TP 3 : Les contrôles de validation

Objectif : Utiliser les contrôles de validation

La validation des données :

Avec la sécurité, la validation des données est en général la chose la plus importante dans un site web. Ici, nous allons pouvoir travailler côté client et côté serveur, c'est indispensable pour prévenir au plus tôt l'utilisateur d'une erreur éventuelle. En effet, il est inutile d'envoyer une demande au serveur si l'information transmise est erronée : cela génère une perte de temps et un encombrement inutile du serveur. La validation côté client est donc celle qui intervient la première et se fait en général en JavaScript. ASP.NET fournit des contrôles de validation qui génèrent le code javascript associé, vous évitant de connaître à fond le langage et de devoir taper le code.

Les contrôles de validation ASP.NET

Les contrôles de validation ASP.NET génère une validation côté client et une validation côté serveur. En effet, ASP.NET génère automatiquement du code JavaScript qu'il insère dans la page HTML renvoyée au client. Ce code sera exécuté si le navigateur du client supporte JavaScript.
Vous pouvez décider d'empêcher la validation côté client en positionnant la propriété **EnableClientScript** à false.

La syntaxe partagée des contrôles de validation des entrées est la suivante :

```
<asp:type_de_validateur id="id_validateur" runat="server"
ErrorMessage="message erreur pour synthèse"
ControlToValidate="txtPeriod"
Display ="static/dynamic/none"
Text ="Texte a afficher"
</asp:type_de_validateur >
```

La propriété **Text** correspond au texte de remplacement affiché à l'emplacement du contrôle de validation, lorsque la propriété **ErrorMessage** et la propriété Text sont utilisées lors du déclenchement du contrôle de validation. Lorsqu'un contrôle *ValidationSummary* est utilisé pour regrouper les messages d'erreur, un astérisque (*) rouge est affiché.

La propriété **Display** définit l'espacement des messages d'erreur de plusieurs contrôles de validation

- **Static** (par défaut) : la présentation est fixe pour la page
- **Dynamic** empêche l'affichage d'espaces lorsque le contrôle n'a pas déclenché d'erreurs
- **None** : pas d'affichage

La propriété **ControlToValidate** permet de nommer le nom du contrôle à valider.

La propriété **EnableClientScript** définit si la validation côté client doit être effectuée (true par défaut)

La propriété **IsValid** peut être testée pour savoir si le contrôle a été validé avec succès.

Principaux contrôles de validation :

Les principaux contrôles de validation sont

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- CustomValidator

Voyons un peu les caractéristiques générales de chacun.

- **RequiredFieldValidator**

Il permet de forcer l'utilisateur à taper une entrée dans un contrôle d'entrée. Il accepte n'importe quel caractère comme réponse valide, mais refuse l'espace et le champ vide.
Il peut posséder une valeur initiale (utile si le contrôle qu'il gère possède une valeur par défaut).
Pour lier ce contrôle à un champ, il faut remplir la propriété *ControlToValidate* avec l'*ID* du champ à vérifier. La propriété *Text* correspond au message à afficher si le contrôle ne passe pas la validation. Attention : le texte va s'afficher à l'endroit où se trouve le contrôle de validation et non pas après le contrôle à valider. Pour qu'il s'affiche juste après il faut donc mettre le contrôle de validation juste après le contrôle qu'il valide.

- **RangeValidator**

Il permet de vérifier l'entrée dans une plage de valeurs ; Il accepte une entrée vide.

- La propriété **MinimumValue** spécifie la valeur minimale
- La propriété **MaximumValue** spécifie la valeur maximale
- La propriété **Type** définit le type de données (les valeurs à comparer sont converties dans ce type de données avant la comparaison)

- **CompareValidator**

Il permet de comparer une entrée à une valeur spécifique, ou à un deuxième contrôle d'entrée. Il accepte une entrée vide.

