Formulaires, récupération et manipulations de données

Rappel : les formulaires en HTML

L'objet de cette partie n'est pas d'apprendre à créer des formulaires en HTML ni de découvrir les différents éléments de formulaire puisque, en principe, vous connaissez déjà le HTML et savez donc créer un formulaire en HTML.

Pour la bonne compréhension de tous, néanmoins, nous allons dans cette partie effectuer quelques rappels sur l'utilité des formulaires et la création de formulaires en HTML.

Ensuite, dans le reste de cette partie, nous allons nous intéresser à comprendre comment on va pouvoir récupérer les données envoyées via un formulaire en PHP et apprendre à les manipuler et à stocker ces données.

Pour ce faire, nous allons utiliser un formulaire très simple comme base de travail.

Définition et rôle des formulaires HTML

Les formulaires HTML vont permettre de recueillir des données envoyées par l'utilisateur. Les formulaires vont se révéler indispensables pour tout site proposant aux utilisateurs de s'inscrire et de se connecter, et vont être l'élément privilégié pour permettre aux utilisateurs d'envoyer, par exemple, un message via un site.

Les formulaires HTML vont pouvoir être composés de champs de texte *(cas d'un champ de formulaire demandant à un utilisateur de renseigner son adresse mail pour se connecter ou pour s'inscrire sur un site par exemple)*, de listes d'options *(choix d'un pays dans une liste de pays par exemple)*, de cases à cocher, etc.

L'intérêt et le rôle principal des formulaires HTML va résider dans le fait que les formulaires vont permettre de transmettre des données. En effet, une fois que l'utilisateur a fini de remplir un formulaire, il va pouvoir le soumettre *(l'envoyer, le "submit")*. Les données envoyées vont ensuite pouvoir être stockées ou traitées / manipulées *(on va par exemple pouvoir vérifier que le couple identifiant / mot-de-passe d'un utilisateur souhaitant se connecter sont valides)*.

L'élément form et ses attributs

Pour créer un formulaire, nous allons utiliser l'élément HTML form. Cet élément form va avoir besoin de deux attributs pour fonctionner normalement : les attributs method et action.

L'attribut method va indiquer comment doivent être envoyées les données saisies par l'utilisateur. Cet attribut peut prendre deux valeurs : get et post.

Que signifient ces deux valeurs et laquelle choisir ? Les valeurs get et post vont déterminer la méthode de transit des données du formulaire. En choisissant get, on indique que les données doivent transiter via l'URL *(sous forme de paramètres)* tandis qu'en choisissant la valeur post on indique qu'on veut envoyer les données du formulaire via transaction post HTTP.

Concrètement, si l'on choisit l'envoi via l'URL *(avec la valeur get)*, nous serons limités dans la quantité de données pouvant être envoyées et surtout les données vont être envoyées en clair. **Évitez donc absolument d'utiliser cette méthode si vous demandez des mots de passe ou toute information sensible dans votre formulaire.**

Cette méthode de transit est généralement utilisée lors de l'affichage de produits triés sur un site e-commerce *(les options de tris sont des éléments de formulaires)*. Regardez bien les URL la prochaine fois que vous allez sur un site e-commerce après avoir fait un tri : si vous retrouvez des éléments de votre tri dedans c'est qu'un get a été utilisé.

En choisissant l'envoi de données via post transaction HTTP *(avec la valeur post)*, nous ne sommes plus limités dans la quantité de données pouvant être envoyées et les données ne sont visibles par personne. **C'est donc généralement la méthode que nous utiliserons pour l'envoi véritablement de données que l'on souhaite stocker** *(création d'un compte client, etc.)*.

L'attribut action va lui nous servir à préciser l'adresse relative du script *(de la page)* qui va traiter les données. En effet, nous ne pouvons pas effectuer les opérations de traitement ou de stockage des données avec le HTML. Pour faire cela, nous devrons utiliser des langages comme le PHP *(pour le traitement)* et le MySQL *(pour le stockage)* par exemple.

Création d'un formulaire HTML

Nous allons créer un petit formulaire en HTML qui va nous servir pour le reste de cette partie.

