vérification des contrôles à l'aide de Postman	29
8.6. Le script /administration/projet/index.php.....	29
8.7. Le script /administration/projet/index.js.....	29

Gestion des données : L'ajout

Objectif de l'activité professionnelle

A la fin de cette activité, vous serez capable :

- D'utiliser les classes techniques 'Input' et 'Table' pour réaliser l'ajout de données dans une table
- De mettre en place une classe métier dérivée de la classe Table permettant d'assurer toutes les opérations sur une table
- De mettre en place une interface graphique permettant la saisie et le contrôle des données côté client
- De mettre en place l'ensemble des contrôles côté serveur
- De prendre en compte les champs facultatifs lors d'une opération d'ajout
- De mettre en place une transaction pour réaliser en une seule opération des ajouts dans plusieurs tables

Indications

Toutes les données saisies doivent forcément être contrôlées avant d'être stockées dans la base de données.

Dans une application accessible au public, les contrôles doivent obligatoirement s'effectuer côté serveur et éventuellement coté client pour éviter des allers-retours inutiles vers le serveur.

Dans une application back office (accès contrôlé) il n'est pas conseillé de réduire les contrôles côté serveur sous prétexte d'être passé par son interface client pour la réalisation des opérations.

Un pirate peut toujours avoir usurpé l'identité d'un administrateur.

Les contrôles peuvent être mis en place à trois niveaux :

Au niveau de la base de données sous la forme de contraintes de clé primaire, de clé étrangère, d'index unique, de contrainte check, de valeur not null et de déclencheur. C'est le verrou le plus sûr car le plus proche des données.

Au niveau de l'application côté serveur (code PHP)

Au niveau du client (code Javascript)

1. Méthode de développement

Nous allons utiliser la méthode de développement "Security-First Development" (Développement orienté sécurité) ou "Security by Design" (Sécurité par conception) qui consiste à intégrer immédiatement les tests de sécurité dès le début du processus de développement plutôt que de la considérer comme une tâche distincte ou un ajout ultérieur.

En adoptant une approche "Security-First", les développeurs intègrent des pratiques de sécurité tout au long du cycle de vie du développement logiciel. Cela inclut la validation des données d'entrée, la protection contre les attaques par injection, la gestion des sessions de manière sécurisée, la protection contre les attaques CSRF (Cross-Site Request Forgery), la sécurisation des communications, etc.

L'objectif est de minimiser les vulnérabilités dès le départ et de réduire les risques liés à la sécurité. Cela peut également inclure l'utilisation d'outils automatisés de test de sécurité, la réalisation d'audits de sécurité réguliers et la formation des développeurs sur les meilleures pratiques de sécurité.

Gestion des données : L'ajout

2. Présentation de la structure de données

La structure de données reste la même que pour l'activité 'consultation'

Cependant des déclencheurs sont ajoutés afin de réaliser les contrôles au niveau de la base de données

Il faut donc commencer par exécuter les deux scripts 'declencheur ajout.sql' contenus dans le répertoire '.sql' du projet.

Ces déclencheurs vont appliquer une mise en forme, vérifier le format des données, l'unicité et la cohérence. Par exemple :

```
create trigger avantAjoutCategorie before insert on categorie
  for each row
begin
  -- Mise en forme et vérification sur le code
  set new.id = ucase(new.id);

  if new.id not regexp '^[A-Za-z][A-Za-z0-9]{1,2}$' then
    signal sqlstate '45000' set message_text = '~Le code ne respecte pas le format attendu.';
  end if;

  if exists(select 1 from categorie where id = new.id) then
    signal sqlstate '45000' set message_text = '~Ce code est déjà attribué à une autre catégorie';
  end if;

  ....
  -- Vérification de l'intervalle des âges
  if new.ageMin not regexp '^([1-9] | [1-9][0-9])$' then
    signal sqlstate '45000' set message_text = '~Le format de l'âge minimale est invalide.';
  end if;

  if new.ageMax not regexp '^([1-9] | [1-9][0-9])$' then
    signal sqlstate '45000' set message_text = '~Le format de l'âge maximale est invalide.';
  end if;

  if new.ageMin > new.ageMax then
    signal sqlstate '45000' set message_text = '~L'intervalle des âges est invalide';
  end if;

  -- chevauchement de la tranche d'âge avec une catégorie existante
  -- pas de chevauchement si : new.ageMax < ageMin ou new.ageMin > ageMax
  -- donc chevauchement si : new.ageMax >= ageMin et new.ageMin <= ageMax

  if exists(select 1 from categorie where new.ageMax >= ageMin and new.ageMin <= ageMax)
  then
    signal sqlstate '45000' set message_text =
      '~L'intervalle de cette catégorie est en conflit avec une autre catégorie';
  end if;
end;
```

Avantages d'un déclencheur

- Centralisation des règles de validité dans la base elle-même, quelle que soit l'application (site web, back-office, phpMyAdmin...)
- Messages fonctionnels personnalisés, retournés directement par le déclencheur
- Garantie d'un respect systématique des règles, indépendamment de l'origine de la requête SQL

Inconvénients d'un déclencheur

- Opacité du code métier : les règles sont implémentées "dans la base", donc moins visibles pour les développeurs. Cela peut rendre le débogage plus complexe.
- En cas de changement de règle métier, il faut modifier le code SQL du déclencheur, ce qui implique une bonne maîtrise de SQL procédural.
- Les performances peuvent être dégradées si le déclencheur effectue des requêtes coûteuses sur des tables volumineuses (ex. : détection de chevauchements dans une grande table de catégories).

L'utilisation d'un déclencheur permet donc de renforcer la robustesse fonctionnelle de la base en y inscrivant des règles métier avancées.

Il permet également de déléguer à la base la responsabilité de vérifier la cohérence des données tout en fournissant à l'utilisateur des messages clairs et pertinents.

Cependant, elle implique une certaine discipline de documentation et une vigilance accrue pour les développeurs qui interagissent avec la base.

Gestion des données : L'ajout

3. Présentation de la structure de l'application

Afin de donner une structure standard et professionnelle, l'application regroupe toutes les opérations de gestion : ajout, modification et suppression au sein du **répertoire administration**

L'interface se compose d'un menu vertical unique permettant de sélectionner la table à gérer et d'un menu horizontal propre à chaque table qui propose les liens vers opérations de gestion sur cette table.

◀

ListeAjoutMise à jour

Catégorie

Tableau des catégories pour la saison en cours

Licencié

Annonce

Projet

Catégorie	Code	Âge entre	Né(e) entre
Benjamin	BE	11-13	2012-2014
Minimes	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

Les menus sont mis en place à l'aide des composants `client menuhorizontal` et `menuvertical` du répertoire `/composant`.

Le fichier de configuration du menu vertical est stocké dans le répertoire `/.config`

Les fichiers de configuration des menus horizontaux sont stockés dans le répertoire `.config` de chaque module

- administration
 - annonce
- categorie
 - .config
 - ajout
 - `<>` index.html
 - `JS` index.js
 - `php` index.php
 - liste
 - `<>` index.html
 - `JS` index.js
 - `php` index.php
 - maj
 - `<>` index.html
 - `JS` index.js
 - `php` index.php
- licencie
- projet

Le répertoire administration comporte un sous répertoire pour chaque table MySQL gérée

Dans chaque sous répertoire nous trouvons les répertoires suivants :

ajout : pour réaliser l'ajout

liste : pour visualiser les enregistrements et proposer éventuellement la suppression d'un enregistrement et ou la modification de la valeur d'une colonne de la table/

maj : pour les opérations de modification et de suppression sur un enregistrement.

