

Implémenter la validation des données en ASP.NET MVC

MOHAMED AIROUCHE

Implémenter la validation des données en MVC

Les données peuvent provenir de nombreuses sources dans une application Web. Vous pouvez les charger à partir de fichiers, les lire à partir d'une base de données ou accepter les valeurs qu'un utilisateur a saisies dans un formulaire. Il est important de vérifier que les données déjà présentes sur votre serveur sont valides.

Vous devez toujours valider les données fournies par les utilisateurs avant de les utiliser dans vos méthodes.

Implémenter la validation des données en MVC

La validation des données d'un formulaire peut s'effectuer à deux niveaux :

- **Au niveau client**

La validation du côté client consiste à vérifier les valeurs des champs du formulaire avant son envoi vers le serveur. Cette validation se fait normalement en JavaScript.

L'avantage de cette validation du côté client est qu'elle permet de ne pas solliciter le serveur tant et aussi longtemps que les valeurs ne sont pas valides à l'intérieur du formulaire.

- **Au niveau serveur**

Malgré une validation du côté client, il est nécessaire de faire une dernière validation du côté serveur avant de traiter l'information.

La validation côté serveur est très importante du point de vue de l'application, car les utilisateurs peuvent contourner la validation côté client.

Implémenter la validation des données en MVC

- ASP.NET MVC utilise les attributs de DataAnnotations pour implémenter les validations. DataAnnotations inclut des attributs de validation intégrés pour différentes règles de validation, qui peuvent être appliqués aux propriétés de la classe de modèle. Le Framework ASP.NET MVC appliquera automatiquement ces règles de validation et affichera les messages de validation dans la vue.

Exemple:

```
[Required(ErrorMessage = "La date est absente !")]  
public DateTime Date { get; set; }
```

- Les attributs DataAnnotations peuvent être appliqués aux propriétés de la classe de modèle pour indiquer le type de donnée que la propriété contiendra.

Exemple:

```
[Display(Name = "Date")]  
[DataType(DataType.Date)]  
public DateTime Date { get; set; }
```

L'attribut DataType spécifie le type des données (Date). Avec cet attribut :

- L'utilisateur n'est pas obligé d'entrer les informations de temps dans le champ de date.
- Seule la date est affichée, pas les informations de temps.

Implémenter la validation des données en MVC

- L' espace de noms *System.ComponentModel.DataAnnotations* inclut les attributs d'annotations des données. Le tableau suivant répertorie quelques attributs de validation d'annotations des données.

Attribut	La description
Required	Indique que la propriété est un champ obligatoire.
StringLength	Définit une longueur maximale pour le champ de chaîne.
Range	Définit une valeur maximale et minimale pour un champ numérique
RegularExpression	Spécifie que la valeur du champ doit correspondre à l'expression régulière spécifiée.
CreditCard	Spécifie que le champ spécifié est un numéro de carte de crédit.
Custom Validation	Méthode de validation personnalisée spécifiée pour valider le champ.

Attribut	La description
EmailAddress	Valide avec le format d'adresse e-mail.
FileExtension	Valide avec l'extension de fichier.
MaxLength	Spécifie la longueur maximale d'un champ de chaîne.
MinLength	Spécifie la longueur minimale d'un champ de chaîne.
Phone	Spécifie que le champ est un numéro de téléphone utilisant une expression régulière pour les numéros de téléphone.

<https://www.tutorialsteacher.com/mvc/implement-validation-in-asp.net-mvc>

Implémenter la validation des données en MVC

- Le Framework ASP.NET MVC offre la possibilité de valider les données des formulaires soumises au contrôleur avant d'être traitées.
- Le Framework ASP.NET MVC simplifier la validation des données en ajoutant des attributs au modèle. La validation se fait du côté serveur, par le contrôleur, qui peut vérifier les données avant de décider si elles sont correctes ou pas.
- Le Framework permet également de réaliser une validation du côté client à l'aide de la librairie JQuery.
- Les attributs de validation sont vérifiés du côté client avant que les valeurs ne soient envoyées sur le serveur, et également sur le serveur avant l'appel de l'action du contrôleur.

