Module 5: Développement de vues ASP.NET MVC 5

# **Contenu:**

|  |  |
| --- | --- |
|  | [Aperçu du module](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P1) |
| **Leçon 1:** | [Création de vues avec la syntaxe Razor](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P2) |
| **Leçon 2:** | [Utilisation des HTML Helpers](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P3) |
| **Lecon 3:** | [Réutiliser le code dans les vues](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P4) |
| **Laboratoire:** | [Développement de vues ASP.NET MVC 5](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P5) |
|  | [Examen du module et points à retenir](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C05.html#P6) |

# **Aperçu du module**

Les vues sont l'un des trois principaux composants du modèle de programmation MVC. Vous pouvez créer l'interface utilisateur de votre application Web en écrivant des vues. Une vue est un mélange de balisage HTML et de code qui s'exécute sur le serveur Web. Vous devez savoir comment écrire le HTML et le code trouvés dans une vue et utiliser les différentes classes d'assistance intégrées à MVC. Vous devez également savoir comment écrire des vues partielles, qui rendent des sections de HTML pouvant être réutilisées à de nombreux endroits de votre application Web. La connaissance de la création de vues vous aidera à développer des pages Web présentant un contenu dynamique.

### **Objectifs**

Après avoir terminé ce module, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Créez une vue MVC et ajoutez-y un balisage Razor pour afficher les données aux utilisateurs. |
| • | Utilisez des aides HTML dans une vue pour afficher des contrôles, des liens et d'autres éléments HTML. |
| • | Réutilisez le balisage Razor à plusieurs emplacements dans une application. |

# Leçon 1: Création de vues avec la syntaxe Razor

Lorsqu'un utilisateur fait une demande à une application Web MVC, un contrôleur répond. Souvent, l'action du contrôleur instancie un objet modèle, par exemple, un produit. Le contrôleur peut obtenir le produit en interrogeant la base de données, en créant un nouveau produit ou en effectuant une autre étape. Pour afficher le produit, le contrôleur crée une vue et lui transmet le produit. La vue crée une page Web en insérant des propriétés du produit, et parfois, d'autres sources, dans le balisage HTML. MVC envoie la page Web terminée au navigateur. Pour créer des vues, vous devez comprendre comment MVC interprète le code que vous placez dans les vues et comment une page HTML terminée est construite. Par défaut, le moteur de vue Razor effectue cette interprétation, mais vous pouvez également utiliser d'autres moteurs de vue. Si vous prenez le temps de comprendre Razor, vous trouverez que c'est un excellent moteur de visualisation polyvalent qui donne un grand degré de contrôle sur le HTML rendu. D'autres moteurs d'affichage ont des syntaxes très différentes, que vous préférerez peut-être si vous avez travaillé avec des versions antérieures d'ASP.NET ou d'autres technologies Web côté serveur, telles que Ruby on Rails.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Ajoutez des vues à une application Web MVC 5. |
| • | Utilisez le symbole @ dans une vue Razor pour traiter le code comme du contenu. |
| • | Décrivez les fonctionnalités de la syntaxe Razor. |
| • | Liez les vues aux classes de modèle pour afficher les données. |
| • | Expliquez comment rendre du HTML accessible aux visiteurs malvoyants. |

## **Ajout de vues**

Dans une application MVC, il existe généralement un contrôleur pour chaque classe de modèle. Par exemple, une classe de modèle nommée Product a généralement un contrôleur appelé ProductController qui contient toutes les actions pertinentes pour les produits. Il se peut que certains contrôleurs ne correspondent à aucune classe de modèle, comme le HomeController. Cependant, chaque contrôleur peut avoir plusieurs vues. Par exemple, vous souhaiterez peut-être créer les vues suivantes pour les objets Product:

|  |  |
| --- | --- |
| • | Des détails. La vue détaillée peut afficher un produit, son prix, son numéro de catalogue et d'autres détails. |
| • | Créer. La vue de création peut permettre aux utilisateurs d'ajouter un nouveau produit dans le catalogue. |
| • | Éditer. La vue d'édition peut permettre aux utilisateurs de modifier les propriétés d'un produit existant. |
| • | Effacer. La vue de suppression peut permettre aux utilisateurs de supprimer un produit du catalogue. |
| • | Indice. La vue d'index peut afficher tous les objets Product du catalogue. |

Par convention, une application Web MVC crée toutes les vues dans le dossier Views de niveau supérieur. Dans ce dossier, il y a un dossier pour chaque contrôleur de l'application. Dans l'exemple précédent, le dossier Views contiendrait un dossier Product. Ce dossier contiendrait les vues Détails, Créer, Modifier, Supprimer et Indexer.

Si vous utilisez le moteur de vue Razor et le langage C #, les vues sont des fichiers avec une extension .cshtml. Si vous utilisez le moteur d'affichage Razor et le langage Visual Basic, les vues sont des fichiers avec une extension .vbhtml. D'autres moteurs de vue utilisent des extensions différentes.