Le répertoire `.config` contient le fichier de configuration du menu horizontale

Rappel : Cette activité ne concerne que le traitement de l'ajout pour les 4 tables.

4. Présentation de l'ajout

L'ajout peut être réalisé de manière standardisée en définissant pour chaque table de la base de données une classe dérivée de la classe Table.

4.1. La classe Table et les Classes Input

La classe Table encapsule l'ensemble des opérations nécessaires à la validation, à la mise en forme et à l'insertion de données dans une table disposant d'une clé primaire simple (souvent nommée id, mais pas nécessairement), indépendamment du nombre de champs que contient la table.

Les colonnes "non calculées" de la table sont déclarées dans la classe dérivée, sous la forme d'objets issus d'une classe Input, ou plus précisément d'une classe fille de Input choisie en fonction du type de donnée attendu : InputEntier, InputTexte, InputDate, InputEmail, InputURL, InputListe, etc.

Ces classes permettent d'intégrer :

- des contrôles de validation : pattern, min, max, minLength, maxLength, etc. ;
- des opérations de filtrage ou de transformation : conversion en majuscules, suppression des accents, normalisation des espaces, etc.

4.2. La script ajax/ajouter.php

Pour toutes les tables disposant d'une clé primaire non composée, l'ajout d'un enregistrement s'effectue via l'appel au script standardisé '/ajax/ajouter.php'

Ce script attend :

- Le nom de la table (i.e. le nom de la classe correspondante) ;
- Un objet { nomColonne : valeur, ... } contenant les valeurs des champs à insérer.

Il n'est pas nécessaire de transmettre tous les champs de la table. Les champs facultatifs peuvent être omis sans générer d'erreur.

```
// Contrôle sur le nom de la classe
if (isset($_POST['table']) || ! class_exists($_POST['table'])) {
    Erreur::envoyerReponse("La table n'est pas transmise ou n'existe pas.", 'global');
}
// récupération des données
$table = $_POST['table'];
// création d'une instantiation dynamique de classe
$table = new $table();
// Ajout dans la table en vérifiant que tous les champs sont corrects
if (!$table->donneesTransmises()) {
    $reponse = ['error' => $table->getLesErreurs()];
} elseif (!$table->checkAll()) {
    $reponse = ['error' => $table->getLesErreurs()];
} else {
    $table->insert();
}
```

Gestion des données : L'ajout

```
if ($table->getLastInsertId()) {  
    $reponse = ['success' => $table->getLastInsertId()];  
} else {  
    $reponse = ['success' => 'Opération réalisée avec succès'];  
}  
}  
header('Content-Type: application/json; charset=utf-8');  
echo json_encode($reponse, JSON_UNESCAPED_UNICODE);
```

Le script `ajax/ajouter.php` réalise l'ensemble des contrôles sur les données transmises et lance la demande d'ajout d'un enregistrement dans la table `coureur`.

- La méthode **`donneesTransmise()`** vérifie la transmission des données.
- La méthode **`checkAll()`** vérifie la conformité des données et prend en charge leur mise en forme.
- La méthode **`insert()`** exécute l'instruction `INSERT` dans la base et intercepte les erreurs MySQL éventuelles (ex. doublon, violation de contrainte).

Si une ou plusieurs erreurs sont détectées, le script retourne une réponse JSON de la forme suivante :

```
{ "error": { "nom": "Ce nom est déjà utilisé.", ... } }
```

Les clés de l'objet retourné permettent d'identifier l'origine de l'erreur :

- `nom` d'un champ → erreur spécifique au champ
- `"global"` → erreur de portée générale (ex. table inexistante)
- `"system"` → erreur technique inattendue

L'utilisation de la classe `Table` permet de centraliser l'ensemble des contrôles métiers, tout en automatisant la vérification et l'insertion des données. Cela limite considérablement le besoin d'écrire du code de validation dans chaque script applicatif.

Cette approche ne peut pas être utilisée dans tous les cas. En particulier :

- Si la table possède une clé primaire composée (composite), la classe `Table` ne convient pas dans sa forme actuelle.
- Si l'ajout doit être encapsulé dans une transaction complexe impliquant plusieurs tables ou étapes critiques (ex. cohérence entre plusieurs insertions).

Nous allons étudier ici différents exemples d'ajout.

Gestion des données : L'ajout

5. Ajout d'une catégorie

5.1. Le rôle du déclencheur avantAjoutCategorie

Lors de la création des tables, plusieurs contraintes d'intégrité sont définies :

Contrainte	Signification
Primary key	Garantit que chaque valeur dans cette colonne (ou combinaison de colonnes) est unique et non nulle .
Foreign key	Garantit que les valeurs insérées existent dans la table référencée (intégrité référentielle).
Unique	Garantit que les valeurs dans cette colonne sont uniques (mais peuvent être nulles, sauf si interdit).
Not Null	Empêche la colonne de contenir des valeurs nulles .
Check	Impose une condition logique à respecter sur les valeurs d'une colonne.

Si l'un de ces contraintes n'est pas vérifiée, alors le serveur MySQL génère une **erreur système** qui peut être interceptée et traitée via une instruction try catch. Toutefois le message d'erreur retourné est souvent technique et peu compréhensible pour un utilisateur final. Par exemple :

```
[23000][1062] Duplicata du champ 'CA' pour la clef 'categorie.PRIMARY'
[23000][1062] Duplicata du champ 'Cadet' pour la clef 'categorie.nom'
[23000][1452] Cannot add or update a child row: a foreign key constraint fails (`gestion`.`coureur`,
CONSTRAINT `coureur_ibfk_2` FOREIGN KEY (`idClub`) REFERENCES `club` (`id`))
```

L'affichage brut de ces messages dans l'interface utilisateur est à proscrire, pour deux raisons principales :

- Ces messages sont trop techniques, parfois même cryptiques pour un utilisateur non initié ;
- Ils peuvent divulguer des informations sensibles sur la structure de la base ou sur son contenu, ce qui pourrait aider un attaquant à affiner une attaque (ex. injection SQL).

Une première solution serait d'afficher un message neutre du type : "Une erreur est survenue. Veuillez réessayer."

Cependant, ce type de message générique ne permet ni à l'utilisateur de comprendre son erreur, ni au développeur de diagnostiquer la cause réelle du problème.

Il est donc préférable de retourner des messages fonctionnels et contextualisés, comme par exemple : "Ce code est déjà attribué à une autre catégorie."

Le déclencheur va prendre en compte les différents contrôles afin de retourner un message compréhensible par l'utilisateur

Ici comme nous l'avons vu dans la présentation de la structure de données, le déclencheur 'avantAjoutCategorie' permet de vérifier l'unicité au niveau du code et de nom, le respect des formats attendus et surtout l'absence de chevauchement d'âges entre deux catégories.

Rappel : La méthode insert de la classe Table, appelée depuis le script '/ajax/ajouter.php' récupère l'éventuel message d'erreur retourné par l'instruction signal en utilisant un bloc try ... catch et renvoie ce message en utilisant la méthode Erreur::traiterReponse(). Cette dernière va détecter qu'il s'agit d'un message provenant d'un déclencheur grâce au caractère ~ placé au début de chaque message.

