Libs Docs

Bastien Helec

09/07/2025

PHP_FORMULAIRE

```
Bastien Helec
Description : Explication des classes FORMULAIRES de la section PHP/FORMULAIRES

09-07-2025
```

Sommaire

Model

FRONT

:

Description :

La partie FRONT des formulaires PHP permet de construire dynamiquement des formulaires complexes, en combinant différents types de champs, listes et éléments, avec ou sans interaction avec la base de données. Chaque classe représente un composant réutilisable pour la génération et la gestion des formulaires.

- Form.php ⇒ Form
- Fields.php ⇒ Fields
- Liste.php ⇒ Liste
- ListeBDD.php ⇒ ListeBDD

- FormElement.php ⇒ FormElement

Form :

Description :

La classe `Form` permet de créer un formulaire complet en regroupant différents éléments (champs, listes, etc.). Elle centralise la gestion des éléments et fournit les données structurées du formulaire.

Arguments du constructeur :

- `$idFormulaire` : Identifiant unique du formulaire.
- `$DefaultbtnStatus` : Booléen indiquant si le bouton par défaut est actif (true par défaut).

Exemple d'utilisation :

```php
<?php
$form = new Form('form_contact');
$field = new Fields('nom', 'text', 'Votre nom', 'required');
$form->addElement($field);
$data = $form->getData();
?>
```

Cela produira :

voir la section vue a ce propos :

Vue Form

Méthodes principales :

- `addElement(FormElement $element)` : Ajoute un élément (champ, liste, etc.) au formulaire.
- `getData()` : Retourne un tableau structuré contenant tous les éléments du formulaire a utilisé dans la vue.

Fields :

Description :

La classe `Fields` représente un champ de formulaire classique (input, textarea, etc.), avec ses propriétés et attributs HTML.

Arguments du constructeur:

`$nom` : Nom du champ. `$type` : Type du champ (text, email, etc.). `$placeholder` : Texte d'exemple affiché dans le champ. `$attribut` : Attributs HTML supplémentaires ('required', 'readonly'). `$fields_type` : Type spécifique si besoin (optionnel). `$value` : Valeur par défaut (optionnel).

Exemple d'utilisation:

```php
<?php
$field = new Fields('email', 'email', 'Votre email', 'required');
$data = $field->getData();
?>
```

Cela produira :

voir la section vue a ce propos :

Vue Fields

Méthodes principales :

`getData()` : Retourne les propriétés du champ sous forme de tableau associatif.

Liste :

Description :

La classe `Liste` permet de créer une liste de choix (type select ou menu) sans interaction avec la base de données, à partir d'un tableau de valeurs fourni.

Arguments du constructeur :

`$nom_btn` : Nom du bouton associé à la liste. `$nom_liste` : Nom de la liste. `$param_liste` : Tableau des valeurs à afficher dans la liste.

Exemple d'utilisation :

```php
<?php
$liste = new Liste('btn_choix', 'liste_couleurs', ['Rouge', 'Vert', 'Bleu']);
$data = $liste->createListe('form_couleurs');
?>
```

Cela produira :

voir la section vue a ce propos :

Vue Liste

Méthodes principales :

`createListe($idFormulaire)` : Retourne un tableau structuré pour générer la liste dans le formulaire.

ListeBDD :

Description :

La classe `ListeBDD` permet de créer une liste de choix alimentée dynamiquement depuis la base de données, en utilisant la classe `Libelle` pour récupérer les valeurs.

Arguments du constructeur :

`$nom_btn` : Nom du bouton associé à la liste.

`$nom_liste` : Nom de la liste.

`$nom_colonne` : Colonne de la table à utiliser pour les valeurs.

`$nom_table` : Table de la base de données à interroger.

`$pdo` : Instance PDO pour la connexion à la base de données.