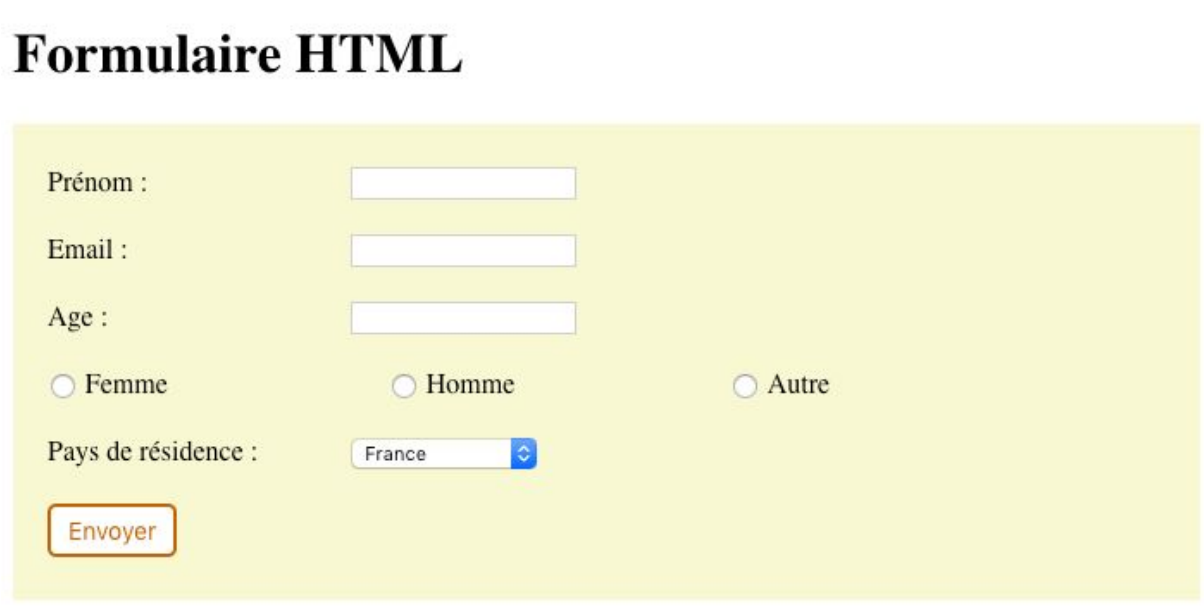
Ce formulaire va nous permettre de récupérer les informations suivantes :

- Le prénom d'un utilisateur ;
- Son adresse mail ;
- Son âge ;
- Son sexe ;
- Son pays de résidence.

Voici les codes HTML et CSS que nous allons utiliser pour ce formulaire ainsi que le résultat obtenu :

```
<form action="formulaire.php" method="post">
    <div class="c100">
        <label for="prenom">Prénom : </label>
        <input type="text" id="prenom" name="prenom">
    </div>
    <div class="c100">
        <label for="mail">Email : </label>
        <input type="email" id="mail" name="mail">
    </div>
    <div class="c100">
        <label for="age">Age : </label>
        <input type="number" id="age" name="age">
    </div>
    <div class="c100">
        <input type="radio" id="femme" name="sexe" value="femme">
        <label for="femme">Femme</label>
        <input type="radio" id="homme" name="sexe" value="homme">
        <label for="homme">Homme</label>
        <input type="radio" id="autre" name="sexe" value="autre">
        <label for="autre">Autre</label>
    </div>
    <div class="c100">
        <label for="pays">Pays de résidence :</label>
        <select id="pays" name="pays">
            <optgroup label="Europe">
                <option value="france">France</option>
                <option value="belgique">Belgique</option>
                <option value="suisse">Suisse</option>
            </optgroup>
            <optgroup label="Afrique">
                <option value="algerie">Algérie</option>
                <option value="tunisie">Tunisie</option>
                <option value="maroc">Maroc</option>
                <option value="madagascar">Madagascar</option>
                <option value="benin">Bénin</option>
                <option value="togo">Togo</option>
            </optgroup>
            <optgroup label="Amerique">
                <option value="canada">Canada</option>
            </optgroup>
        </select>
    </div>
    <div class="c100" id="submit">
        <input type="submit" value="Envoyer">
    </div>
</form>
```

```css
form{
    width: 100%;
    background-color: rgba(220,220,0,0.2);
    padding : 5px 0px;
}
.c100{
    width: 100%;
    margin: 20px;
}
label{
    display: inline-block;
    min-width: 25%;
}
input[type="submit"]{
    color: RGB(200,100,0);
    border-radius: 5px;
    padding: 5px 10px;
    font-size: 14px;
    border: 2px solid RGB(200,100,0);
}
input[type="submit"]:hover{
    background-color: RGB(200,100,0);
    color: #fff;
    cursor: pointer;
    box-shadow: 0px 0px 5px 0px #777;
}
```