5.2. La classe métier

Afin de pouvoir utiliser le script /ajax/ajouter.php pour ajouter une nouvelle catégorie, il faut dériver la classe Catégorie de la classe technique Table et lui ajouter un constructeur qui hérite du constructeur de la classe Table et qui définit les colonnes de la table dont les valeurs seront saisies par l'utilisateur.

Chaque colonne est représentée par un objet d'une classe dérivée de la classe Input : InputTest, InputInt, InputDate, InputMail, inputUrl etc.

Chaque objet 'Input' encapsule la valeur de la colonne et l'ensemble des règles de validité que doit respecter cette valeur.

Chaque règle est exprimée par une propriété. Les principales propriétés sont :

- Require : La valeur est obligatoire
- MinLength : La valeur doit avoir un nombre minimum de caractères
- MaxLength : La valeur doit avoir un nombre maximum de caractères
- Pattern : la valeur doit respecter l'expression régulière

Concernant la table catégorie(id, nom, ageMin, ageMax) nous utiliserons les objets suivants

Colonne	Objet	Propriété	Valeur
id	InputText	Require MinLength MaxLength Pattern	true 2 3 ^[A-Za-z][A-Za-z0-9]{1,2}\$
nom	InputText	Require MinLength MaxLength Pattern	true 5 20 ^[A-Za-z][A-Za-z]*(?: \d{1,2})?\$
ageMin	InputInt	Require Min Max	true 4 99
ageMax	InputInt	Require Min Max	true 4 99

Le (?: dans une expression régulière est appelé un groupe non capturant

Gestion des données : L'ajout

La traduction dans la classe `Categorie` est la suivante :

```
class Categorie extends Table {
    public function __construct() {
        parent::__construct('categorie');

        // définition de la clé primaire si ce n'est pas un auto-increment
        // colonne id
        $input = new InputText();
        $input->Require = true;
        // des lettres suivi éventuellement d'un espace obligatoirement suivi d'un ou 2 chiffres
        $input->Pattern = "[A-Za-z][A-Za-z0-9]{1,2}";
        $input->MinLength = 2;
        $input->MaxLength = 3;
        $this->columns['id'] = $input;

        // colonne nom
        $input = new InputText();
        $input->Require = true;
        // des lettres suivi éventuellement d'un espace obligatoirement suivi d'un ou 2 chiffres
        $input->Pattern = "[A-Za-z][A-Za-z]*(?: \\d{1,2})?";
        $input->MinLength = 5;
        $input->MaxLength = 20;
        $this->columns['nom'] = $input;

        // colonne ageMin
        $input = new InputInt();
        $input->Require = true;
        $input->Min = 4;
        $input->Max = 99;
        $this->columns['ageMin'] = $input;

        // colonne ageMax
        $input = new InputInt();
        $input->Require = true;
        $input->Min = 4;
        $input->Max = 99;
        $this->columns['ageMax'] = $input;
    }
}
```

L'attribut `columns` est un tableau associatif contenant l'ensemble des objets représentant chaque colonne de la table.

La clé de ce tableau correspond au nom de la colonne.

Gestion des données : L'ajout

5.3. Vérification des contrôles à l'aide de Postman

Le script ajouter.php permet de rajouter un enregistrement dans une table. Il faut pour cela lui passer en paramètre le nom de la table et les noms et valeur des champs renseignés.

Dans le cadre des opérations de mise à jour, il est souhaitable de vérifier le bon fonctionnement des contrôles mis en place dans la description de la classe `Categorie` et dans le déclencheur `avantAjoutCategorie` avant même de développer la partie cliente.

Cette vérification s'effectue à partir du client http Postman. Nous allons donc nous en servir ici simplement pour vérifier le bon fonctionnement du script `/ajax/ajouter.php` dans le cadre de l'ajout d'une catégorie.

Ouvrir l'application du bureau de Postman et se connecter
Créer dans son espace de travail, la **collection Gestion**
Dans cette collection créer le **dossier Ajout**
Dans ce dossier créer la **requête categorie**

url : `http://gestion/ajax/ajouter.php`
body : form-data
key : table (categorie), id, nom, ageMin et ageMax

Cette url sera appelée par un appel Ajax, il faut ajouter dans l'entête de la requête la clé `X-Requested-With` avec la valeur `XMLHttpRequest` pour simuler cet appel afin d'obtenir l'éventuel message d'erreur dans le bon format.

Rappel : La réponse en cas d'erreur est retournée dans le format json lors d'un appel Ajax.
Dans un appel direct, l'utilisateur est redirigé vers la page d'erreur.

Réaliser les contrôles suivants

Contrôle n° 1 : des valeurs ne sont pas transmises

Exemple de réponse attendue : `{"error":{"ageMax":"Veuillez renseigner ce champ."}}`

Contrôle n° 2 : des valeurs ne respectent pas le format attendu

Id : 99, nom : Benjamin, ageMin : a; ageMax : b

Remarque : MySQL réalise des conversions automatiques, par exemple `a11 = 11`

Contrôle n° 3 : L'intervalle des âges n'est pas valide

ageMin = 13 et ageMax = 11

Contrôle n° 4 : L'intervalle des âges chevauche une autre catégorie

ageMin = 11 et ageMax = 14

Contrôle n° 4 : Ajout validé

Réponse attendue : `{"success":"Opération réalisée avec succès"}`

Contrôle n° 5 : Détection d'un doublon au niveau de l'id

Réponse attendue : `{"error":{"global":"Ce code est déjà attribué à une autre catégorie"}}`

Contrôle n° 6 : Détection d'un doublon au niveau du nom

Id : Po, nom : benjamin, ageMin : 9 , ageMax : 10

Réponse attendue : `{"error":{"global":"Ce nom est déjà utilisé par une autre catégorie"}}`

Gestion des données : L'ajout

On peut voir que la réponse est au format JSON avec une clé 'success' ou une clé 'error'. La clé 'success' contient un message de réussite.

La clé 'error' comporte un ensemble d'élément de type {'clé' : 'valeur'}.

Les valeurs possibles pour cette clé sont :

- **'global'** : Lorsque le message ne s'adresse pas à un champ particulier.
- **'system'** : Lorsque le message d'erreur est lié à une erreur technique non prévue
- **le nom d'un champ** dont la valeur est à l'origine de l'erreur

5.4. L'interface : /administration/categorie/ajout/index.html

Code (2 ou 3 caractères alphanumériques, en commençant par une lettre) *

Nom (5 à 20 caractères, lettres non accentuées avec 1 ou 2 chiffres à la fin) *

Âge minimum *

Âge maximum *

Pour chaque balise Input, nous retrouvons, exprimée dans ses attributs, les règles de validité qui seront vérifiées côté client cette fois-ci.

Champ	Code	Nom	ageMin - ageMax
Type	Text	text	number
required	true	true	true
pattern	^[A-Za-z][A-Za-z0-9]{1,2}\$	^[A-Za-z][A-Za-z]*(?:\d{1,2})?\$	1 (min) 99 (max)
maxlength	3	30	
minlength	2	3	

L'ensemble des champs sont encapsulés dans une balise <div class='formulaire'>

La classe assure par défaut le centrage du formulaire de saisie si ce dernier a défini une largeur

```
<div class="formulaire" style='width:600px;'>
```

5.5. Le script /administration/categorie/ajout/index.php

Il n'a aucune donnée à transmettre côté client.