Exemple d'utilisation :

```php
<?php
$listeBDD = new ListeBDD('btn_choix', 'liste_categories', 'nom', 'categories', $pdo);
$data = $listeBDD->createListe('form_categories');
?>
```

Cela produira :

voir la section vue a ce propos :

Vue ListeBDD

Méthodes principales :

`createListe($idFormulaire)` : Retourne un tableau structuré pour générer la liste dans le formulaire, avec les valeurs issues de la BDD.

`getData()` : Retourne les données de la liste pour interaction avec le formulaire.

FormElement :

Description :

`FormElement` est une interface qui impose la méthode `getData()` à toutes les classes d'éléments de formulaire, assurant ainsi une structure homogène pour la récupération des données de chaque composant.

Méthodes principales :

`getData()` : Retourne les données de l'élément sous forme de tableau.

Back

:

Description :

La partie BACK des formulaires PHP gère le traitement, la validation, l'insertion et la gestion des données envoyées par les formulaires côté serveur. Elle centralise la logique métier, la communication avec la base de données, la gestion des retours utilisateur (bannières, messages) et l'intégration avec des services externes (ex : Nextcloud).

C'est le "Back-end" l'avantage ici est que le back n'est pas statiques, il sera générer en fonction des besoins.

- Form.php ⇒ BaseForm
- Auth.php ⇒ Auth
- Glob_Handling.php ⇒ Glob_Handling
- Delete.php ⇒ Delete
- Update.php ⇒ Update
- Nextcloud_calendar.php ⇒ Nextcloud_calendar

BaseForm :

Description :

La classe `BaseForm` sert de classe de base pour tous les traitements de formulaires côté back. Elle gère l'initialisation des propriétés communes comme l'identifiant de la bannière, le message de retour, la détection de la méthode POST et la connexion PDO.

Arguments du constructeur :

- `$id_banner` : Identifiant de la bannière à afficher côté client.
- `$message` : Message à afficher à l'utilisateur.
- `$pdo` : Instance PDO pour la connexion à la base de données.

Auth :

Description :

La classe `Auth` hérite de `BaseForm` et gère l'authentification des utilisateurs. Elle vérifie les identifiants (email et mot de passe) reçus via POST, compare le mot de passe avec le hash stocké en base, et retourne un message de succès ou d'erreur, ainsi que les informations utilisateur si la connexion réussit.

Exemple d'utilisation :

```php
<?php
$auth = new Auth('banner_connexion', '', $pdo);
$auth->FORM_Connect(
    'id,prenom,nom,email,password', // Colonnes à récupérer
    'prenom', 'nom', 'email',       // Champs pour le prénom, nom, email
    'email', 'password', 'password',// Champs POST pour email et mot de passe
    'utilisateurs'                  // Table cible
```

```
);
?>
```

Il n'y a pas de vue comme c'est gérer en back-end

Méthodes principales :

- `FORM_Connect(...)` : Traite la connexion utilisateur, vérifie les identifiants, gère la session et retourne un message JSON structuré.

Delete :

Description :

La classe `Delete` hérite de `BaseForm` et permet de gérer la suppression d'une entrée dans la base de données via un formulaire. Elle vérifie la confirmation de suppression, exécute la requête SQL correspondante et retourne un message JSON de succès ou d'erreur, accompagné d'une bannière d'information.

Exemple d'utilisation :

```php
<?php
$delete = new Delete('banner_suppression', '', $pdo);
$delete->set_delete(
    'categories',           // Table cible
    "id = 5",               // Condition de suppression
    'confirmer_suppression' // Nom du champ de confirmation dans le formulaire
);
?>
```

Il n'y a pas de vue comme c'est gérer en back-end

Méthodes principales :

`set_delete($db_table, $cdt, $name_suppresion, $cdt_plus = "", $cdts = "")` : Vérifie la confirmation, exécute la suppression et retourne un message JSON structuré.