Implémenter la validation des données en MVC

- Pour activer la validation du côté client, il faut mettre dans le fichier web.config les deux propriétés ClientValidationEnabled et UnobtrusiveJavaScriptEnabled à true comme indiqué ci-dessous :

```
<appSettings>  
  <add key="ClientValidationEnabled" value="true" />  
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />  
</appSettings>
```

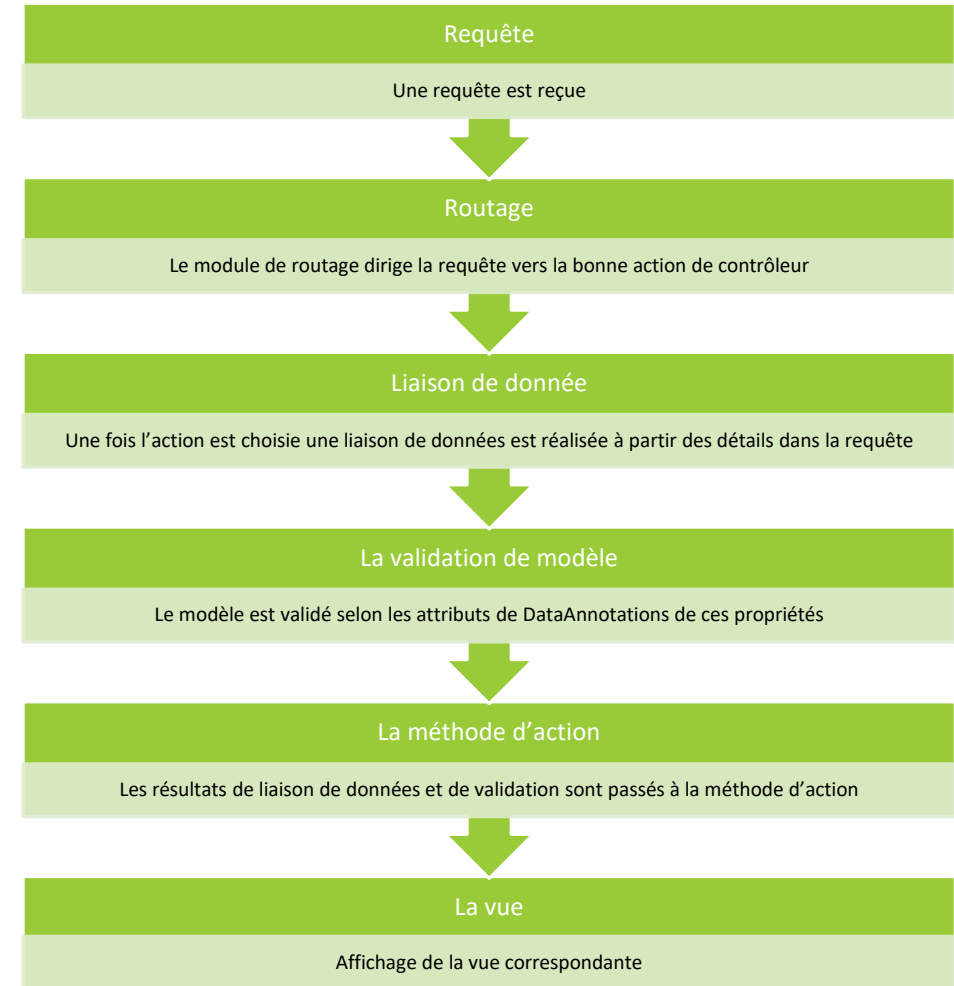
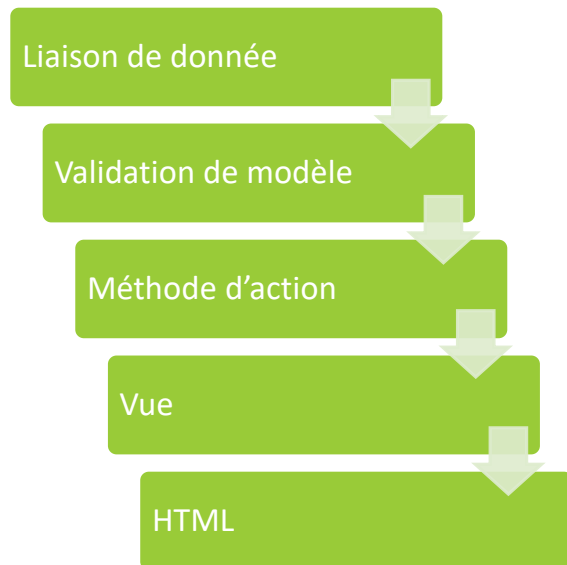
- Par la suite, il faut inclure les références aux fichiers JavaScript suivants :

```
<script src="~/Scripts/jquery-1.10.2.js"></script>  
<script src="~/Scripts/jquery.validate.js"></script>  
<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
```