Gestion des données : L'ajout

5.6. Le script /administration/categorie/ajout/index.js

Il faut commencer par mettre en place les balises qui afficheront les éventuels messages d'erreur en dessous des champs. Cette action est automatisée à l'aide de la fonction **configurerFormulaire()** du module **fonction/formulaire.js** qui va parcourir tous les champs de l'interface pour y placer juste après une balise `<div class='messageErreur'></div>`. La classe 'messageErreur' est générée dynamiquement par la fonction `configurerFormulaire()`.

```
configurerFormulaire();
```

la fonction `filtrerLaSaisie()` du même composant permet de filtrer les caractères acceptés dans les champs

```
filtrerLaSaisie('nom', /[A-Za-z0-9 ]/);  
filtrerLaSaisie('id', /[A-Za-z0-9]/);  
filtrerLaSaisie('ageMin', /[0-9]/);  
filtrerLaSaisie('ageMax', /[0-9]/);
```

Attention : l'utilisation de cette fonction rend impossible le copier/coller dans le champ.

Le bouton 'Ajouter' (événement click) doit commencer par une mise en forme des données saisies.

Le champ id est mis en majuscule et les éventuels espaces superflus sont retirés

Le champ nom doit être écrit avec la première lettre en majuscule

```
id.value = id.value.trim().toUpperCase();  
nom.value = ucFirst(nom.value.trim());
```

La fonction `ucFirst` appartient au module `format.js`

Ensuite il faut vérifier les règles de format (pattern, minlength, maxlength) en appelant la fonction `donneesValides` du module `contrôle.js` qui se charge de contrôler chaque champ et d'afficher sous le champ l'erreur rencontrée

```
if (!donneesValides()) {  
    return;  
}
```

La vérification suivante concerne les règles métiers nous savons déjà qu'elles sont prises en charge au niveau de la base données au travers des déclencheurs, des contraintes d'intégrité, d'unicité

Nous pouvons donc appeler la fonction `ajouter`

```
ajouter()
```

La fonction `ajouter` doit appeler le script standard `/ajax/ajouter.php` en lui passant les paramètres attendus : `table`, `id`, et `nom`

L'appel est réalisé par la fonction `appelAjax()` de la bibliothèque `fonction/ajax.js`. Cette fonction utilise l'API `fetch` pour réaliser l'appel Ajax et prend en charge toute la gestion d'erreur.

Gestion des données : L'ajout

L'étude du script standard `ajax/ajouter.php` montre que la réponse au format Json comporte soit une clé 'success' soit une clé 'error'.

La clé 'error' est gérée en interne par la fonction `appelAjax` qui affichera alors l'erreur ou les erreurs correspondantes

Rappel : on peut trouver les sous-propriétés suivantes :

- **'global'** : Lorsque le message ne s'adresse pas à un champ particulier. Ce message sera affiché au niveau de la balise `<div id=msg>` si elle existe sur l'interface client ou dans le cas contraire dans une fenêtre modale à l'aide de la fonction `afficherErreur()` de la bibliothèque de fonction `fonction/afficher.js`
- **'system'** : Lorsque le message d'erreur est lié à une erreur technique non prévue (détectée dans un bloc `try catch`). Dans ce cas le message sera affiché dans la console et un message 'non technique' sera affiché dans une fenêtre modale pour prévenir l'utilisateur. Le message d'erreur est aussi enregistré dans le journal 'erreur.log' stocké dans le répertoire `backoffice/.log` (ce répertoire étant créé automatiquement)
- **Toute valeur correspondant à un champ de saisie** : Dans ce cas le message sera affiché sous le champ concerné

La syntaxe de la fonction `appelAjax` est la suivante :

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

Coté développeur, il suffit donc d'appeler la fonction `appelAjax` en lui passant l'url, les données et la fonction à exécuter en cas de succès, ici nous retournons vers la page affichant les catégories afin de voir immédiatement le résultat.

```
function ajouter() {  
    appelAjax({  
        url: '/ajax/ajouter.php',  
        data: {  
            table: 'categorie',  
            id: id.value,  
            nom: nom.value,  
            ageMin: ageMin.valueAsNumber,  
            ageMax: ageMax.valueAsNumber  
        },  
        success: () => {  
            retournerVers("Catégorie enregistrée", "/consultation/categorie");  
        }  
    });  
}
```

La propriété `valueAsNumber` utilisable sur un champ de type 'number' permet de récupérer une valeur numérique.

1. Compléter le script `index.js` et réaliser les tests fonctionnels nécessaires, après avoir relancer le script `insert.sql` afin de ne pas tenir compte des catégories ajoutées lors de la phase de contrôle sous Postman

Gestion des données : L'ajout

6. Ajout d'un coureur

Nous allons étudier un second exemple afin de montrer la prise en compte au niveau de la classe Table des champs facultatifs (colonne avec la contrainte null) et des contraintes d'intégrité référentielle.

6.1. Le déclencheur associé

L'étude du déclencheur 'avantAjoutCoureur' permet de lister les contrôles pris en charge à ce niveau

Colonne	Contrôle
licence	format : <code>^[0-9]{6,7}\$</code> unicité : <code>if exists(select 1</code>
nom prenom	Mise en majuscule : <code>set new.nom = ucase(new.nom);</code> Longueur : entre 3 et 30 Format : <code>^[A-Za-z](?[A-Za-z])*\$</code>
dateNaissance	Format : <code>^(19 20)[0-9]{2}-(0[1-9] 1[0-2])-(0[1-9] [12][0-9] 3[01])\$</code>
idCategorie	Intégrité : recherche à partir de l'âge du coureur
	Unicité du triplet nom, prenom et dateNaissance
idClub	Intégrité : recherche dans la table club <code>if exists(select 1</code>
email	Si renseigné : format et unicité
telephone	Si renseigné : format

6.2. La classe métier Coureur

Le constructeur permet de définir les colonnes à l'aide d'objets 'Input'. La colonne idCategorie dont la valeur est déduite de la date de naissance est automatiquement calculée par le déclencheur 'avantAjoutCoureur', elle n'est donc pas reprise dans le constructeur

Colonne	Objet	Propriété	Valeur
licence	InputText	Require MinLength MaxLength Pattern	true 6 7 <code>^[0-9]{6,7}\$</code>
nom prenom	InputText	Require MinLength MaxLength Pattern EnMajuscule SupprimerAccent SupprimerEspaceSuperflu	true 3 30 <code>^[A-Z](?[A-Z]+)*\$</code> true true true
sexe	InputList	Values EnMajuscule	M ou F true
dateNaissance	InputDate	Require Min Max	true <code>Categorie::getDateNaissanceMax();</code> <code>Categorie::getDateNaissanceMin();</code>
idClub	InputList	Values	<code>select id from club</code>
email	InputEmail	Require MaxLength	false 100
telephone	InputText	Require Pattern MinLength -MaxLength	false <code>^0(6 7)[0-9]{8}</code> 10

Gestion des données : L'ajout

6.3. Vérification des contrôles à l'aide de Postman

Dans le dossier 'Ajout', ajouter la requête 'coureur'

url : <http://gestion/ajax/ajouter.php>

body : form-data

key : table (coureur), licence, nom, prenom, dateNaissance, dexe, idClub ffa, email, telephone