Update :

Description :

La classe `Update` hérite de `BaseForm` et permet de gérer la modification d'une entrée dans la base de données via un formulaire. Elle construit la requête SQL d'UPDATE, exécute la modification et retourne un message JSON de succès ou d'erreur, accompagné d'une bannière d'information.

Exemple d'utilisation :

```php
<?php
$update = new Update('banner_modification', '', $pdo);
$update->set_update(
    ['nom' => 'Nouveau Nom'], // Tableau associatif colonne => valeur à mettre à jour
    'categories',             // Table cible
    '',                       // Jointure éventuelle
    "id = 5"                  // Condition de mise à jour
);
?>
```

Il n'y a pas de vue comme c'est gérer en back-end

Méthodes principales :

`set_update($after_set, $db_table, $join = "", $cdt_simple ="" , $sup_cdts = "")` : Exécute la mise à jour et retourne un message JSON structuré.

Glob_Handling :

Description :

La classe `Glob_Handling` hérite de `BaseForm` et permet de gérer dynamiquement l'insertion de données dans la base, avec gestion des traitements spécifiques (ex : hash de mot de passe), génération de messages de retour et affichage de bannières de succès ou d'erreur.

Exemple d'utilisation:

```php
<?php
$handler = new Glob_Handling('banner_ajout', '', $pdo);
$handler->FORM_Interact(
    'categories',        // Table cible
    'nom',               // Colonne(s)
    ['nom'],             // Champs du formulaire
    ['password' => 'password_hash'] // Traitement spécifique
);
?>
```

Il n'y a pas de vue comme c'est gérer en back-end

Méthodes principales:

`FORM_Interact($db_table, $db_columns, array $f_name, $F_action=[], $cdt_plus=null, $cdts_db=null)` : Gère l'insertion des données du formulaire dans la base, applique les traitements nécessaires, et retourne un message JSON structuré.

`$f_name` : Tableau (Array) qui contient les nom des champs.

`$F_action` : Tableau (Array) qui vient effectuer des actions sur des type de champs spécifiques.

Nextcloud_Calendar :

++***WORK IN PROGRESS***++

Description :

La classe `Nextcloud_calendar` hérite de `BaseForm` et permet de créer et d'envoyer des événements au format iCalendar (ICS) vers un serveur Nextcloud via WebDAV.

Elle gère la génération de l'événement, la connexion cURL sécurisée et le retour d'état.

Arguments du constructeur :

`$calendar_base_url_dav` : URL de base WebDAV du calendrier Nextcloud.

`$calendar_name` : Nom du calendrier cible.

`$objet` : Objet (titre) de l'événement.

`$start_date`, `$end_date` : Dates de début et de fin de l'événement.

`$username` : Nom d'utilisateur Nextcloud.

Exemple d'utilisation :

```php
<?php
$calendar = new Nextcloud_calendar(
    'https://nextcloud.example.com/remote.php/dav/calendars',
    'agenda_salle',
    'Réunion importante',
```

```
    '2025-07-10 14:00:00',
    '2025-07-10 16:00:00',
    'Salle_Agenda'
);
$calendar->send();
?>
```

Il n'y a pas de vue comme c'est gérer en back-end

Méthodes principales :

`send()` : Génère l'événement et l'envoie à Nextcloud via cURL, puis affiche un message de succès ou d'erreur dans la console. Exemple d'utilisation :

Vue-Implementation

:

VUE FRONT :

Cette section présente des exemples d'implémentation côté vue pour l'affichage dynamique des formulaires et de leurs éléments, en cohérence avec la structure générée par les classes du **Model FRONT**.

Form.view.php :

Description :

Affichage d'un formulaire complet à partir des données structurées générées par la classe `Form`.
Chaque élément du formulaire (champ, liste, etc.) est affiché selon son type.