- La propriété **ValueToCompare** sert de valeur de comparaison (Il permet de séparer plusieurs valeurs)
- La propriété **ControlToCompare** identifie un 2eme contrôle de comparaison

(qui prime sur ValueToCompare dans le cas ou les 2 sont définis).
- La propriété **Type** définit le type de données
- La propriété **Operator** indique l'opérateur de comparaison à utiliser.

➢ **Le contrôle RegularExpressionValidator**

Pour comparer si l'entrée utilisateur correspond à un modèle à un modèle défini par une expression régulière (numéro de téléphone, code postal …), on utilisera un contrôle **RegularExpressionValidator**.
La propriété **ValidationExpression** permet de rentrer une chaîne de caractères, conforme au modèle choisi : Visual Studio .Net propose un ensemble de modèles d'expressions régulières prédéfinies ; pour créer un modèle, sélectionner le modèle **Personnalisé** : le dernier modèle utilisé permet de créer son propre modèle.

➢ **CustomValidator**

Si les validations proposées par les classes de contrôle précédentes sont insuffisantes, il est possible de fournir sa propre logique de validation :
- La propriété **ClientValidationFonction** spécifie le script que doit exécuter le contrôle sur le client.
- **OnServerValidate** spécifie le script que doit exécuter le contrôle sur le server.
- La propriété **ValidateEmptyText** est nouvelle dans le .NET Framework 2.0. Elle obtient ou définit une valeur booléenne indiquant si le texte vide doit être validé.

➢ **Le contrôle ValidatonSummary**

Le contrôle *ValidationSummary* est un contrôle qui permet de regrouper toutes le texte des erreurs d'un formulaire à un seul endroit et de mettre juste un * devant les champs qui posent problème. Il va vous permettre d'afficher un sommaire, un résumé, de tous ce qui n'a pas été validé. Le texte qui y sera affiché sera celui de la propriété *ErrorMessage*. Il faut savoir que si on ne spécifie pas la propriété *Text*, elle prendra la valeur de la propriété *ErrorMessage*. En revanche si on ne défini que la propriété *Text*, la propriété *ErrorMessage* sera vide et ne prendra pas *Text* comme message par défaut. Ce contrôle de validation va afficher la propriété *ErrorMessage* de chaque contrôle de validation qui ne sera pas validé (du moins du moment qu'il a une définition de la propriété *ErrorMessage* comme nous allons le voir dans l'atelier).

La propriété **HeaderText** spécifie l'en tête du message affiché
- La propriété **DisplayMode** (List/ BulletList/SingleParagraph) spécifie le mode d'affichage
- La propriété **ShowMessageBox** permet d'afficher ce contrôle dans une MessageBox
- La propriété **ShowSummary** affichera les erreurs sur la page Web

La validation d'une page

La plateforme .Net permet de vérifier si tous les contrôles d'une page sont valides avant d'exécuter une opération, côté client à l'aide du contrôle **ValidationSummary**, côté serveur par la propriété **Page.IsValid**

La propriété **IsValid** vérifie que tous les contrôles de validation présents sur la page sont valides : on peut l'utiliser, par exemple dans le gestionnaire d'évènement d'un bouton ; en cas d'erreur, la page est automatiquement retournée au navigateur.

La propriété CausesValidation

La propriété CausesValidation va permettre de définir si les validations seront effectuées pour ce contrôle lors d'un PostBack. En effet si dans votre formulaire vous avez un bouton Valider et un bouton Quitter, vous ne pourrez pas utiliser le bouton Quitter avant que tous les champs soient correctement remplis.