- Contrôle ° 1 : des valeurs non valides
 - licence : avec des lettres, nom et prénom avec des chiffres, idClub inexistant, ffa : 2
- Exemple de réponse attendue

```
{"error":{
  "licence":"Veuillez respecter le format demandé ^[0-9]{6,7}$",
  "nom":"Veuillez respecter le format demandé ^[A-Z](?[A-Z]+)*$",
  "prenom":"Veuillez respecter le format demandé ^[A-Z](?[A-Z]+)*$",
  "sexe":"Veuillez entrer une des valeurs acceptées",
  "dateNaissance":"non-respect du format attendu",
  "ffa":"Veuillez entrer une des valeurs acceptées"}
}
```

Contrôle n° 2 : une date de naissance hors intervalle

```
{"error":{"dateNaissance":"La date doit être égale ou antérieure au 31\12\2014"}}
```

Contrôle n° 3 : un numéro de club inexistant : 080999 et 80021

Erreur déclenchée par l'objet InputList et/ou par le trigger avantAjoutCoureur

```
{"error":{"idClub":"Veuillez entrer une des valeurs acceptées"}}
{"error":{"global":"Ce club n'existe pas."}}
```

Contrôle n° 4 : Ajout validé avec ou sans téléphone et email:

```
{"success":"Opération réalisée avec succès"}
```

Contrôle n° 5 : Contrôle unicité (relancer la même requête)

```
{"error":{"global":"Ce numéro de licence est déjà attribué"}}
```

Contrôle n° 6 : Contrôle unicité (relancer la même requête en changeant le numéro de licence)

```
{"error":{"global":"Un homonyme est déjà présent dans la table"}}
```


Gestion des données : L'ajout

6.4. L'interface : /administration/licencie/ajout/index.html

Étape 1 / 3 Suivant ▷

Identité du licencié

Nom (30 caractères maximum, caractères autorisés : lettres non accentuées et espace) *

Dupont

Prénom (30 caractères maximum, caractères autorisés : lettres non accentuées et espace) *

Hervé

Sexe

Homme

Date de naissance (La date doit être comprise entre le 01/01/1926 et le 31/12/2014)

31/12/2014

Étape 2 / 3 ◁ Précédent Suivant ▷

Informations du licencié

Licence (Numéro composé de 6 ou 7 chiffres) *

000000

Club *

ALBERT MEAULTE AEROSPA.AC

Validé par le Web Service de la FFA ☐

Étape 3 / 3 ◁ Précédent

Informations optionnelles

Adresse mail

Téléphone portable (10 chiffres, 06XXXXXXX ou 07XXXXXXX)

+ Ajouter

Au niveau de la date de naissance, l'intervalle des dates valides est dynamique. Il est calculé à partir des catégories. La valeur des attributs min et max seront donc récupérés côté serveur, puis envoyés côté client

La mise en place des étapes est simple : il suffit d'encapsuler chaque étape dans une balise <div id='etape'>

Les règles de validités sont les suivantes :

Champ	Nom et prénom	Licence	telephone
Type	text	text	number
required	true	true	false
pattern	^[A-Za-z](?[A-Za-z])*\$	^[0-9]{6,7}\$	^0(6 7)[0-9]{8}\$
maxlength	30	6	10
minlength	3	7	10

Gestion des données : L'ajout

Champ	sexe
Type	select
Valeur	M et F

Champ	dateNaissance
Type	date
required	Déterminé dynamiquement en fonction de la date courante et des catégories
min	Déterminé dynamiquement en fonction de la date courante et des catégories
max	Déterminé dynamiquement en fonction de la date courante et des catégories

Champ	club
Type	select
Valeur	Alimenté dynamiquement à partir de la table club

Champ	ffa
Type	checkbox

Champ	mail
Type	text (préféré à email pour une expression régulière de vérification plus précise)
pattern	^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*\.[a-zA-Z]{2,4}\$

6.5. Le script /administration/licencie/ajout/index.php

Il doit récupérer les clubs afin d'alimenter la zone de liste idClub

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

Il doit récupérer l'intervalle des dates de naissance acceptées

```
$dateMin = json_encode(Categorie::getDateNaissanceMin());  
$dateMax = json_encode(Categorie::getDateNaissanceMax());
```

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

L'utilisation de `json_encode()` pour toutes les valeurs injectées en JavaScript y compris les simples chaînes de caractères garantit :

- l'échappement des caractères spéciaux,
- une conversion syntaxiquement correcte (strings entre guillemets, booléens/objets bien formatés, etc.)

Cela évite aussi de devoir délimiter la valeur par des guillemets ou apostrophe qui ouvre la possibilité d'une faille de sécurité.

2. Compléter le script index.php

Gestion des données : L'ajout

6.6. Le script /administration/licencie/ajout/index.js

Le script index.php a transmis côté client les clubs et l'intervalle des dates de naissance.

La mise en place du système d'étapes s'effectue simplement en lançant la fonction *initialiserEtapes()*;

La fonction *configurerFormulaire()* est appelée pour ajouter sous les champs les balises qui afficheront les éventuels messages d'erreur.

Il faut alimenter la zone de liste à l'aide de la variable *lesClubs*

```
for (const club of data)
  idClub.add(new Option(club.nom, club.id));
```

Des filtres sur la saisie de certains champs sont mis en place à l'aide de la fonction *filtrerLaSaisie* :

```
filtrerLaSaisie('telephone', /[0-9]/);
filtrerLaSaisie('licence', /[0-9]/);
filtrerLaSaisie('nom', /[A-Za-z'-]/);
filtrerLaSaisie('prenom', /[A-Za-zÀÁÂÃÄÅÇÈÉÊËÌÍÎÏÐÓÔÕÖÙÚÛÜÝàáâãäåçèéêëìíîïðóôõöùúûüýÿ'-]/);
```

Il faut alimenter les attributs *min* et *max* du champ *dateNaissance* et afficher cet intervalle au niveau du label. Ce traitement est pris en charge par la fonction *configurerDate(input, min, max, valeur)* du composant *contrôle.js*. Elle renseigne les 2 attributs de la balise 'dateNaissance' mais aussi complète le label associé en lui ajoutant un commentaire indiquant l'intervalle dans le format français.

```
configurerDate(dateNaissance, {
  min: dateMin,
  max: dateMax,
  valeur: dateMax
});
```

Le bouton 'btnAjouter' doit lancer la fonction *ajouter* si les données sont valides. Certaines mises en forme peuvent être appliquées initialement comme la suppression des espaces superflus, la mise en majuscule, ou la suppression des accents.

```
btnAjouter.onclick = () => {
  // mise en forme des données
  nom.value = enleverAccent(supprimerEspace(nom.value)).toUpperCase();
  prenom.value = enleverAccent(supprimerEspace(prenom.value)).toUpperCase();
  // contrôle des champs de saisie
  msg.innerHTML = '';
  if (donneesValides()) {
    ajouter();
  }
};
```

La mise en forme du nom et du prénom utilise les fonctions *enleverAccent* et *supprimerEspace* du composant *format.js*.

La fonction *ajouter()* réalise un appel Ajax du script *ajax/ajouter.php* afin d'ajouter ce nouveau coureur.