Vous pouvez récupérer les codes via ce lien :

- https://codepen.io/isl-php/pen/zYBZdMj

Ici, on utilise des éléments input pour demander le prénom, le mail et l'âge de l'utilisateur. On ajuste la valeur de l'attribut type en fonction du type de données attendues *(texte, nombre, etc...)*.

On utilise également un input type="radio" pour le choix du sexe. L'utilisation de boutons radio est ici toute indiquée puisque l'utilisateur ne devrait pouvoir faire qu'un choix dans la liste d'options et car cela va grandement faciliter le traitement des données par la suite puisque nous allons toujours recevoir soit la valeur *« femme »* soit la valeur *« homme »* soit *« autre »*.

Enfin, nous utilisons un élément select pour le choix des pays avec différents éléments option pour chaque option. Pour l'exemple, nous groupons les pays par continent avec l'élément optgroup.

Notez qu'on utilise également à chaque fois des éléments label pour indiquer à l'utilisateur ce qu'il doit renseigner comme information. On lie également chaque label à son champ de formulaire grâce aux attributs for du label et id de l'input ou du select en leur donnant la même valeur.

Côté mise en forme et CSS, on va se contenter du strict minimum puisqu'encore une fois ce n'est pas l'objet de mon cours. Nous allons ici nous servir de nos *div class="c100"* pour mettre chaque champ de formulaire sur une nouvelle ligne et allons mettre rapidement en forme les autres éléments.

Notez bien ici les valeurs données aux attributs method et action de notre élément form car c'est cela qui va particulièrement nous intéresser.

Ici, on indique qu'on choisit d'envoyer nos données par transaction http de type **post**. Nous allons envoyer les données vers le script *formulaire.php*. C'est ce script là qui va s'occuper du traitement des données du formulaire et sur lequel nous allons nous concentrer par la suite.

Si vous cliquez sur le bouton de validation, vous pouvez observer que vous êtes renvoyés vers cette page *(ce script)*. Cependant, comme nous n'avons pas encore créé le script *(la page)*, vous devriez avoir une erreur de type *« page introuvable »*.

Nous avons là la base, nous allons maintenant apprendre à récupérer et à manipuler les données envoyées via les formulaires en PHP.

Récupérer et manipuler les données des formulaires HTML en PHP

Comme nous l'avons vu précédemment, le HTML nous permet de créer des formulaires. Par contre, pour récupérer et manipuler les données envoyées, nous allons devoir utiliser du PHP.

Dans cette partie, nous allons voir comment récupérer et manipuler les données récoltées via les formulaires.

Les superglobales $_POST et $_GET

Dans la partie précédente, nous avons créé un formulaire dans une page qui s'appelait formulaire.html.

Notre élément form possédait les attributs suivants :

- method="post"
- action="formulaire.php"

Cela signifie que les données vont être envoyées via transaction post http à la page *(ou au script)* formulaire.php.

La première chose à comprendre ici est que toutes les données du formulaire vont être envoyées et être accessibles dans le script PHP mentionné en valeur de l'attribut action, et cela quelle que soit la méthode d'envoi choisie *(post ou get)*.

En effet, le PHP possède dans son langage deux variables superglobales $_GET et $_POST qui sont des variables tableaux et dont le rôle va justement être de stocker les données envoyées via des formulaires.

Plus précisément, la superglobale $_GET va stocker les données envoyées via la méthode get et la variable $_POST va stocker les données envoyées via la méthode post.

> Les valeurs vont être stockées **sous forme d'un tableau associatif** c'est-à-dire sous la forme clef => valeur où la clef va correspondre à la valeur de l'attribut name d'un champ de formulaire et la valeur va correspondre à ce qui a été rempli *(ou coché, ou sélectionné)* par l'utilisateur pour le champ en question.