La proproété ValidationGroup

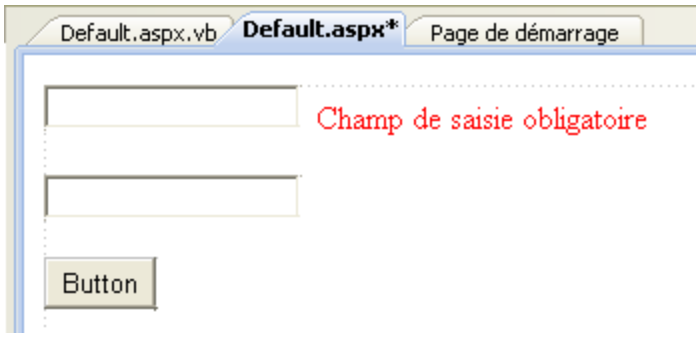
ValidationGroup est une propriété que possèdent les contrôles de validation ainsi que le contrôle Button. Il va vous permettre de créer un groupe de validation. C'est-à-dire que lorsque vous allez appuyer sur le bouton, seul le groupe spécifié dans ValidationGroup du bouton sera validé/ vérifié. Pour créer le groupe de validation il va falloir ajouter cette propriété à chacun des contrôles que vous souhaitez mettre dans le groupe. Comme il a été dit, le bouton doit posséder cette propriété lui aussi. Cela peut vous permettre par exemple de ne valider que ce que vous souhaitez suivant le groupe auquel appartient un utilisateur.

Lab 1 : Utiliser le contrôle RequiredFieldValidator

1. Lancez **Visual Studio**.
2. Créez un nouveau projet de type Web.
3. Ajoutez une zone de texte (TextBox1), une zone de texte (Textbox2), un bouton (Button1) et un RequiredFieldValidator (RequiredFieldValidator1)

4. Modifiez les deux propriétés suivantes de RequiredFieldValidator1 :

   ControlToValidate -> TextBox1
   ErrorMessage -> Champ de saisie obligatoire



5. Compilez et essayez de valider le formulaire sans avoir saisi de valeur dans la première zone de texte.

6. Notez que le contrôle s'est effectué au niveau du client et qu'aucun post-back vers le serveur n'a eu lieu.

7. Observez le code source de la page générée.

8. Modifiez la propriété suivante :

   EnableClientScript -> False

9. Compilez et essayez de valider le formulaire sans avoir saisi de valeur dans la première zone de texte : le comportement est exactement identique excepté que le contrôle cette fois-ci, s'effectue que côté serveur et non côté client.

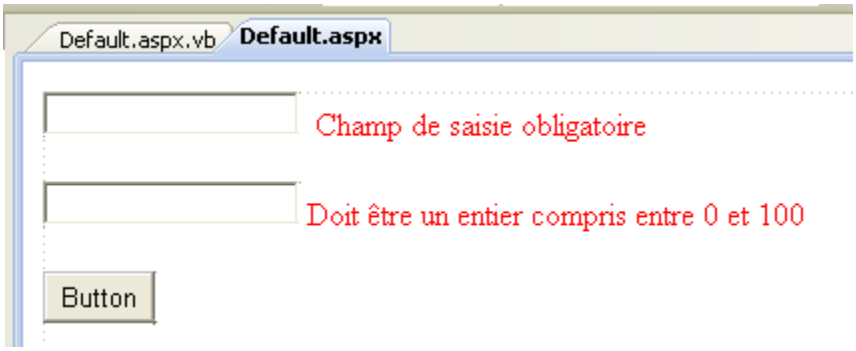
10. Observez le code source de la page générée.

Lab 2 : Utiliser le contrôle RangeValidator

1. Ajoutez un RangeValidator1
2. Modifiez les deux propriétés suivantes de RangeValidator1 :

   ControlToValidate -> TextBox2

ErrorMessage -> Doit être un entier compris entre 0 et 100
MinimumValue -> 0
MaximumValue -> 100
Type -> Integer



3. Compilez et essayez de valider le formulaire (on suppose que la première zone de texte contient une valeur quelconque) en saisissant la valeur 150 dans la seconde zone de texte puis réessayez avec la valeur 80.

Lab 3 : Utiliser le contrôle CompareValidator

1. Lancer **Visual Studio**.
2. Créez un nouveau projet de type Web.
3. Ajoutez deux zones de texte (TextBox1 et TextBox2), un bouton (Button1) et un CompareValidator1.
4. Modifiez les deux propriétés suivantes de CompareValidator1 :

   ControlToValidate -> TextBox1
   ControlToCompare -> TextBox2
   ErrorMessage -> Champs non identiques
   Operator->Equal

.