**Comment créer un fichier de vue**

Vous pouvez ajouter un fichier de vue dans Visual Studio en procédant comme suit:

|  |  |
| --- | --- |
| 1. | Dans le volet Explorateur de solutions, sélectionnez le dossier dans lequel vous souhaitez créer la vue. Par exemple, pour un contrôleur nommé ProductController, MVC s'attend à ce que les fichiers de vue se trouvent dans le dossier Views / Product. |
| 2. | Cliquez avec le bouton droit sur le dossier sélectionné, pointez sur Ajouter, puis sélectionnez Afficher.  Vous pouvez également créer un fichier de vue pour une action de contrôleur particulière en ouvrant le fichier de code de contrôleur, en cliquant avec le bouton droit sur l'action, puis en cliquant sur Ajouter une vue. Si vous créez une vue à l'aide du fichier de code du contrôleur, certaines propriétés de la boîte de dialogue Ajouter une vue sont remplies par défaut. |

Le tableau suivant décrit les propriétés que vous devez remplir dans la boîte de dialogue Ajouter une vue.

| **Biens** | **La description** |
| --- | --- |
| Afficher le nom | C'est le nom de la vue. Le nom du fichier de vue est ce nom avec l'extension appropriée ajoutée. Le nom que vous choisissez doit correspondre au nom renvoyé par l'action de contrôleur correspondante. Si le contrôleur d'action ne spécifie pas le nom de la vue à utiliser, MVC suppose que le nom de la vue correspond au nom de l'action du contrôleur. |
| Classe de modèle | Si vous créez une vue fortement typée, vous devez spécifier la classe de modèle à lier à la vue. Visual Studio utilisera cette classe lorsqu'il formule des invites IntelliSense et vérifie les erreurs de compilation. |
| Modèle | Un modèle est une vue de base que Visual Studio peut utiliser pour créer la vue. Si vous spécifiez une classe de modèle pour la vue, Visual Studio peut créer des modèles simples pour les vues Créer, Modifier, Détails, Supprimer et Liste. Lorsque vous serez plus expérimenté dans l'utilisation des vues, vous préférerez peut-être créer une vue à partir du modèle d'échafaudage vide, plutôt que d'utiliser des modèles d'échafaudage remplis. |
| Bibliothèques de scripts de référence | Lorsque vous cochez cette case, des liens vers des fichiers de script courants côté client sont inclus dans la vue. Ces liens incluent la bibliothèque JavaScript jQuery. |
| Créer comme vue partielle | Une vue partielle est une section du code Razor que vous pouvez réutiliser dans plusieurs vues de votre application. |
| Utiliser une page de mise en page | Une page de mise en page peut être utilisée pour imposer une mise en page et une image de marque standard sur de nombreuses pages de l'application Web. |

**Question:**Vous utilisez le moteur de vue Razor et Visual Basic pour créer des vues. Vous cliquez avec le bouton droit sur l'action Supprimer dans le fichier CustomerController.vb. Quel est le nom du fichier de vue que Visual Studio créera par défaut?

## **Différencier le code côté serveur du HTML**

Le moteur de vue Razor interprète les fichiers de vue et exécute tout code côté serveur contenu dans les fichiers de vue. Pour ce faire, Razor doit distinguer le code côté serveur du contenu HTML qui doit être envoyé au navigateur sans modification.

Le moteur de vue Razor recherche le symbole @ pour identifier le code côté serveur.

**Utilisation de @ pour déclarer le code côté serveur**

<p> Razor n'interprétera aucun caractère de cette ligne comme du code, car il n'y a pas de symbole «at» </p>

<p>

Le titre de la photo est:

@ Model.Title

</p>

Une section de code marquée du symbole @ est appelée expression de code Razor. Dans la syntaxe Razor, vous marquez le début d'une expression de code Razor avec le symbole @. Razor n'a pas de symbole de fin d'expression. Au lieu de cela, Razor déduit la fin d'une expression de code en utilisant un moteur d'analyse assez sophistiqué.

**Modification de l'interprétation du code et du contenu**

Parfois, vous devrez peut-être modifier la logique utilisée par Razor pour interpréter les expressions de code. Par exemple, si vous souhaitez afficher un symbole @ dans le navigateur, vous utilisez @@. Razor l'interprète comme une séquence d'échappement et restitue un seul symbole @. Cette technique est utile pour le rendu des adresses e-mail.

Razor a une logique sophistiquée pour distinguer le code du contenu et souvent le symbole @ est tout ce qui est nécessaire. Cependant, vous pouvez parfois constater qu'une erreur se produit car Razor interprète mal le contenu en tant que code. Pour corriger ces erreurs, vous pouvez utiliser le délimiteur @:, qui déclare explicitement une ligne comme contenu et non comme code.

Si vous souhaitez déclarer plusieurs lignes comme contenu, utilisez la balise <text> au lieu du délimiteur @:. Razor supprime les balises <text> et </text> avant de renvoyer le contenu au navigateur.

**Utilisation du délimiteur @:**

@ (

@: Ce texte est dans un bloc de code mais le délimiteur le déclare comme contenu.