Implémenter la validation des données en MVC

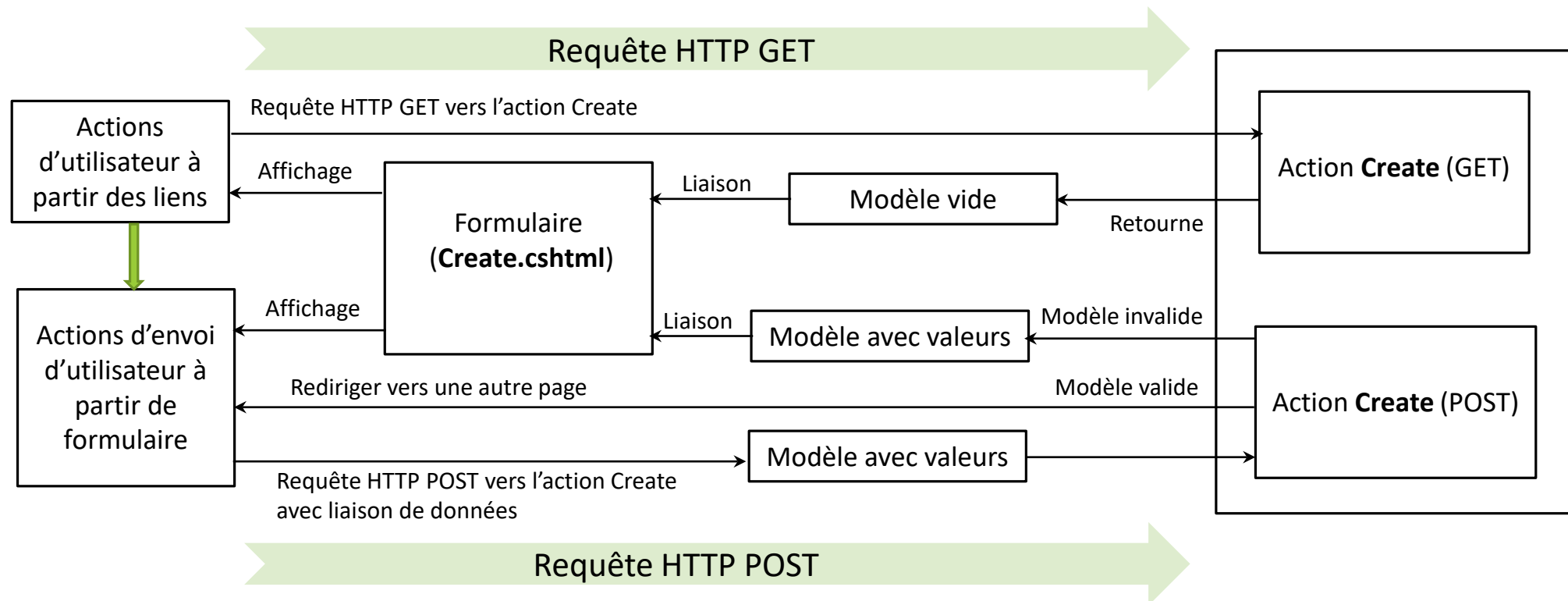
La validation du côté serveur se produit généralement après la liaison de modèle. La méthode d'action s'exécute même si la validation n'a pas réussi.

Les figures suivantes montrent la place de la validation du modèle dans un processus, montrant comment une demande à une page contenant un formulaire qui demande les détails d'un utilisateur est liée et validée.



Liaison et validation des données d'un formulaire

Exemple montrant la validation et la liaison de modèle dans un processus de création d'un élément de modèle utilisant un formulaire (**Create.cshtml**) et deux méthodes d'action **Create** (HTTP GET, HTTP POST).