Ce qu'il fait :

- Affichage conditionnel : Le formulaire n'est généré que si `$formData` n'est pas vide, sinon un message indique qu'il n'y a rien à afficher.
- Titre dynamique : Le titre du formulaire est généré à partir de l'identifiant, chaque partie séparée par un underscore étant mise en majuscule.
- Structure du formulaire : Le formulaire est construit dynamiquement à partir du tableau `$formData['elements']`.
  - Champs classiques : Les champs sont générés selon leurs propriétés (`type`, `nom`, `placeholder`, etc.).
  - Les attributs HTML comme `required` ou `readonly` sont ajoutés si spécifiés.
  - Les champs spéciaux comme `<br>` ou `<hr>` sont gérés.
  - Les champs de type `number` ont un minimum fixé à `1`.
- Listes dynamiques (ListeBDD) :
  - Les listes sont affichées sous forme de `<div>` contenant un bouton et une liste `<ul>` d'options.
  - Chaque option est un `<li>` avec un identifiant et une valeur, les underscores sont remplacés par des espaces pour l'affichage.
  - Un champ caché `<input type="hidden">` est ajouté pour stocker la valeur sélectionnée.
- Bouton d'envoi :
  - Un bouton « Envoyer » est affiché si la propriété `DefaultbtnStatus` est vraie.
- Accessibilité et personnalisation :

– Les identifiants et noms des champs sont générés dynamiquement pour garantir l'unicité et faciliter la gestion côté JS/PHP.

```php
<?php
if (!empty($formData)) {
    echo '<div class="Formulaire" id="'.$formData['id']. '_div">';
    $Titre_Array = explode('_', $formData['id']);
    $Titre = '';
    foreach ($Titre_Array as $cmp_titre){
        $Titre .= ucfirst($cmp_titre) . ' ';
    }
    $Titre = trim($Titre);
    echo '<h2 style="text-align:center;"> '.$Titre.'</h2>';
    echo "<form id='{$formData['id']}_form'>";

    foreach ($formData['elements'] as $element) {
        // Affichage des champs classiques
        if (isset($element['nom'])) {
            $attribut = '';
            if (isset($element['attribut']) && $element['attribut'] === 'requis') {
                $attribut = 'required';
            }
            if (isset($element['attribut']) && $element['attribut'] === 'readonly') {
                $attribut = 'readonly';
            }
            $value = '';
            if ($element['value'] !== '') {
                $value = 'value="'.$element['value'].'"';
            }
            if (isset($element['fields_type']) && $element['fields_type'] === '') {
                echo '<input type= "'.$element['type']. '" name= "'. $element['nom']. '" id= "'.$formData[
            }
            elseif (isset($element['fields_type']) && $element['fields_type'] === 'br') {
                echo '<br>';
            }
            elseif (isset($element['fields_type']) && $element['fields_type'] === 'hr') {
                echo '<hr>';
            }
            elseif ($element['type'] == 'number') {
                echo '<input type= "'.$element['type']. '" min="1" '.$value.' name= "'. $element['nom']. '
            }
            else {
                echo '<'.$element['fields_type'].' type= "'.$element['type']. '" name= "'. $element['nom']
            }
        }
        // Affichage des listes dynamiques (ListeBDD)
        elseif (isset($element['nom_liste'])) {
            echo '<div class="select" id= '.$formData['id'].'_' . $element['nom_liste'] . '_select>';
            echo '<button id="'.$formData['id'].'_'.$element['nom_liste'].'_choix_btn" type="button">' . h
            echo '<ul id= "'.$formData['id']. '_' . $element['nom_liste'] . '_options">';
            foreach ($element['libelle'] as $opts) {
                $libelle = str_replace('_', ' ', $opts['libelle']);
                echo "<li id = '{$opts['libelle']}' data-value='{$opts['libelle']}'>" . htmlspecialchars($
            }
            echo '</ul>';
            echo '<input type="hidden" id= "'.$formData['id']. '_' . $element['nom_liste']. '_choix_valeur
            echo '</div>';
        }
```

```
    }
    if ($formData['DefaultbtnStatus'] === true){
        echo '<button type="submit" id="'.$formData['id'].'_submit">Envoyer</button>';
    }
    echo '</form>';
    echo '</div>';
} else {
    echo '<p>Aucun élément à afficher.</p>';
}
?>
```

Fields.view.php :

Description :

Affichage d'un ou plusieurs champs de formulaire classiques (input, textarea, etc.), générés par la classe `Fields`.