)

**Encodage HTML**

Pour des raisons de sécurité, lorsque Razor exécute un code côté serveur qui renvoie une chaîne, il encode la chaîne au format HTML avant de l'afficher dans le navigateur. Par exemple, vous pouvez avoir une classe de modèle Comment avec les propriétés suivantes:

|  |  |
| --- | --- |
| • | Matière. Cette propriété contient un sujet pour le commentaire sous forme de chaîne simple sans informations de mise en forme. |
| • | HtmlComment. Cette propriété contient le texte du commentaire lui-même. Plus important encore, il inclut les informations de mise en forme sous forme de balises HTML. Par exemple, il peut inclure des balises <i> pour mettre un mot en italique. |

Si vous avez utilisé la syntaxe normale @ Model.HtmlComment pour renvoyer le texte du commentaire formaté, Razor encode «<» comme «& lt» et «>» comme «& gt» dans la source HTML. Cela fait que l'utilisateur voit «<i>» sur la page Web. Ce n'est pas le comportement que vous envisagez car vous souhaitez que les utilisateurs voient le texte en italique.

Pour rendre du texte sans encodage HTML, utilisez l'assistant Html.Raw (). Par exemple, utilisez @ Html.Raw (Model.HtmlComment).

**Noter:**Razor encode le texte au format HTML pour une raison de sécurité spécifique: pour empêcher les utilisateurs malveillants d'injecter du code malveillant dans votre application Web. Si de telles attaques réussissent, le code injecté peut contourner les restrictions de sécurité de votre application Web et refuser le service aux utilisateurs, ou accéder à des informations sensibles. Par conséquent, vous ne devez utiliser l'assistant Html.Raw () que lorsque cela est absolument nécessaire. De plus, si vous devez utiliser l'assistant Html.Raw (), vous devez vous assurer qu'il ne fonctionne pas sur des chaînes non cochées entrées par les utilisateurs.

## **Caractéristiques de la syntaxe Razor**

Razor inclut de nombreuses fonctionnalités utiles que vous pouvez utiliser pour contrôler la manière dont ASP.NET MVC restitue votre vue au format HTML. Ces fonctionnalités comprennent les suivantes:

**Commentaires de rasoir**

Vous souhaiterez peut-être inclure des commentaires dans votre code Razor pour le décrire aux autres développeurs de votre équipe. Il s'agit d'une technique importante qui améliore la productivité des développeurs en facilitant la compréhension du code.

**Un commentaire de rasoir**

@ \* Ce texte ne sera pas rendu par le moteur de visualisation Razor, car il s'agit d'un commentaire. \* @

**Expressions de code implicites et parenthèses**

Razor utilise des expressions de code implicites pour déterminer les parties d'une ligne qui sont du code côté serveur. Habituellement, les expressions de code implicites affichent le code HTML souhaité, mais il se peut que vous constatiez parfois que Razor interprète une expression comme HTML alors qu'elle doit être exécutée en tant que code côté serveur.

**Une expression de code implicite**

<span> Prix TTC: @ Model.Price \* 1.2 </span>

Vous pouvez contrôler et modifier ce comportement en utilisant des parenthèses pour encadrer l'expression que Razor doit évaluer.

**Utilisation de parenthèses pour délimiter explicitement des expressions**

<span> Prix TTC: @ (Model.Price \* 1.2) </span>

**Blocs, boucles et conditions de code Razor**

**Un bloc de code de rasoir**

@ {

// Razor interprète tout le texte entre ces accolades comme du code côté serveur.

}

Razor inclut également des blocs de code qui exécutent des instructions conditionnelles ou parcourent des collections.

**Un rasoir si un bloc de code**

@if (Model.InStock)

{

Html.ActionLink ("Acheter ce produit", "AddToCart")

}

Les boucles Razor sont utiles pour créer des vues d'index, qui affichent de nombreux objets d'une classe de modèle particulière. Une page de catalogue de produits, qui affiche de nombreux produits de la base de données, est un bon exemple de vue d'index. Pour implémenter une vue d'index, une action de contrôleur transmet une collection énumérable d'objets à la vue.

**Un bloc de code Razor Foreach**

@foreach (élément var dans le modèle)

{

<div> @ item.ProductName </div>

}

**Question:**Vous souhaitez décrire un bloc de code aux développeurs dans votre fichier de vue. Vous ne souhaitez pas que votre description soit transmise au navigateur. Quelle syntaxe devez-vous utiliser?

## **Liaison des vues aux classes de modèle et affichage des propriétés**

De nombreuses vues Razor sont conçues pour afficher les propriétés d'une classe de modèle spécifique. Si vous liez ces vues à la classe qu'elles affichent, vous obtenez une aide supplémentaire telle que des commentaires IntelliSense lorsque vous écrivez le code Razor. D'autres vues peuvent afficher des propriétés de différentes classes de modèle dans différents cas, ou peuvent ne pas utiliser du tout une classe de modèle. Ces vues ne peuvent pas être liées à une classe de modèle et sont appelées vues dynamiques. Vous devez comprendre comment écrire du code Razor pour ces deux situations.

**Vues fortement typées**

Dans de nombreux cas, lorsque vous créez une vue, vous savez que l'action du contrôleur transmettra toujours un objet d'une classe de modèle spécifique. Par exemple, si vous écrivez la vue Afficher pour l'action Afficher dans le contrôleur de produit, vous savez à partir du code d'action que l'action passera toujours un objet Produit à la vue.