Implémenter la validation des données en MVC

Exemple :

```
[HttpGet]
public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(Employee employee)
{
    if (ModelState.IsValid)
    {
        employees.Add(employee);
        return RedirectToAction("Index");
    }
    return View();
}
```

La première méthode d'action Create avec HttpGet affiche le formulaire de création initial. La deuxième méthode avec HttpPost gère l'envoi des données du formulaire. La deuxième méthode Create avec HttpPost vérifie si le modèle contient des erreurs de validation avec ModelState.IsValid. L'obtention de cette propriété évalue tous les attributs de validation qui ont été appliqués à l'objet. Si l'objet comporte des erreurs de validation, la méthode Create réaffiche le formulaire. S'il n'y a pas d'erreur, la méthode enregistre l'élément (employee) dans la liste des employés.

Dans l'exemple si on active la validation de côté client, le formulaire n'est pas envoyé au serveur quand des erreurs de validation sont détectées du côté client; la seconde méthode Create n'est jamais appelée. Si la validation du client est désactivée, la propriété ModelState.IsValid de la méthode Create permet de vérifier si le modèle contient des erreurs de validation.

<https://docs.microsoft.com/fr-fr/aspnet/mvc/overview/getting-started/introduction/adding-validation>

Implémenter la validation des données en MVC

- Utilisez les méthodes HTML helpers **ValidationMessageFor** ou **ValidationMessage** pour afficher les messages d'erreur au niveau du champ dans la vue.
- Utilisez la méthode HTML helpers **ValidationSummary** pour afficher tous les messages d'erreur au même endroit dans la vue.
- Vérifiez si le modèle est valide avant la mise à jour dans la méthode d'action à l'aide de `ModelState.IsValid`.

<https://www.tutorialsteacher.com/mvc/implement-validation-in-asp.net-mvc>

Implémenter la validation des données en MVC

Exemple de quelques attributs utilisés pour la validation

Les attributs de validation spécifient le comportement que vous souhaitez appliquer sur les propriétés du modèle sur lesquelles ils sont appliqués. Voici quelques attributs de validation le plus utilisés:

- L'attribut Required
- L'attribut Compare
- L'attribut Range
- L'attribut RegularExpression
- L'attribut Custom

Implémenter la validation des données en MVC

L'attribut Required : cet attribut assure que le champ contient une valeur.

Exemple:

```
public class Model1
{
    [Required(ErrorMessage = "Nom absent !")]
    public string Nom { get; set; }
    [Required(ErrorMessage = "Prénom absent !")]
    public string Prenom { get; set; }
}
```

Nom	<input type="text"/>
	Nom absent !
Prenom	<input type="text"/>
	Prénom absent !
	<input type="button" value="Validation"/>

Implémenter la validation des données en MVC

La méthode **ValidationSummary** est souvent utilisée conjointement avec d'autres attributs pour afficher le sommaire des erreurs. Dans cet exemple, la méthode HTML helpers **ValidationSummary** est utilisée avec l'attribut **Required** pour afficher tous les messages d'erreur au même endroit dans la vue.

Exemple:

```
public class Model1
{
    [Required(ErrorMessage = "Nom absent !")]
    public string Nom { get; set; }
    [Required(ErrorMessage = "Prénom absent !")]
    public string Prenom { get; set; }
}
```

Utilisation de ValidationSummary avec premier paramètre = false

Nom

Nom absent !

Prenom

Prénom absent !

- Nom absent !
- Prénom absent !

Nom

Prenom

Implémenter la validation des données en MVC

L'attribut Compare : cet attribut assure que la valeur d'un champ correspond à une valeur spécifique ou à la valeur d'un autre champ.

Exemple:

```
public class Model2
{
    [Required(ErrorMessage = "Absent !")]
    public string MotPassePrecedent { get; set; }
    [Required(ErrorMessage = "Absent !")]
    public string NouveauMotPasse1 { get; set; }
    [Required(ErrorMessage = "Absent !")]
    [Compare("NouveauMotPasse1", ErrorMessage = "Deux
    nouveaux mots de passe ne concordent pas !")]
    public string NouveauMotPasse2 { get; set; }
}
```

MotPassePrecedent	<input type="text" value="12"/>
NouveauMotPasse1	<input type="text" value="22"/>
NouveauMotPasse2	<input type="text" value="223"/>
Deux nouveaux mots de passe ne concordent pas !	
<input type="button" value="Validation"/>	

Implémenter la validation des données en MVC

L'attribut Range : Cet attribut assure que la valeur est incluse dans un intervalle.