Ce qu'il fait:

Ce code PHP affiche dynamiquement un ou plusieurs champs de formulaire à partir d'un tableau `$fields`.

Pour chaque champ (`$Field`), il vérifie si le champ est requis (`requis`) et ajoute l'attribut HTML `required` si besoin. Ensuite, il distingue deux cas :

- Si le type de champ (`fields_type`) n'est pas `input` , il génère un champ `<input>` classique avec les attributs appropriés (`type` , `name` , `placeholder` , etc.).
- Sinon, il génère un élément HTML du type spécifié par `fields_type` (par exemple, `textarea`), en lui appliquant également les bons attributs.

Chaque champ est donc affiché avec ses propriétés et contraintes, ce qui permet de générer des formulaires personnalisés et dynamiques.

```
<?php
if (!empty($fields)) {
    foreach ($fields as $Field) {
        $requis = ($Field['requis'] === 'requis') ? 'required' : '';
        if ($Field['fields_type'] !== 'input') {
            echo '<input type="' . htmlspecialchars($Field['type']) . '" name="' . $idFormulaire . '_' . h
        } else {
            echo '<'.$Field['fields_type'] .'  type="' . htmlspecialchars($Field['type']) . '" name="' . $
        }
    }
}
?>
```

ListeBDD.view.php :

Description :

Affichage d'une liste de choix alimentée dynamiquement depuis la base de données, générée par la classe `ListeBDD`.

Ce qu'il fait:

Ce code PHP affiche dynamiquement une liste de choix (type menu déroulant ou liste d'options) alimentée depuis la base de données, à partir du tableau `$listeBDD`.

- **Condition d'affichage** : La liste n'est générée que si `$listeBDD` n'est pas vide.
- **Structure HTML** :

  - Un `<div>` avec une classe `select` et un identifiant unique basé sur le formulaire et le nom de la liste.
  - Un bouton `<button>` qui affiche le nom du bouton associé à la liste.
  - Une liste non ordonnée `<ul>` contenant chaque option de la liste.
- **Pour chaque option** :
  - La valeur de la colonne est échappée pour la sécurité HTML.
  - Les underscores (`_`) sont remplacés par des espaces pour un affichage plus lisible.
  - Chaque option est un `<li>` avec un identifiant et un attribut `data-value` correspondant à la valeur réelle.
- **Champ caché** :
  - Un `<input type="hidden">` est ajouté pour stocker la valeur sélectionnée par l'utilisateur, ce qui facilite la récupération côté JavaScript ou lors de la soumission du formulaire.

Ce code permet donc d'afficher une liste dynamique, personnalisée et sécurisée, directement liée aux données de la

```php
<?php
if (!empty($listeBDD)) {
    echo "<div class='select' id='".$idFormulaire."_".$nom_liste."_select'>
    <button id='".$idFormulaire."_".$nom_liste."_choix_btn'>".$nom_btn."</button>
    <ul id='".$idFormulaire."_".$nom_liste."_options'>";
    foreach ($listeBDD as $row) {
        $value = htmlspecialchars($row[$colonne]);
        $libelle = str_replace('_', ' ', $value);
        echo "<li id='$value' data-value='$value'>$libelle</li>";
    }
    echo "</ul>
    <input type='hidden' id='".$idFormulaire."_".$nom_liste."choix_valeur' id='".$idFormulaire."_name_choi
    </div>";
}
?>
```

Toutes les vues peuvent être utilisé entre elle (elles sont d'ailleurs pensé pour).