Dans de tels cas, vous pouvez créer une vue fortement typée. Une vue fortement typée est une vue qui inclut une déclaration de la classe de modèle. Lorsque vous déclarez la classe de modèle dans la vue, Visual Studio vous aide avec des commentaires IntelliSense supplémentaires et la vérification des erreurs lorsque vous écrivez le code. Cela est possible car Visual Studio peut vérifier les propriétés incluses dans la classe de modèle. Le dépannage des erreurs d'exécution est également facile. Vous devez créer des vues fortement typées chaque fois que vous le pouvez afin de bénéficier de cet IntelliSense supplémentaire et de la vérification des erreurs. Si vous suivez cette pratique, vous ferez moins d'erreurs de codage. Une vue fortement typée fonctionne uniquement avec un objet de la classe de modèle dans la déclaration.

Pour créer une vue fortement typée dans Visual Studio, dans la boîte de dialogue Ajouter une vue, choisissez une classe de modèle à lier à la vue.

**Le modèle de déclaration**

@model MyWebSite.Models.Product

Plus loin dans le fichier de vue, vous pouvez accéder aux propriétés de l'objet de modèle à l'aide de l'assistant de modèle. Par exemple, vous pouvez accéder à un numéro de catalogue de produit de cette manière: @ Model.CatalogNumber.

**Liaison à une liste énumérable**

Certaines vues affichent de nombreuses instances d'une classe de modèle. Par exemple, une page de catalogue de produits affiche de nombreuses instances de la classe de modèle de produit. Dans de tels cas, l'action du contrôleur transmet une liste d'objets de modèle à la vue, au lieu d'un seul objet de modèle. Vous pouvez toujours créer une vue fortement typée dans cette situation, mais vous devez utiliser une déclaration de modèle différente. Vous parcourez généralement les éléments de la liste en utilisant une boucle Razor foreach.

**Une vue d'une liste d'objets**

@model IEnumerable <MyWebSite.Models.Product>

<h1> Catalogue de produits </h1>

@foreach (produit var dans le modèle)

{

<div> Nom: @ Product.Name </div>

}

**Utilisation des vues dynamiques**

Parfois, vous souhaiterez peut-être créer une vue pouvant afficher plusieurs classes de modèle. Par exemple, vous pouvez créer une vue qui peut afficher vos propres produits et produits d'un autre fournisseur, ensemble. Ces classes de modèle peuvent être nommées Product et ThirdPartyProduct. Certaines propriétés peuvent être similaires tandis que d'autres diffèrent. Par exemple, Product et ThirdPartyProduct peuvent inclure la propriété Price tandis que seule la classe de modèle ThirdPartyProduct inclut la propriété Supplier.

De plus, parfois, l'action du contrôleur ne transmet aucune classe de modèle à la vue. L'exemple le plus courant d'une telle vue sans classe est la page d'accueil du site.

Dans de tels cas, vous pouvez créer une vue dynamique. Une vue dynamique n'inclut pas la déclaration @model en haut de la page. Vous pouvez choisir d'ajouter la déclaration @model pour changer la vue en une vue fortement typée, plus tard.

Lorsque vous créez des vues dynamiques, vous recevez des commentaires IntelliSense et une vérification des erreurs moins utiles, car Visual Studio ne peut pas vérifier les propriétés de la classe de modèle pour vérifier le code. Dans de tels scénarios, vous devez vous assurer que vous n'accédez qu'aux propriétés existantes. Pour accéder à une propriété qui peut exister ou non, telle que la propriété ThirdPartyProduct.Supplier dans l'exemple précédent, vérifiez la propriété pour null avant de l'utiliser.

**Question:**Vous souhaitez écrire une vue qui affiche dix objets de la classe de modèle Photo. Quel modèle de déclaration devez-vous utiliser?

## **Rendu du HTML accessible**

Internet est pour tout le monde, indépendamment des handicaps qu'un utilisateur individuel peut avoir. En outre, si les utilisateurs handicapés ne peuvent pas facilement naviguer sur votre site Web, ils peuvent visiter les sites Web de vos concurrents et votre entreprise peut perdre des affaires. Vous devez donc vous assurer que les personnes handicapées, telles que les personnes malvoyantes ou malentendantes, peuvent utiliser votre application Web et que leurs navigateurs Web peuvent analyser le contenu HTML présenté sur votre site. Étant donné que les vues MVC sont responsables du rendu du HTML dans une application Web MVC, vous rendez le HTML accessible en écrivant des vues qui respectent les consignes d'accessibilité. Lorsque vous écrivez des vues, gardez à l'esprit les défis et les meilleures pratiques suivants.

Les utilisateurs ont des exigences différentes en fonction de leurs capacités et de leurs handicaps. Par exemple, considérez les facteurs suivants:

|  |  |
| --- | --- |
| • | Les utilisateurs malvoyants peuvent utiliser un navigateur standard, mais ils peuvent augmenter la taille du texte avec une loupe d'écran afin de pouvoir lire le contenu. |
| • | Les utilisateurs profondément aveugles peuvent utiliser un navigateur avec un logiciel de synthèse vocale ou du matériel de synthèse vocale en braille. |
| • | Les utilisateurs daltoniens peuvent rencontrer des difficultés si une différence de couleur est utilisée pour mettre le texte en surbrillance. |
| • | Les utilisateurs sourds peuvent ne pas être en mesure d'accéder au contenu audio. |
| • | Les utilisateurs ayant une dextérité limitée peuvent avoir du mal à cliquer sur de petites cibles. |
| • | L'utilisateur épileptique peut avoir des crises s'il est présenté avec un contenu clignotant. |