A noter : On va également pouvoir utiliser la variable superglobale $_REQUEST pour accéder aux données d'un formulaire sans se soucier de la méthode d'envoi. Cependant, utiliser **$_REQUEST ne présente généralement que peu d'intérêt en pratique et peut potentiellement ouvrir des failles de sécurité dans nos formulaires. C'est la raison pour laquelle je n'en parlerai pas plus dans ce cours.**

Affichage simple des données de formulaire reçues

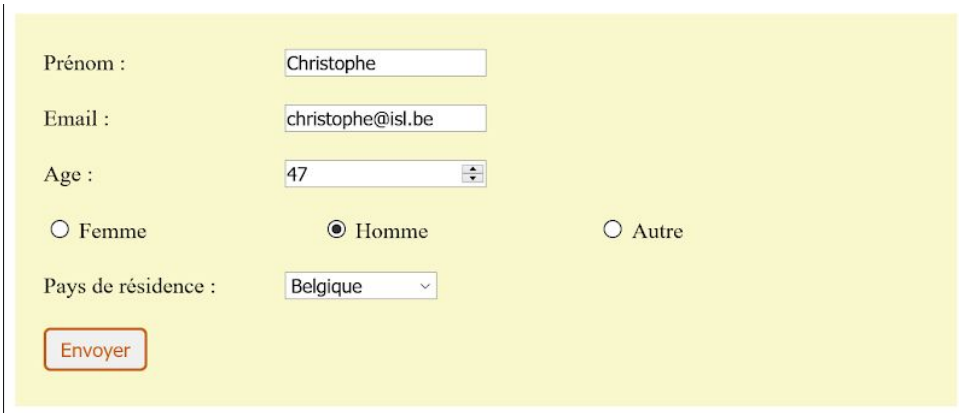
Comme $_GET et $_POST sont des variables superglobales, elles seront toujours accessibles n'importe où dans le script par définition.

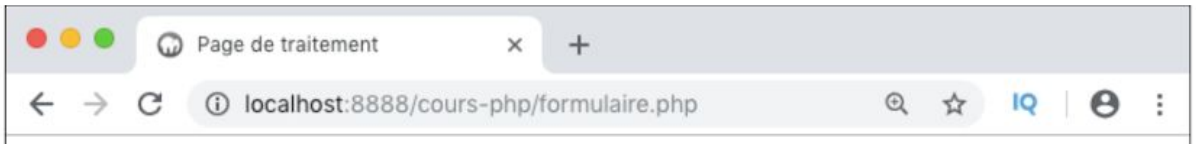
On va alors très facilement pouvoir accéder aux données envoyées dans les formulaires en parcourant nos variables superglobales $_GET ou $_POST.

Par exemple, on va pouvoir très simplement afficher à l'utilisateur les données reçues. Pour cela, nous allons echo les valeurs contenues dans $_POST via notre page d'action formulaire.php *(du moins en théorie)* :

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Page de traitement</title>
        <meta charset="utf-8">
    </head>
    <body>
        <p>Dans le formulaire précédent, vous avez fourni les
        informations suivantes :</p>