Exemple:

```
public class Model3
{
    [Required(ErrorMessage = "Age absent !")]
    [Range(18,120,ErrorMessage = "L'âge doit être
entier et dans l'intervalle 18 à 120 !")]
    public int Age { get; set; }
}
```

Entrez l'âge de la personne (18 à 120) :

Age

L'âge doit être entier et dans l'intervalle 18 à 120 !

Validation

Implémenter la validation des données en MVC

L'attribut `RegularExpression` : cet attribut permet de valider le contenu d'un champ à partir d'une *expression régulière*.

Le tableau ci-dessous présente quelques caractères (et séquences de caractères) qu'il est possible de retrouver dans une expression régulière.

Caractères	Description http://www.regexlib.com/
<code>^</code>	Début de l'expression
<code>\$</code>	Fin de l'expression
<code>{n}</code>	Répéter « n » fois
<code>C{1,4}</code>	« c », « cc », « ccc » ou « cccc »
<code>[abc]</code>	« a », « b » ou « c »
<code>[a-z]</code>	« a » à « z »
<code>c?</code>	Possiblement le caractère « c »
<code>\d</code>	Un chiffre
<code>\w</code>	Un chiffre ou une lettre
<code>\s</code>	Espace blanc
<code>.</code>	N'importe quel caractère
<code> </code>	Ou logique

Exemple:

```
public class Model4
{
    [Display(Name = "Tel")]
    [DataType(DataType.PhoneNumber)]
    [RegularExpression(@"^\(\d{3}\)\s\d{3}-\d{4}$",
        ErrorMessage = "Le numéro de téléphone ne respecte pas le bon format!")]
    public string NoTelephone { get; set; }
}
```

Format à respecter: (999) 999-9999

Tel

Le numéro de téléphone ne respecte pas le bon format!

Implémenter la validation des données en MVC

Attributs personnalisés (custom attributes):

Pour les scénarios non gérés par les attributs de validation prédéfinis, vous pouvez créer des attributs de validation personnalisés. Créez une classe qui hérite de [ValidationAttribute](#) et substituez la méthode [IsValid](#).

La méthode `IsValid` accepte un objet nommé *value*, qui est l'entrée à valider. Une surcharge accepte également un objet `ValidationContext`, qui fournit des informations supplémentaires telles que l'instance de modèle créée par la liaison de modèle.

L'exemple suivant vérifie qu'un code d'employé est valide lorsqu'il est dans l'intervalle 1000 à 2000 ou 3000 à 4000. L'attribut [CustomRangeEmployeeAttribute](#) s'exécute uniquement sur le serveur.

Implémenter la validation des données en MVC

Exemple:

```
public class Model5
{
    [Required(ErrorMessage = "Code employé absent !")]
    [CustomRangeEmployee(ErrorMessage = "Code employé
    invalide ou hors intervalle !")]
    public int CodeEmploye { get; set; }
}
```

Un code d'employé est valide lorsqu'il est dans l'intervalle 1000 à 2000 ou 3000 à 4000

CodeEmploye

Code employé invalide ou hors intervalle !

Validation

Ajouter du code pour l'attribut personnalisé (custom attribute)

```
internal class CustomRangeEmployeeAttribute : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        int code = Convert.ToInt16(value);
        return ((code >= 1000 && code <= 2000) || (code >= 3000 && code <= 4000));
    }
}
```

Exemple d'un modèle avec contraintes de validité

- On ajoute des contraintes à la classe étudiant

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Web;
6
7  namespace exemple02MVC.Models
8  {
9      22 références
10     public class Etudiant
11     {
12         [Required(ErrorMessage = "Le paramètre Id est requis")]
13         13 références
14         public int Id { get; set; }
15         [Required(ErrorMessage = "Le paramètre Nom est requis")]
16         [MaxLength(4, ErrorMessage = "Le paramètre Nom ne peut avoir plus de 4 caractères")]
17         14 références
18         public string Nom { get; set; }
19         [Required(ErrorMessage = "Le paramètre Age est requis")]
20         [Range(16, 65, ErrorMessage = "Information incorrecte")]
21         14 références
22         public int Age { get; set; }
23
24         //Constructeur par défaut
25         4 références
26         public Etudiant() { }
27         //Constructeur
28         2 références
29         public Etudiant(int Id, string Nom, int Age) { this.Id = Id; this.Nom = Nom; this.Age = Age; }
30     }
31 }
```

Exemple d'un modèle avec contraintes de validité

- Création des actions pour l'ajout d'un étudiant.