Vous pouvez vous assurer que votre contenu est accessible au plus large éventail d'utilisateurs en respectant les consignes suivantes:

|  |  |
| --- | --- |
| • | Ne vous fiez pas aux différences de couleur pour mettre en évidence le texte ou tout autre contenu. Par exemple, les liens doivent être soulignés ou mis en forme en gras pour les mettre en valeur auprès des utilisateurs daltoniens. |
| • | Fournissez toujours un contenu alternatif équivalent pour le contenu visuel et auditif. Par exemple, complétez toujours l'attribut alt pour les images. Utilisez cet attribut pour décrire l'image afin que le logiciel de synthèse vocale ou le matériel de synthèse vocale puisse rendre des mots significatifs à l'utilisateur. |
| • | Utilisez des feuilles de balisage et de style pour séparer le contenu de la structure et du code de présentation. Cela aide les logiciels d'interprétation de texte à rendre le contenu aux utilisateurs sans être dérouté par le code de structure et de présentation. Par exemple, vous devez appliquer les bonnes pratiques suivantes pour afficher du contenu sur votre page Web:   |  |  | | --- | --- | | o | Évitez d'utiliser des tableaux pour afficher le contenu. Vous ne devez utiliser des tableaux que pour présenter le contenu tabulé. Les tableaux peuvent être utilisés pour rendre des graphiques et une image de marque sur une page Web, mais dans un site accessible, utilisez des feuilles de style positionnelles pour afficher le contenu. Les lecteurs de texte ne lisent pas les feuilles de style. | | o | Évitez d'utiliser des tableaux imbriqués. Dans un tableau imbriqué, une cellule de tableau contient un autre tableau. Ceux-ci sont particulièrement déroutants pour les lecteurs de texte car ils lisent chaque cellule du tableau dans un ordre séquentiel. L'utilisateur est susceptible de devenir désorienté et incapable de déterminer quelle cellule est en cours de lecture et ce que cela signifie. | | o | Évitez d'utiliser des images contenant du texte important. Les lecteurs de texte ne peuvent pas restituer du texte à partir d'un fichier image. Utilisez plutôt un balisage pour rendre ce texte. | |

**Lecture supplémentaire:**Le World Wide Web Consortium (W3C) a un projet appelé la Web Accessibility Initiative (WAI) qui promeut le contenu Web accessible. Ce projet a publié les Directives pour l'accessibilité du contenu Web (WCAG). Ces directives sont acceptées par l'industrie de l'édition Web comme définitives. Pour lire les directives complètes, accédez à:<http://go.microsoft.com/fwlink/?LinkID=288955&clcid=0x409>

**Noter:**Dans ASP.NET MVC, le développeur a un contrôle complet sur le code HTML que le serveur Web envoie au navigateur Web. Cependant, vous devez comprendre les principes d'accessibilité pour écrire des vues MVC accessibles. Assurez-vous que toute votre équipe de développement connaît les exigences d'accessibilité et les meilleures pratiques associées.

**Question:**Vous souhaitez présenter le logo de votre entreprise dans la partie supérieure de la page. Le logo est un fichier .gif qui comprend le nom de l'entreprise dans une police inhabituelle. Comment pouvez-vous vous assurer que le logo est accessible?

# Leçon 2: Utilisation des HTML Helpers

Le Framework ASP.NET MVC 5 comprend de nombreuses fonctions d'assistance HTML que vous pouvez utiliser dans les vues. Les aides HTML sont des méthodes C # que vous pouvez appeler pour rendre des valeurs, des étiquettes et des contrôles d'entrée tels que des zones de texte. Les Helpers incluent une logique commune qui facilite le rendu du HTML pour les propriétés de classe de modèle MVC et d'autres tâches. Certains assistants HTML examinent la définition d'une propriété dans une classe de modèle, puis rendent le code HTML approprié. Certains assistants HTML appellent les actions du contrôleur et d'autres valident les données d'entrée utilisateur. Vous pouvez également créer des helpers HTML personnalisés. Vous devez connaître l'assistant HTML approprié à utiliser dans n'importe quel scénario et comment transmettre les paramètres appropriés aux assistants HTML appropriés.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Utilisez les assistants Html.ActionLink et Url.Action pour appeler une action de contrôleur. |
| • | Utilisez les aides Html.DisplayNameFor et HtmlDisplayFor pour afficher la valeur d'une propriété de classe de modèle. |
| • | Utilisez les aides Html.LabelFor et HTML.EditorFor pour restituer un contrôle d'éditeur pour une propriété de classe de modèle. |
| • | Utilisez l'assistant Html.BeginForm pour rendre un formulaire pour une action de contrôleur. |
| • | Utilisez les aides HTML pour valider les données saisies par les utilisateurs. |
| • | Utilisez divers assistants HTML pour créer une interface utilisateur pour une application Web. |

## **Utilisation d'Actions Helpers**

Les helpers HTML sont des méthodes simples qui, dans la plupart des cas, renvoient une chaîne. Cette chaîne est une petite section de HTML que le moteur de vue peut insérer dans le fichier de vue pour générer la page Web terminée. Vous pouvez écrire des vues qui rendent n'importe quel type de contenu HTML sans utiliser un seul assistant si vous préférez ne pas utiliser d'aide HTML. Cependant, les assistants HTML facilitent la gestion du contenu HTML en rendant les chaînes HTML courantes pour les scénarios souvent utilisés.