        <?php
            echo 'Prénom : '.$_POST["prenom"].'<br>';
            echo 'Email : ' .$_POST["mail"].'<br>';
            echo 'Age : ' .$_POST["age"].'<br>';
            echo 'Sexe : ' .$_POST["sexe"].'<br>';
            echo 'Pays : ' .$_POST["pays"].'<br>';
        ?>
    </body>
</html>
```





Dans le formulaire précédent, vous avez fourni les informations suivantes :

Prénom : Christophe
Email : christophe@isl.be
Age : 47
Sexe : homme
Pays : belgique

En pratique, cependant, nous n'allons pas créer des formulaires pour afficher les données aux utilisateurs mais bien pour utiliser les données de notre côté. Généralement, après l'étape de récupération des données, on affichera une informations à l'utilisateur comme quoi ses données ont bien été transmises ou enregistrées ou … *(à voir en fonction des besoins)*.

Manipulation et stockage de données

Sur les sites Internet, les formulaires sont utilisés en général pour effectuer différents types d'opération comme :

- Donner la possibilité à un utilisateur de s'inscrire ;
- Donner la possibilité à un utilisateur de se connecter ;
- Permettre à un utilisateur de poster un commentaire ;
- Permettre à un utilisateur d'envoyer un message *(idem lorsque vous envoyez un email, vous remplissez un formulaire dont les données vont être transmises à la personne que vous désirez contacter)* ;
- Etc.

Dans chacun de ces cas, nous allons manipuler les données envoyées dans des buts différents : enregistrer les données pour une inscription, vérifier les données de connexion envoyées, enregistrer et afficher un commentaire, etc.

Sécurisation et validation des formulaires en PHP

Il est primordial de comprendre l'importance de la validation des données envoyées par les formulaires et allons voir comment mettre en place un premier système de validation de ces données.

Ne jamais faire confiance aux données utilisateurs

La sécurisation des formulaires est un aspect essentiel de la création de ceux-ci.

Lorsqu'on crée des formulaires, c'est généralement pour demander aux utilisateurs de nous envoyer des données. Si on ne met pas en place des systèmes de filtre sur le type de données qui peuvent être envoyées pour chaque champ et de vérification ensuite de la qualité des données envoyées, les données récoltées vont alors pouvoir être aberrantes ou même potentiellement dangereuses.

En effet, sans contrainte sur les données qui peuvent être envoyées, rien n'empêche un utilisateur d'envoyer des données invalides, comme par exemple un prénom à la place d'une adresse email ou un âge de 2000 ans ou encore de tenter de nous envoyer un script potentiellement dangereux.

Ici, il va falloir faire la différence entre deux types d'utilisateurs qui vont être gérés de façons différentes :

1. Les utilisateurs maladroits qui vont envoyer des données invalides par mégarde ;
2. Les utilisateurs malveillants qui vont tenter d'exploiter des failles de sécurité dans nos formulaires pour par exemple récupérer les données personnelles d'autres utilisateurs.

Pour le premier groupe d'utilisateurs qui ne sont pas mal intentionnés, la première action que nous allons pouvoir prendre va être d'ajouter des contraintes directement dans notre formulaire pour limiter les données qui vont pouvoir être envoyées. Pour cela, nous allons pouvoir utiliser des attributs HTML comme min, max, required, etc. ainsi que préciser les bons types d'input à chaque fois.

Nous allons ensuite également pouvoir tester que les données nous conviennent dès le remplissage d'un champ ou au moment de l'envoi du formulaire grâce au HTML ou au JavaScript (principalement) et bloquer l'envoi du formulaire si des données ne correspondent pas à ce qu'on attend.

Tout cela ne va malheureusement pas être suffisant contre les utilisateurs malintentionnés pour la simple et bonne raison que **n'importe qui peut neutraliser toutes les formes de vérification effectuées dans le navigateur**. Pour cela, il suffit par exemple de désactiver

l'usage du JavaScript dans le navigateur et d'inspecter le formulaire pour supprimer les attributs limitatifs avant l'envoi.

Contre les utilisateurs malveillants, nous allons donc également devoir vérifier les données après l'envoi du formulaire et neutraliser les données potentiellement dangereuses. Nous allons effectuer ces vérifications en PHP, côté serveur.

Ces deux niveaux de vérifications *(dans le navigateur / côté serveur)* doivent être implémentés lors de la création de formulaires. En effet, n'utiliser qu'une validation dans le navigateur laisse de sérieuses failles de sécurité dans notre formulaire puisque les utilisateurs malveillants peuvent désactiver ces vérifications.

N'effectuer qu'une série de vérifications côté serveur, d'autre part, serait également une très mauvaise idée d'un point de vue expérience utilisateur puisque ces vérifications sont effectuées une fois le formulaire envoyé.

Ainsi, que faire si des données aberrantes mais pas dangereuses ont été envoyées par un utilisateur maladroit ? Supprimer les données ? Le recontacter pour qu'il soumette à nouveau le formulaire ? Il est bien plus facile dans ce cas de vérifier directement les données lorsqu'elles sont saisies dans le navigateur et de lui faire savoir si une donnée ne nous convient pas.

La validation des données du formulaire dans le navigateur

Les processus de validation des données que nous allons pouvoir mettre en place dans le navigateur vont s'effectuer avant ou au moment de la tentative d'envoi du formulaire.

L'objectif va être ici de bloquer l'envoi du formulaire si certains champs ne sont pas correctement remplis et de demander aux utilisateurs de remplir correctement les champs invalides.

Nous allons pouvoir faire cette vérification principalement en HTML et / ou en JavaScript.

Le HTML5 propose aujourd'hui des options de validation relativement puissantes et couvrant la majorité de nos besoins. Je vais donc ici n'utiliser que du HTML.

Notez toutefois que si vous utilisez du JavaScript dans vos formulaires pour par exemple modifier les données de la page sans avoir à la rafraîchir, il faudra bien évidemment également sécuriser vos scripts JavaScript *(cela dépassant le cadre de mon cours, nous ne l'aborderons pas)*.

La validation des données en HTML va principalement passer par l'ajout d'attributs dans les éléments de formulaire. Nous allons ainsi pouvoir utiliser les attributs suivants :

Attribut	Définition
size	Permet de spécifier le nombre de caractères dans un champ
minlength	Permet de spécifier le nombre minimum de caractères dans un champ
maxlength	Permet de spécifier le nombre maximum de caractères dans un champ
min	Permet de spécifier une valeur minimale pour un champ de type number ou date
max	Permet de spécifier une valeur maximale pour un champ de type number ou date
step	Permet de définir un multiple de validité pour un champ acceptant des donnés de type nombre ou date. En indiquant step="4" , les nombres valides seront -8, -4, 0, 4, 8, etc.
autocomplete	Permet d'activer l'autocomplétion pour un champ : si un utilisateur a déjà rempli un formulaire, des valeurs lui seront proposées automatiquement lorsqu'il va commencer à remplir le champ
required	Permet de forcer le remplissage d'un champ. Le formulaire ne pourra pas être envoyé si le champ est vide
pattern	Permet de préciser une expression régulière. La valeur du champ devra respecter la contrainte de la regex pour être valide

Reprenons notre formulaire précédent et ajoutons quelques contraintes sur les données que l'on souhaite recevoir :

- Le prénom est désormais obligatoire et ne doit comporter que des lettres + éventuellement des espaces, tirets ou apostrophes. Sa taille ne doit pas accéder 20 caractères ;
- Le mail doit avoir au moins 1 caractère de type lettre ou chiffre + le symbole « @ » + à nouveau au moins 1 caractère de type lettre ou chiffre + le symbole « . » + au moins deux caractères de type lettre ou chiffre. Il est également obligatoire ;
- L'âge doit être un nombre et être compris entre 12 et 99 ans.

Nous allons donc écrire :