Gestion des données : L'ajout

Les champs 'email' et 'téléphone' étant optionnel, il faut passer par un objet FormData pour transmettre uniquement les données renseignées.

```
function ajouter() {
    // Alimentation de l'objet formData pour le transfert des données
    const formData = new FormData();
    formData.append('table', 'coureur');
    formData.append('nom', nom.value);
    formData.append('prenom', prenom.value);
    formData.append('sexe', sexe.value);
    formData.append('dateNaissance', dateNaissance.value);
    formData.append('licence', licence.value);
    formData.append('idClub', idClub.value);
    formData.append('ffa', ffa.checked ? '1' : '0');

    // prise en compte des champs optionnels
    if (email.value.length > 0) {
        formData.append('email', email.value);
    }
    if (telephone.value.length > 0) {
        formData.append('telephone', telephone.value);
    }
    // demande d'ajout dans la base de données
    appelAjax({
        url: '/ajax/ajouter.php',
        data: formData,
        success: (data) => {
            retournerVersApresConfirmation(data.success, '/consultation/coureur');
        }
    });
}
```

3. Compléter le script index.js et réaliser tous les tests fonctionnels

4. Supprimer le déclencheur 'avantAjoutCoureur' en exécutant la commande 'drop trigger if exists avantAjoutCoureur;' (script declencheur.sql) et relancer les tests fonctionnels pour voir les différences.

Gestion des données : L'ajout

7. Ajout d'une annonce

Il est souvent nécessaire de permettre aux utilisateurs de mettre en forme le texte saisi depuis un champ de type 'textarea' sans que cela nécessite des connaissances en HTML.

Il existe différents composants permettant de le faire. Ils sont couramment utilisés dans les systèmes de gestion de contenu (CMS), les forums, les blogs et d'autres applications web.

Ils incluent la mise en forme du texte, le support des médias incorporés, la gestion des listes, la création de liens, la gestion des images, etc. Ils offrent en quelque sorte une expérience de type traitement de texte dans un navigateur web.

Les plus connus sont CkEditor et TinyMCE, on peut citer aussi Quill ou Summernote. Nous utiliserons cette année TinyMCE car la version 4 de CkEditor n'est plus maintenue et sa version 5 n'offre pas les mêmes avantages et ne semble plus gratuite.

7.1. La classe métier

La table annonce comprend les colonnes suivantes : id, date, nom, description, actif
Le champ id est du type auto-incrément. Il n'est donc pas repris dans la classe.

Colonne	Objet	Propriété	Valeur
nom	InputText	Require MinLength MaxLength Pattern	true 8 100 ^[A-Za-zÀÇÈÉÊàâäåçèéêî]([A-Za-zÀÇÈÉÊàâäåçèéêî]+)*\$
description	InputTextarea	Require	true
date	InputDate	Require Min Max	true date("Y-m-d") date("Y-m-d", strtotime("+1 year"))
actif	InputInt	Min Max	0 1

Colonne modifiable en mode colonne : actif

7.2. L'interface : /administration/annonce/ajout/index.html

Étape 1 / 2

Suivant >

Date (La date doit être comprise entre le 17/07/2025 et le 17/07/2026) *

08/03/2026

Nom *

COURSE SOLIDAIRE FEMININE

Étape 2 / 2 < Précédent

Description de l'épreuve *

Paragraph B I U A v v v v

La 16ème édition de la Course Solidaire Féminine se déroulera dans le cadre de la Journée Internationale des Droits de la Femme, le dimanche 8 mars 2026, au parc de la Hotoie d'Amiens. Cette course est organisée par l'US Camon Athlétisme. L'objectif de cette course est d'encourager la pratique du sport chez les femmes. Pour cette raison, la distance est volontairement limitée à 5 km afin que les débutantes puissent en faire un premier objectif atteignable en un temps de préparation raisonnable.



p > img Build with tinyMCE

+ Ajouter

Url de l'image : <https://www.uscathle.org/evenements/solidaire/images/2025.jpg>

7.3. Le script /administration/annonce/ajout/index.php

Il doit charger le composants tinyMce

```
$head = <<<HTML
  <script src="/composant/tinymce/tinymce.min.js" referrerpolicy="origin"></script>
HTML;
```

5. Compléter le script index.php

7.4. Le script /administration/annonce/ajout/index.js

Il initialise le composant tinyMce et les attributs min, max, valeur du champ de type date à l'aide de la fonction configurerDate(). La fonction getDateRelative de la bibliothèque date.js permet de fixer les valeurs de min, max et valeur en fonction des règles établies : entre aujourd'hui et dans un an

```
const min = getDateRelative("annee", 0); // un an avant
const max = getDateRelative("annee", 1); // un an après
const valeur = getDateRelative("mois", 2); // une annonce est souvent réalisée 2 mois à l'avance

configurerDate(date, {
  min: min,
  max: max,
  valeur: valeur
});
```

Gestion des données : L'ajout

Le composant tinyMce est initialisée

```
// Initialisation de TinyMCE
tinymce.init({
  license_key: 'gpl',
  selector: '#description',
  height: 300,
  menubar: false,
  plugins: 'link lists image',
  toolbar: [
    'styles | bold italic underline | forecolor backcolor | fontselect | link | image | bullist
    numlist'
  ],
  fontsize_formats: '8pt 10pt 12pt 14pt 18pt 24pt 36pt',
});
```

Pour récupérer la valeur du champ 'description' géré par le composant :

```
btnAjouter.onclick = () => {
  // récupération de la valeur du composant tinymce
  description.value = tinymce.get('description').getContent();
  if (donneesValides()) {
    ajouter();
  }
};
```

L'ajout suit le même principe que dans l'ajout d'un coureur.

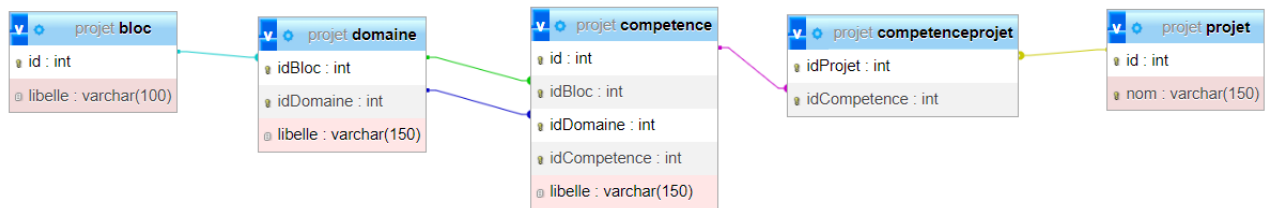
```
function ajouter() {
  appelAjax({
    url: '/ajax/ajouter.php',
    data: {
      table: 'annonce',
      nom: nom.value,
      description: description.value,
      date: date.value
    },
  },
  success: () => {
    retournerVersApresConfirmation("Annonce enregistrée", "/consultation/annonce/");
  }
});
}
```

6. Compléter le script index.js

Gestion des données : L'ajout

8. Ajout d'un projet et de ses compétences

Cet exemple peut être repris dans le cadre de votre portfolio dans lequel vous devez indiquer l'ensemble des projets réalisés au cours de vos deux années de formation en indiquant les compétences du référentiel mises en œuvre :



La création d'un projet se traduit au niveau de la base de données par l'ajout d'un enregistrement dans la table projet et par l'ajout de plusieurs enregistrements dans la table competenceProjet.