5. Compilez et essayez de valider le formulaire en saisissant des chaines différentes, ensuite des chaines identiques.

NB : Remarquez que vous pouvez :

- Utiliser d'autres types d'opérateur dans la propriété *Operator*.
- Comparer une zone de texte avec une valeur prédéfinie en utilisant la propriété *ValueToCompare*.

Lab 4 : Utiliser le contrôle CustomValidator

1. Lancez **Visual Studio**.
2. Créez un nouveau projet de type Web.
3. Ajoutez une zone de texte, un bouton (Button1) et un CustomValidator1.
4. Modifiez les deux propriétés suivantes de CustomValidator1 :

   ControlToValidate -> TextBox1

ErrorMessage -> Doit être paire
OnServerValidate= "FonctionValidation"

Dans le code behind (code du formulaire), ajoutez le code suivant afin de tester la validité de la valeur saisie :

```
public void FonctionValidation(...)
{
int val = int.parse(args.value);
if (val % 2 = 0 )
args.IsValid = false;
else
args.IsValid = true;
}
```

Recompilez et testez le fonctionnement du formulaire

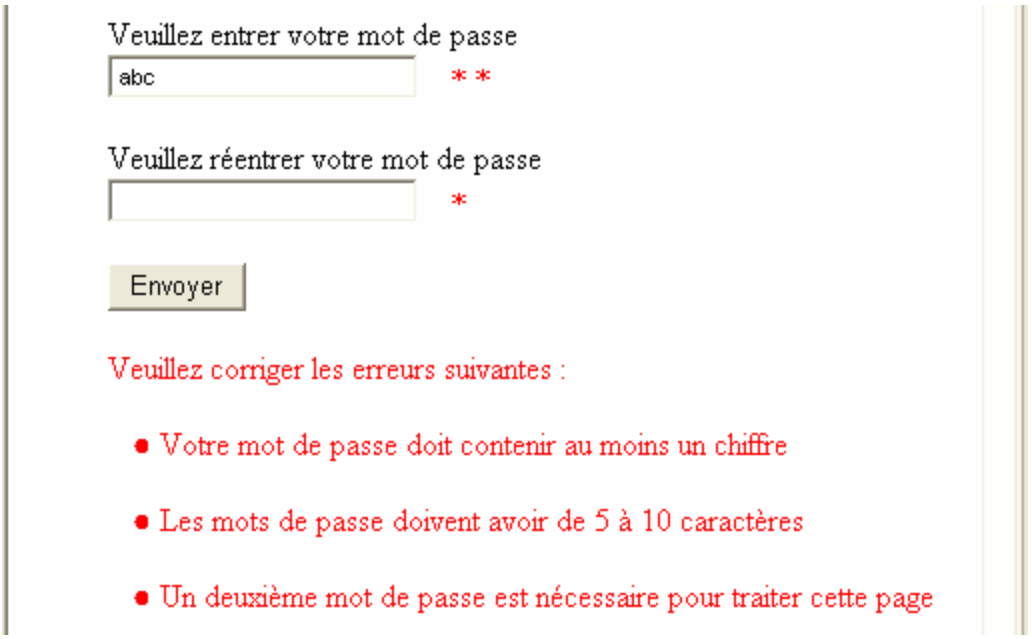
Observez le code source de la page. Vous remarquez l'ajout d'un code JavaScript.