```
<div class="c100">
    <label for="prenom">Prénom : </label>
    <input type="text" id="prenom" name="prenom"
           required pattern="^[A-Za-z '-]+$" maxlength="20">
</div>
<div class="c100">
    <label for="mail">Email : </label>
    <input type="email" id="mail" name="mail"
           required pattern="^[A-Za-z]+@{1}[A-Za-z]+\.{1}[A-Za-z]{2,}$">
</div>
<div class="c100">
    <label for="age">Age : </label>
    <input type="number" id="age" name="age" min="12" max="99">
</div>
```

Ici, on utilise l'attribut required pour nos champs « prenom » et « mail » pour indiquer qu'ils doivent être automatiquement remplis.

Ensuite, on utilise maxlength pour limiter la taille de notre champ à 20 caractères. On utilise également min et max pour fournir un intervalle de valeurs valides pour le champ âge.

Finalement, on utilise l'attribut pattern pour ajouter des contraintes sur la forme des données qui doivent être renseignées pour nos champs prenom et mail en fournissant des regex *(expressions régulière)* adaptées *(nous en reparlerons plus tard dans le cours)*.

L'idée n'est pas ici de faire un cours sur les expressions régulières, nous en parlerons plus en détail plus tard dans le cours ou si vous êtes impatient, vous pouvez consulter le document Moodle mis à disposition et qui vous renvoie vers un tuto en ligne.

La validation des données du formulaire sur serveur

La validation dans le navigateur va nous éviter une immense majorité de données invalides et donc d'avoir des données à priori exploitables.

Comme mentionné plus haut, elle n'est pas suffisante contre des utilisateurs malveillants puisque n'importe qui peut neutraliser les attributs HTML ou le JavaScript en les désactivant dans le navigateur avant d'envoyer le formulaire.

Une validation côté serveur, en PHP, va donc s'imposer pour filtrer les données potentiellement dangereuses.

Dans le cadre de ce cours, nous n'aurons pas recours aux validations côté client *(via le navigateur)*. Toutes les validation que nous ferons, nous le ferons côté serveur.