```
// L'action permettant de lancer la vue d'ajout d'un étudiant
0 références
public ActionResult Create()
{
    return View();
}

// L'action de la vue d'ajout d'un étudiant pour la requête post
[HttpPost]
0 références
public ActionResult Create(Etudiant e)
{
    return View(e);
}
```

- Création d'une vue pour l'ajout d'un étudiant.

```
1  @model exemple02MVC.Models.Etudiant
2  @{
3      ViewBag.Title = "Create";
4  }
5  <h2>Ajout d'un étudiant</h2>
6
7  @using (Html.BeginForm())
8  {
9      @Html.AntiForgeryToken()
10
11     <div class="form-horizontal">
12
13         <hr />
14         @Html.ValidationSummary(true, "", new { @class = "text-danger" })
15         <div class="form-group">
16             @Html.LabelFor(model => model.Nom, htmlAttributes: new { @class = "control-label col-md-2" })
17             <div class="col-md-10">
18                 @Html.EditorFor(model => model.Nom, new { htmlAttributes = new { @class = "form-control" } })
19                 @Html.ValidationMessageFor(model => model.Nom, "", new { @class = "text-danger" })
20             </div>
21         </div>
22         <div class="form-group">
23             @Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })
24             <div class="col-md-10">
25                 @Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })
26                 @Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger" })
27             </div>