Les assistants Html.ActionLink () et Url.Action () vous permettent de rendre du HTML qui appelle des actions de contrôleur.

**L'assistant Html.ActionLink ()**

Lorsque l'utilisateur envoie une demande à une application Web MVC, l'infrastructure ASP.NET MVC 5 transmet la demande au contrôleur et à l'action appropriés, transmettant ainsi les paramètres corrects. Il est possible pour les utilisateurs de faire de telles demandes en tapant une URL dans la barre d'adresse d'un navigateur Web, mais ils connaissent rarement le nom du contrôleur, l'action ou les paramètres. Au lieu de cela, ils cliquent généralement sur un lien. L'assistant Html.ActionLink () peut être utilisé pour rendre des liens vers des actions pour vous. L'assistant renvoie un élément <a> avec la valeur de paramètre href correcte lorsque les utilisateurs cliquent sur un lien dans une page Web.

**Utilisation de l'assistant Html.ActionLink ()**

@ Html.ActionLink ("Cliquez ici pour voir la photo 1", "Afficher", nouveau {id = 1})

Par défaut, l'exemple restitue le code HTML suivant:

<a href="/photo/display/1"> Cliquez ici pour voir la photo 1 </a>

Notez que vous transmettez des paramètres à l'assistant à l'aide d'un objet anonyme. De cette manière, vous pouvez passer plusieurs paramètres.

**Noter:**L'assistant Html.ActionLink () fonctionne avec le moteur de routage pour formuler l'URL correcte pour le lien. L'assistant rendra la valeur href ci-dessus, en supposant qu'elle soit appelée à partir d'une vue PhotoController. Il existe cependant plusieurs surcharges de l'assistant ActionLink si vous souhaitez l'utiliser pour appeler des actions dans d'autres contrôleurs.

**L'assistant Url.Action**

L'assistant Url.Action () restitue une URL à l'aide du moteur de routage sans rendre l'élément <a> englobant.

Vous pouvez utiliser l'assistant Url.Action () pour remplir l'attribut src d'un élément <img> ou d'un élément <script>. Cet assistant est utile chaque fois que vous souhaitez formuler une URL vers une action de contrôleur sans afficher un lien hypertexte.

**Rendu d'une image à partir d'une action**

<img alt = "Cette image provient d'une action" src = "@ Url.Action (" GetImage ", new {id = 1})" />

**Lecture supplémentaire:**Pour plus d'informations sur les helpers Html.ActionLink () et Url.Action (), consultez:

<http://go.microsoft.com/fwlink/?LinkID=288959&clcid=0x409>

<http://go.microsoft.com/fwlink/?LinkID=288960&clcid=0x409>

**Question:**Vous souhaitez rendre une balise HTML5 <audio> pour lire un fichier son à partir d'une action. Souhaitez-vous utiliser l’assistant Html.ActionLink () ou l’assistant Url.Action ()?

## **Utilisation des assistants d'affichage**

MVC comprend plusieurs assistants qui affichent les propriétés des classes de modèle. Vous pouvez utiliser ces aides pour créer des vues qui affichent les détails du produit, les détails des commentaires, les détails de l'utilisateur, etc. Html.DisplayNameFor () restitue le nom d'une propriété de classe de modèle. Html.DisplayFor () restitue la valeur d'une propriété de classe de modèle. Ces deux assistants examinent la définition de la propriété dans la classe de modèle, y compris les annotations d'affichage des données, pour s'assurer qu'elles rendent le code HTML le plus approprié.

**L'assistant Html.DisplayNameFor ()**

Vous utilisez l'assistant Html.DisplayNameFor () pour restituer le nom d'affichage d'une propriété de la classe de modèle. Si votre vue est fortement typée, Visual Studio vérifie si la classe de modèle contient une propriété avec le bon nom lorsque vous tapez le code. Sinon, vous devez vous assurer que vous utilisez une propriété qui existe ou vérifier qu'elle n'est pas nulle avant de l'utiliser. Vous spécifiez la propriété de la classe de modèle à l'assistant HTML Html.DisplayNameFor () à l'aide d'une expression lambda.

**Utilisation de l'assistant Html.DisplayNameFor ()**

@ Html.DisplayNameFor (modèle => model.CreatedDate)

Le texte rendu par l'assistant Html.DisplayNameFor () dépend de la classe de modèle. Si vous avez utilisé une annotation DisplayName pour donner un nom plus descriptif à une propriété, Html.DisplayNameFor () utilisera cette valeur d'annotation DisplayName; sinon, il rendra le nom de la propriété.

**L'assistant Html.DisplayFor ()**

L'assistant Html.DisplayFor () prend en compte toutes les annotations d'affichage que vous spécifiez dans la classe de modèle, puis rend la valeur de la propriété.