Il n'est donc pas possible d'utiliser le script /ajax/ajouter.php pour 2 raisons :

- La table competenceProjet possède une clé compose : idProjet, idCompétence
- les deux opérations ne doivent pas être dissociées,

Pour être certains d'exécuter toutes les instructions 'insert' comme un seul bloc, il faut mettre en place une transaction.

Une transaction permet :

- d'exécuter un groupe d'opérations SQL comme une seule unité logique ;
- d'assurer la cohérence et l'intégrité des données, même en cas d'erreur ou d'interruption inattendue ;
- de garantir l'atomicité : soit toutes les opérations sont réalisées avec succès, soit aucune modification n'est appliquée à la base.

En cas de succès, on valide les opérations avec COMMIT.

En cas d'erreur, on annule tout changement avec ROLLBACK.

Gestion des données : L'ajout

8.1. La classe métier **Projet** (classemetier/projet.php)

La classe **Projet** doit prendre en charge l'enregistrement d'un projet et de ses compétences.

La méthode `enregistrer()` permet de le faire en mettant en place une transaction afin de garantir l'atomicité de l'opération.

```
public static function enregistrer(string $nom, array $lesCompetences): string {
    $db = Database::getInstance();

    // Démarrage d'une transaction
    $db->beginTransaction();
    // ajout du projet
    $sql = <<<SQL
        insert into projet(nom) values(:nom);
SQL;
    $c = $db->prepare($sql);
    $c->bindValue('nom', $nom);
    try {
        $c->execute();
    } catch (Exception $e) {
        $db->rollBack();
        return $e->getMessage();
    }
    // récupération de l'id attribué au projet
    $idProjet = $db->lastInsertId();

    // ajout des compétences du projet
    foreach ($lesCompetences as $idCompetence) {
        $sql = <<<SQL
            insert into competenceprojet(idProjet, idCompetence) values(:idProjet, :idCompetence);
SQL;
        $c = $db->prepare($sql);
        $c->bindValue('idProjet', $idProjet);
        $c->bindValue('idCompetence', $idCompetence);
        try {
            $c->execute();
        } catch (Exception $e) {
            $db->rollBack();
            return $e->getMessage();
        }
    }
    $db->commit();
    return "Opération réalisée avec succès";
}
```

Gestion des données : L'ajout

8.2. Les déclencheurs

Les contrôles sont mis en place sous la forme de triggers de type before à la fois sur la table projet et sur la table competenceprojet.

Pour le projet il faut enlever les espaces superflus, vérifier le nombre de caractères et vérifier que le nom du projet n'est pas déjà utilisé.

```
create trigger avantAjoutProjet before insert on Projet
for each row
begin
    # mise en forme et vérification du nom
    set new.nom = TRIM(REGEXP_REPLACE(new.nom, '\\s+', ' '));
    if length(new.nom) not between 10 and 150 then
        signal sqlstate '45000' set message_text = "#Le nom du projet doit comporter entre 10 et 150 caractères";
    end if;
    if exists(select 1 from projet where nom = new.nom) then
        signal sqlstate '45000' set message_text = "#Ce projet existe déjà";
    end if;
end; niveau
```

Pour l'ajout d'une compétence dans le projet, il faut vérifier que :

- Le projet et la compétence existe ;
- La compétence est bien une compétence du bloc 1 ;
- Cette compétence n'a pas déjà été ajoutée dans le projet.

```
create trigger avantAjoutCompetenceprojet before insert on competenceprojet
for each row
begin
    declare existeCompetence int default 0;
    declare bloc int default 0;
    declare message text;
    -- vérifie que le projet cible existe
    if not exists (select 1 from projet where id = new.idProjet) then
        signal sqlstate '45000' set message_text = "#Ce projet n'existe pas";
    end if;
    -- vérifie que la compétence existe et récupère son idbloc
    select count(*), ifnull(idbloc, 0) into existeCompetence, bloc
    from competence
    where id = new.idCompetence;
    if existeCompetence = 0 then
        set message = concat('#La compétence ', new.idCompetence, ' n'existe pas');
        signal sqlstate '45000' set message_text = message;
    end if;
    -- vérifie que la compétence appartient au bloc 1
    if bloc != 1 then
        set message = concat('#La compétence ', new.idCompetence, ' ne fait pas partie du bloc 1. ');
        signal sqlstate '45000' set message_text = message;
    end if;
end
```

Gestion des données : L'ajout

8.3. L'interface

Nous utilisons encore une fois le composant `etape.js` pour obtenir une saisie par étape mais avec ici l'intégration d'une étape récapitulative.

La première étape consiste à saisir le nom du projet

Étape 1 / 3

Suivant ▷

Nom du projet * Mon nouveau projet

L'étape suivante permet de sélectionner les compétences. Au vu du nombre restreint de compétences, une représentation de chaque compétence par une case à cocher est la meilleure solution, elle évite à l'utilisateur de sélectionner plusieurs fois la même compétence et permet facilement de retirer une compétence.

Étape 2 / 3

< Précédent Suivant ▷

Liste des compétences du bloc 1 couvertes par ce projet

☒ Recenser et identifier les ressources numériques
☒ Exploiter des référentiels, normes et standards adoptés par le prestataire informatique
☐ Mettre en place et vérifier les niveaux d'habilitation associés à un service
☐ Vérifier les conditions de la continuité d'un service informatique
☐ Gérer des sauvegardes
☒ Vérifier le respect des règles d'utilisation des ressources numériques
☐ Collecter, suivre et orienter des demandes

Il suffit de cliquer sur une case pour sélectionner/désélectionner une compétence.

L'identifiant de chaque compétence sélectionnée est conservé dans un tableau `lesCompetencesDuProjet`.

Ce tableau sera transmis dans son format Json au script php chargé d'enregistrer le nouveau projet

L'étape finale, récapitule les informations sur le projet (nom + compétences) et permet de valider la saisie

Étape 3 / 3

< Précédent

Résumé du projet

Nom du projet :

Compétences sélectionnées :

Recenser et identifier les ressources numériques
Exploiter des référentiels, normes et standards adoptés par le prestataire informatique
Vérifier le respect des règles d'utilisation des ressources numériques

Enregistrer le projet et ses compétences

Gestion des données : L'ajout

Le bouton de validation 'btnEnregistrer' réalise un appel Ajax du script `ajax/enregistrer.php` afin d'enregistrer le projet avec l'ensemble de ses compétences.

La particularité repose ici sur le fait de devoir transmettre les compétences stockées côté client dans le tableau `lesCompetencesDuProjet`.

Le transfert entre le client et le serveur ne pouvant se faire que dans un format texte, il faut convertir ce tableau dans le format JSON.