28         </div>
29         <div class="form-group">
30             <div class="col-md-offset-2 col-md-10">
31                 <input type="submit" value="Create" class="btn btn-default" />
32             </div>
33         </div>
34     </div>
35 }
```

Exemple d'un modèle avec contraintes de validité

➤ L'affichage sur un navigateur.

Browser window: Create - Mon application ASP.NET

Address bar: localhost:61269/Etudiant/Create

Header: Nom de l'application

Ajout d'un étudiant

Nom

Age

Ajouter

© 2019 - My ASP.NET Application



Browser window: Create - Mon application ASP.NET

Address bar: localhost:61269/Etudiant/Create

Header: Nom de l'application

Ajout d'un étudiant

Nom

Le paramètre Nom est requis

Age

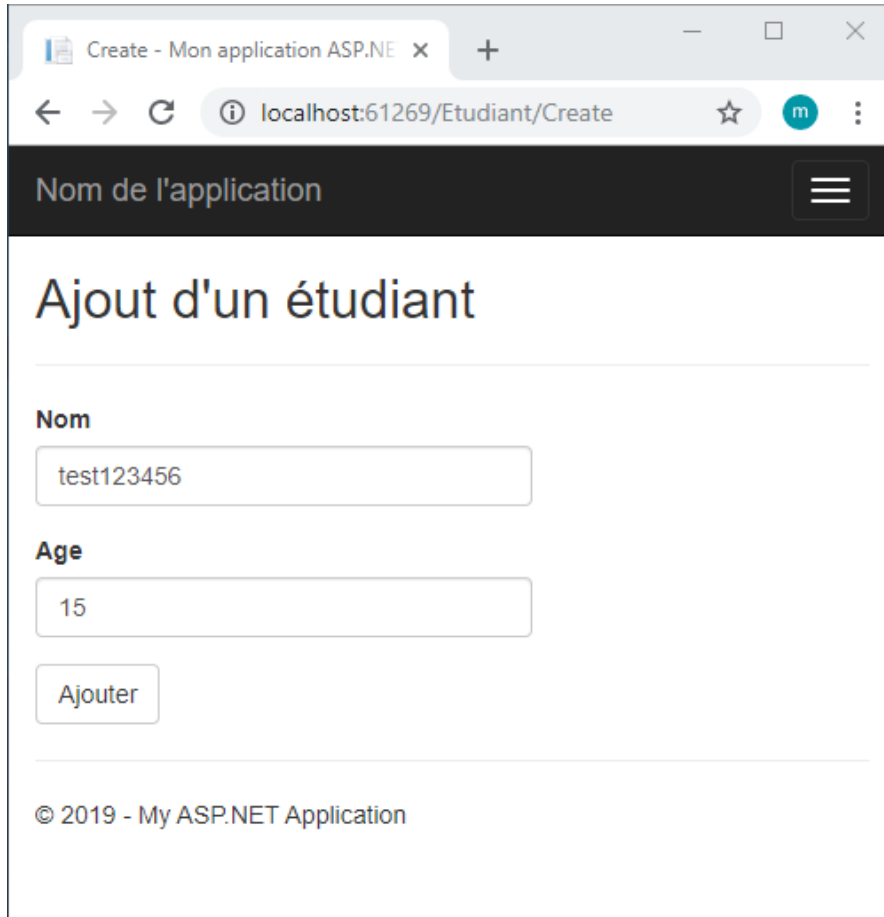
Le paramètre Age est requis

Ajouter

© 2019 - My ASP.NET Application

Exemple d'un modèle avec contraintes de validité

➤ L'affichage sur un navigateur.



Browser window: Create - Mon application ASP.NET

URL: localhost:61269/Etudiant/Create

Nom de l'application

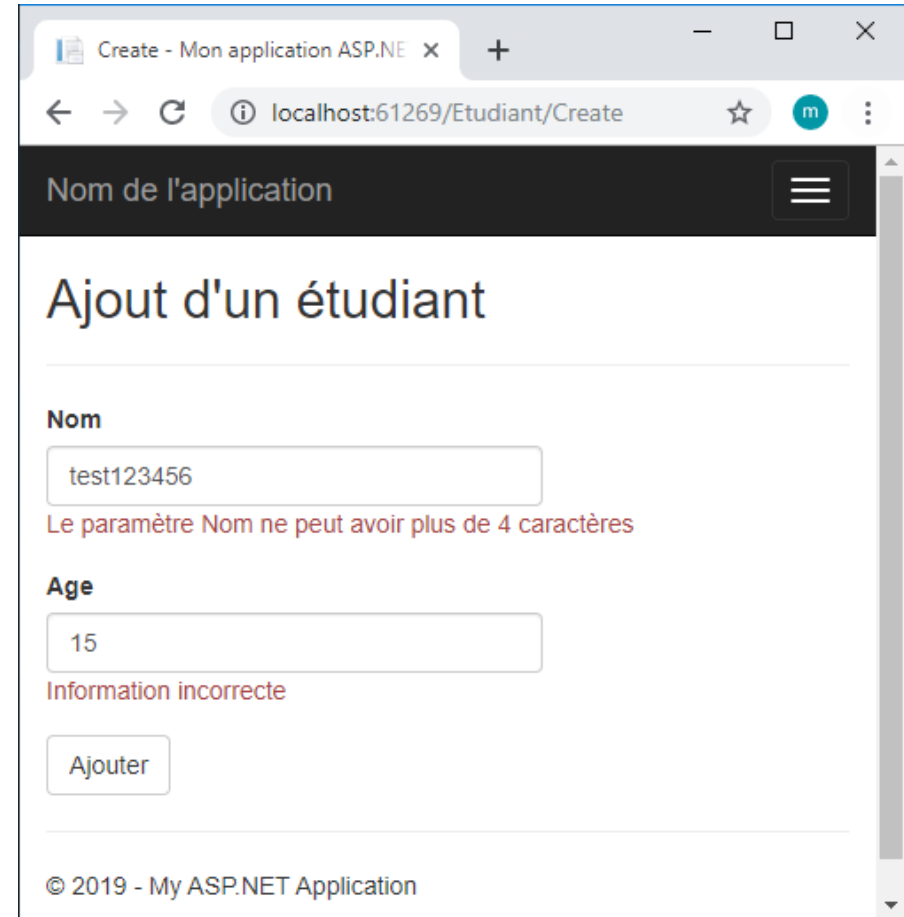
Ajout d'un étudiant

Nom

Age

Ajouter

© 2019 - My ASP.NET Application



Browser window: Create - Mon application ASP.NET

URL: localhost:61269/Etudiant/Create

Nom de l'application

Ajout d'un étudiant

Nom

Le paramètre Nom ne peut avoir plus de 4 caractères

Age

Information incorrecte

Ajouter

© 2019 - My ASP.NET Application