**Utilisation de l'assistant Html.DisplayFor ()**

Cela a été créé sur: @ Html.DisplayFor (model => model.CreatedDate)

Si vous avez utilisé l'annotation DisplayFormat dans la classe de modèle avec DataFormatString défini sur «{0: jj / MM / aa}», l'assistant Html.DisplayFor () garantit que la date est affichée au format de date courte.

**Question:**Vous voulez vous assurer qu'une vue affiche «Ce produit a été modifié pour la dernière fois le» avant la propriété ModifiedDate. Ce texte est déclaré dans la classe avec l'annotation DisplayName. Quel code écririez-vous dans la vue?

## **L'assistant de création de formulaire**

Pour accepter les entrées de l'utilisateur, vous devez fournir un formulaire sur votre page Web. Un formulaire typique se compose d'un ensemble d'étiquettes et de contrôles d'entrée. Les étiquettes indiquent à l'utilisateur la propriété pour laquelle il doit fournir une valeur. Les contrôles d'entrée permettent à l'utilisateur de saisir une valeur. Les contrôles d'entrée peuvent être des zones de texte, des cases à cocher, des boutons radio, des sélecteurs de fichiers, des listes déroulantes ou d'autres types de contrôle. Il y a généralement un bouton d'envoi et un bouton d'annulation sur les formulaires.

**Rendu des formulaires HTML**

Pour créer un formulaire en HTML, vous devez commencer par un élément <form> dans la page Web HTML. Tous les libellés et contrôles d'entrée doivent se trouver dans l'élément <form>. Dans une vue MVC, vous pouvez utiliser l'assistant Html.BeginForm () pour rendre cet élément et définir l'action du contrôleur à laquelle le formulaire envoie des données.

Vous pouvez également spécifier la méthode HTTP utilisée par le formulaire pour envoyer des données. Si le formulaire utilise la méthode POST, qui est la méthode par défaut, le navigateur envoie les valeurs du formulaire au serveur Web dans le corps du formulaire. Si le formulaire utilise la méthode GET, le navigateur envoie les valeurs du formulaire au serveur Web dans la chaîne de requête de l'URL.

Dans le HTML rendu, l'élément <form> doit être fermé avec une balise </form>. Dans les vues Razor, vous pouvez vous assurer que l'élément de formulaire est fermé avec un bloc de code @using. Razor affiche la balise </form> au niveau du crochet fermant du bloc de code.

**Utilisation de l'assistant Html.BeginForm ()**

@using (Html.BeginForm ("S'inscrire", "Client", FormMethod.Post))

{

@ \* Placez les contrôles d'entrée ici \* @

}

**Utilisation de formulaires pour publier des fichiers**

En HTML, si vous souhaitez permettre aux utilisateurs du site Web de télécharger des fichiers, vous devez utiliser un formulaire HTML avec des propriétés spéciales. Plus précisément, l'élément <form> doit inclure l'attribut enctype défini sur la valeur multipart / form-data et le formulaire doit utiliser le verbe HTTP POST.

Vous pouvez spécifier des attributs supplémentaires pour l'élément <form> en les transmettant dans un paramètre htmlAttributes en tant que nouveaux objets. C'est une bonne méthode à utiliser pour ajouter l'attribut enctype. Si le formulaire comprend des fichiers téléchargés, vous devez définir l'attribut enctype sur multipart / form-data.

**Utilisation de l'assistant Html.BeginForm ()**

@using (Html.BeginForm ("Créer", "Photo", FormMethod.Post, new {enctype = "multipart / form-data"}))

{

@ \* Placez les contrôles d'entrée ici \* @

}

**Question:**Vous avez créé un formulaire avec un contrôle de sélection de fichier qui utilise la méthode GET. Vous avez défini l'attribut enctype sur multipart / form-data, mais lorsque vous essayez d'accéder au fichier dans la méthode d'action, une exception est levée. Qu'avez-vous peut-être mal fait?

## **Utilisation des assistants de l'éditeur**

Dans les formulaires HTML, il existe de nombreux contrôles d'entrée HTML que vous pouvez utiliser pour collecter des données auprès des utilisateurs. Dans les vues Razor, les aides Html.LabelFor () et Html.EditorFor () facilitent le rendu des contrôles d'entrée les plus appropriés pour les propriétés d'une classe de modèle. Pour comprendre comment ces aides rendent le HTML, vous devez d'abord comprendre les contrôles d'entrée HTML. Certains contrôles HTML courants sont décrits dans le tableau suivant.

| **Contrôler** | **Exemple** | **La description** |
| --- | --- | --- |
| Zone de texte | <input type = "text" name = "Title"> | Zone de texte sur une seule ligne dans laquelle l'utilisateur peut saisir une chaîne. L'attribut name est utilisé pour identifier la valeur entrée dans la chaîne de requête ou pour envoyer des données de formulaire à l'aide de la méthode POST. |
| Zone de texte multiligne | <textarea name = "Description"  rows = "20" cols = "80"> | Une zone de texte multiligne dans laquelle l'utilisateur peut saisir des chaînes plus longues. |
| Case à cocher | <input type = "case à cocher"  name = "ContactMe">  Veuillez m'envoyer les détails de l'offre | Une case à cocher soumet une valeur booléenne. |

**Lecture supplémentaire:**Pour plus d'informations sur tous les éléments de formulaire HTML, accédez à:<http://go.microsoft.com/fwlink/?LinkID=288961&clcid=0x409>

**L'assistant Html.LabelFor ()**

L'assistant Html.LabelFor () est similaire à l'assistant Html.DisplayNameFor () car il restitue le nom de la propriété que vous spécifiez, en tenant compte de l'annotation DisplayName si elle est spécifiée dans la classe de modèle. Cependant, contrairement à l'assistant Html.DisplayNameFor (), l'assistant Html.LabelFor () restitue un élément <label>.