Cette opération s'effectue très facilement en utilisant la méthode `stringify` de la classe `JSON` : `JSON.stringify(lesCompetencesduProjet)`

Côté serveur la chaîne JSON reçue est convertie en tableau à l'aide de la fonction `json_decode` : `json_decode($_POST['lesCompetences'])`

8.4. Le script `ajax/enregistrer.php`

Il n'est pas possible d'utiliser les méthodes héritées de la classe `Table` pour deux raisons :

- L'opération doit se dérouler dans une transaction puisqu'elle concerne l'ajout dans plusieurs tables : projet + `competenceProjet`
- La clé primaire de la table `competenceProjet` est composée de deux champs : `idProjet` et `idCompetence`

Il faut donc prendre en charge la totalité des opérations d'ajout ainsi que des nombreux contrôles à réaliser :

- Les paramètres doivent être transmis : le nom du projet et le tableau ;
- Le nom du projet doit être composé d'au moins 10 caractères sans dépasser 150 ;
- *Le nom doit être unique ;*
- Au moins une compétence doit être transmise ;
- *Chaque compétence doit être numérique et doit correspondre à une compétence du bloc 1 présente dans la table 'competence' ;*

Les contrôles en italique sont pris en charge par les déclencheurs **`avantAjoutProjet`** et **`avantAjoutCompetenceProjet`**

Il reste à vérifier que les paramètres sont bien transmis et qu'au moins une compétence est transmise.

Commençons par la vérification de la transmission des paramètres attendus `idProjet` et `lesCompetences`.

La classe technique `Std` propose un ensemble de méthodes facilitant le contrôle des données côté serveur.

```
if (!Std::existe('nom', 'lesCompetences')) {  
    Erreur::envoyerReponse('Requête invalide, des paramètres manquent.', 'global');  
}
```

Les informations transmises sont alors récupérées et éventuellement mises en forme ou filtrées afin d'éviter toute injection SQL ou faille XSS.

Gestion des données : L'ajout

```
// récupération et filtrage des paramètres
$nom = trim($_POST['nom']);
// supprimer toutes les balises HTML et PHP
$nom = strip_tags($nom),
// récupération des compétences dans un tableau
$lesCompetences = json_decode($_POST['lesCompetences']);
```

Si la conversion échoue, les compétences transmises ne respectent pas le format attendu

```
if (is_null($lesCompetences)) {
    Erreur::envoyerReponse('Le format de transmission des compétences n\'est pas conforme',
'global');
}
```

Il faut avoir transmis au moins une compétence :

```
if (count($lesCompetences) === 0) {
    Erreur::envoyerReponse('Le projet doit posséder au moins une compétence', 'global');
}
```

Au niveau du projet, nous vérifions que le nom n'est pas vide et qu'il n'est pas déjà présent dans la table projet

```
if (empty($nom)) {
    Erreur::envoyerReponse('Le nom du projet doit être renseigné', 'nom');
}

if (Projet::getByName($nom)) {
    Erreur::envoyerReponse('Le nom du projet est déjà utilisé', 'nom');
}
```

Nous pouvons maintenant appeler la méthode enregistrer de la classe projet

```
$sreponse = Projet::enregistrer($nom, $lesCompetences);
if ($sreponse === "Opération réalisée avec succès") {
    echo json_encode($sreponse);
} else {
    Erreur::envoyerReponse($sreponse);
}
```

7. Compléter le script ajax/enregistrer.php.

Gestion des données : L'ajout



8.5. Vérification des contrôles à l'aide de Postman

Ajouter la requête 'Projet' dans le dossier 'Ajout' de la collection 'Gestion'

url : `http://gestion/ajout/projet/ajax/enregistrer.php`

méthode : `post`

form-data

	Key		Value
	nom	Text	nouveau projet
	lesCompetences	Text	[1, 2, 10, '99']

Réponse : `{"error":{"global":"Le format de transmission des compétences n'est pas conforme"}}`

Test avec `lesCompetences = [1, 2, 10, 99]`

Réponse : `{"error":{"global":"La compétence 99 n'existe pas"}}`

Test avec `lesCompetences = [1, 2, 10, 23]`

Réponse : `{"error":{"global":"La compétence 23 ne fait pas partie du bloc 1."}}`

Test avec `nom = 'court'`

Réponse : `{"error":{"global":"Le nom du projet doit comporter entre 10 et 150 caractères"}}`

Test avec `nom = 'nouveau projet'` et `lesCompetences = [1, 2, 10]`

8.6. Le script `/administration/projet/index.php`

Le script doit récupérer et transmettre côté client les compétences du bloc 1

```
$lesCompetencesBloc1 = json_encode(Competence::getLesCompetences(1));

$head = <<<HTML
    <script>
        const lesCompetencesBloc1 = $lesCompetencesBloc1;
    </script>
HTML;
```

8. Compléter le script `index.php`

8.7. Le script `/administration/projet/index.js`

A l'issue du chargement de la page, il faut générer les cases à cocher pour chaque compétence.

Le clic sur une case doit déclencher la mise à jour du tableau `lesCompetencesduProjet` : un ajout si la case est cochée et une suppression si la case est décochée.

En même temps la balise `<tbody id='lesLignes'>` présente sur l'étape 3 doit être mise à jour de la même façon (ajout ou suppression d'une ligne)

```
for (const competence of lesCompetencesBloc1) {
  let div = document.createElement('div');
  div.classList.add("d-flex", "mb-1");
  let uneCase = document.createElement('input');
  uneCase.type = 'checkbox';
  uneCase.classList.add("form-check-input", "my-auto", "m-3");
  uneCase.style.width = '25px';
  uneCase.style.height = '25px';
  // pour permettre de récupérer toutes les cases
  uneCase.name = 'competence';
  uneCase.onclick = function () {
    let action = uneCase.checked ? "add" : "delete";
    if (action === "add") {
      // ajout de la compétence dans le tableau
      lesCompetencesDuProjet.push(competence.id);
      // ajout d'une ligne dans la liste des compétences sélectionnées
      const tr = lesLignes.insertRow();
      tr.id = competence.id;
      tr.insertCell().innerText = competence.libelle;

    } else {
      // suppression de la compétence du tableau
      const index = lesCompetencesDuProjet.findIndex(id => id === competence.id);
      lesCompetencesDuProjet.splice(index, 1);
      // suppression de la ligne dans la liste des compétences sélectionnées
      document.getElementById(competence.id)?.remove();
    }
  };
  div.appendChild(uneCase);
  let label = document.createElement('label');
  label.innerText = competence.libelle;
  label.classList.add("my-auto");
  div.appendChild(label);
  listeCompetence.appendChild(div);
}
```

L'événement clic du bouton enregistrer procède aux vérifications avant de lancer la fonction enregistrer

```
btnEnregistrer.onclick = () => {
  // vérification sur le champ nom
  if (donneesValides()) {
    // il faut au moins une compétence
    if (lesCompetencesDuProjet.length === 0) {
      messageBox('Il faut associer au moins une compétence au projet', 'error');
    } else {
      enregistrer();
    }
  } else {
    messageBox("Des données sont manquantes ou incorrectes, veuillez vérifier votre saisie",
    'error');
  }
};
```

Gestion des données : L'ajout

La fonction `enregistrer()` réalise un appel Ajax du script `ajax/enregistrer.php`.

Il faut envoyer le nom du projet et le tableau des compétences. Ce dernier doit être converti en chaîne JSON afin de pouvoir être transmis

```
function enregistrer() {  
    msg.innerHTML = "";  
    appelAjax({  
        url: 'ajax/enregistrer.php',  
        data: {  
            nom: nom.value,  
            lesCompetences: JSON.stringify(lesCompetencesDuProjet)  
        },  
        success: () => retournerVers('Projet enregistré', '/consultation/projet')  
    });  
}
```

9. Compléter le script `index.js` et réaliser les tests fonctionnels.