Exercises:

Exercice 1

Exercice 2

Exercice 3

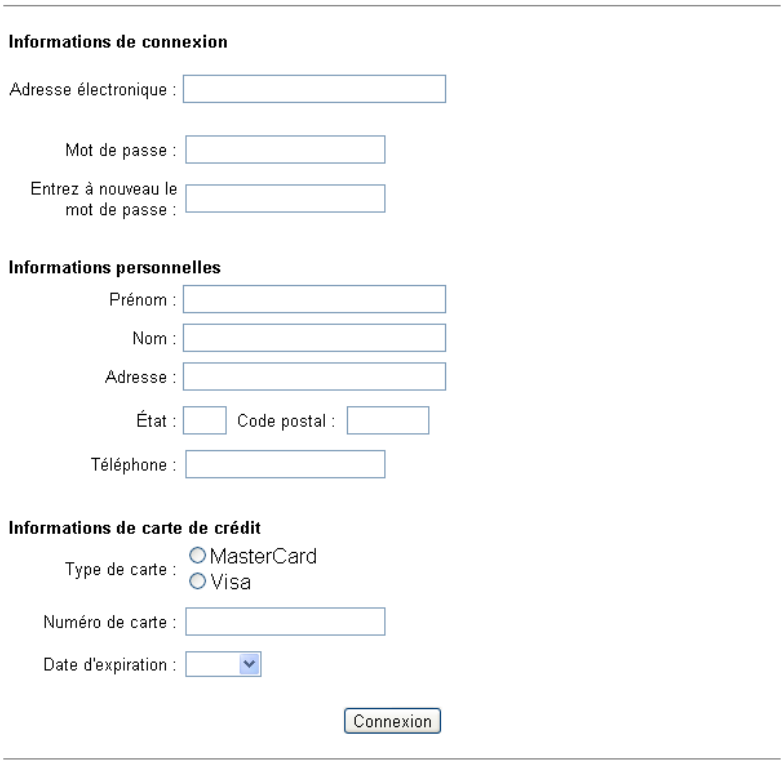


Note : Utiliser le contrôle ValidationSummary

Exercice 4

Cet exercice illustre un formulaire d'inscription standard utilisant des variantes de contrôles de validation décrits dans les annexes.

Exemple de validation de formulaire de connexion

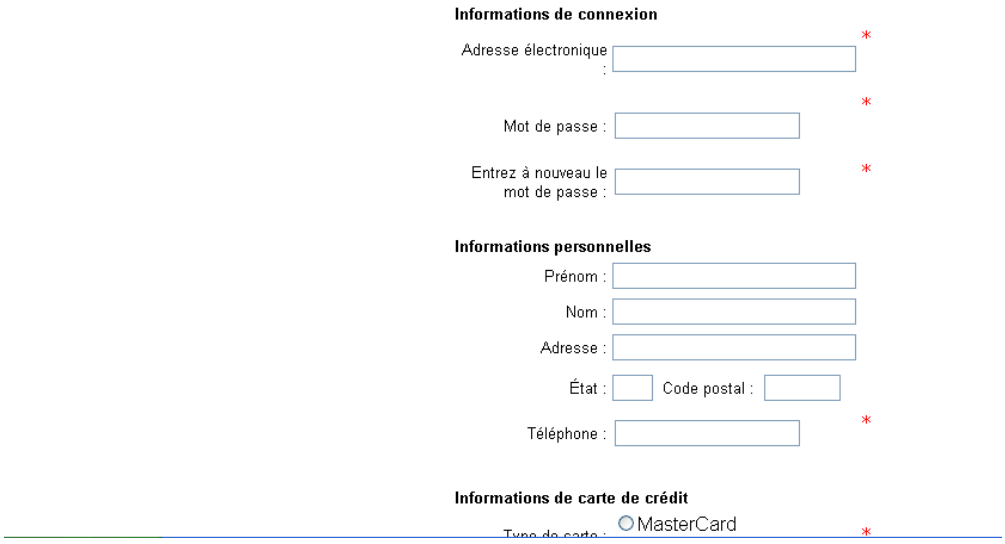


- La date d'expiration doit être supérieure à la date courante

Si des champs sont vides :

Vous devez entrer une valeur valide dans les champs ci-dessous :

- Adresse électronique.
- Mot de passe.
- Entrez à nouveau le mot de passe.
- Téléphone.
- Type de carte.
- Numéro de carte.
- Date d'expiration.



Si les contraintes ne sont pas vérifiées