**Utilisation de l'assistant Html.LabelFor ()**

@ Html.LabelFor (modèle => model.CreatedDate)

**L'assistant Html.EditorFor ()**

L'assistant Html.EditorFor () restitue les éléments HTML <input> et d'autres contrôles de formulaire pour les propriétés d'une classe de modèle. Cet assistant rend l'élément le plus approprié pour chaque type de données de propriété. Par exemple, l'assistant Html.EditorFor () rend <input type = "text"> pour une propriété de chaîne. Si la propriété de chaîne est annotée avec [DataType (DataType.MultilineText)], l'assistant Html.EditorFor () restitue un élément <textarea> à la place. Le tableau suivant décrit le code HTML que Html.EditorFor () rend pour différentes classes de modèle.

| **Contrôler** | **Propriété de classe de modèle** | **HTMLRendu par EditorFor ()** |
| --- | --- | --- |
| Zone de texte | chaîne publique Title {get; ensemble; } | <input type = "text" name = "Title"> |
| Zone de texte multiligne | [DataType (DataType.MultilineText)]  chaîne publique Description {get; ensemble;} | <textarea name = "Description"  rows = "20" cols = "80"> |
| Case à cocher | [Display (Name = "Veuillez m'envoyer les détails de l'offre")]  public bool ContactMe {get; ensemble; } | <input type = "case à cocher"  name = "ContactMe">  Veuillez m'envoyer les nouveaux détails de l'offre. |

Si l'action transmet un objet de modèle existant à la vue, l'assistant Html.EditorFor () remplit également chaque contrôle avec les valeurs actuelles de chaque propriété.

**Utilisation de l'assistant Html.EditorFor ()**

@ Html.EditorFor (modèle => model.ContactMe)

**Question:**Vous avez une propriété dans la classe de modèle Product nommée ProductID. Vous souhaitez l'inclure dans la page HTML afin que le script côté client puisse utiliser la valeur ProductID. Cependant, vous ne souhaitez pas que la valeur soit affichée aux utilisateurs. Dans la classe de modèle, vous avez annoté la propriété ProductID avec l'attribut [HiddenInput (DisplayValue = false)]. Comment l'assistant Html.EditorFor () rendra-t-il cette propriété?

## **Utilisation des assistants de validation**

Lorsque vous demandez des informations aux utilisateurs, vous souhaitez souvent que les données qu'ils saisissent dans un certain format soient utilisées ultérieurement dans l'application Web. Par exemple, vous souhaiterez peut-être vous assurer que les utilisateurs entrent une valeur pour la propriété LastName. Vous pouvez également vous assurer que les utilisateurs saisissent une adresse e-mail valide pour la propriété EmailAddress. Vous pouvez définir ces exigences en utilisant des annotations de données de validation dans la classe de modèle. Dans les actions du contrôleur, vous pouvez vérifier la propriété ModelState.IsValid pour vérifier si un utilisateur a entré des données valides.

Lorsque les utilisateurs soumettent un formulaire, s'ils ont saisi des données non valides, la plupart des sites Web affichent des messages de validation. Ces messages sont souvent surlignés en rouge, mais d'autres couleurs ou formats peuvent être utilisés. Souvent, il y a un récapitulatif de validation en haut de la page, tel que «Veuillez saisir des informations valides pour les champs suivants: Nom, Adresse e-mail». Le formulaire peut afficher des messages de validation détaillés à côté de chaque contrôle dans lequel l'utilisateur a entré des données non valides, ou mettre en évidence les problèmes dans les données fournies par l'utilisateur avec un astérisque.

Lorsque votre vue MVC affiche un formulaire, vous pouvez facilement afficher les messages de validation à l'aide des assistants de validation. Les assistants de validation ne rendent le HTML que lorsque les utilisateurs ont saisi des données non valides.

**L'assistant Html.ValidationSummary ()**

Utilisez l'assistant Html.ValidationSummary () pour afficher un résumé de toutes les données non valides du formulaire. Cet assistant est généralement appelé en haut du formulaire. Lorsque l'utilisateur soumet des données non valides, les messages de validation sont affichés pour chaque champ non valide dans une liste à puces. Vous pouvez contrôler les messages de validation dans la classe de modèle en utilisant cette syntaxe dans l'annotation:

[Obligatoire (ErrorMessage = "Veuillez saisir votre nom de famille")]

**Utilisation de l'assistant Html.ValidationSummary ()**

@Html. ValidationSummary ()

**L'assistant Html.ValidationMessageFor ()**

Utilisez l'assistant Html.ValidationMessageFor () pour afficher les messages de validation à côté de chaque entrée du formulaire.

**Utilisation de l'aide Html.ValidationMessageFor ()**

@ Html.ValidationMessageFor (model => model.LastName)

## **Démonstration: comment utiliser les HTML Helpers**

Dans cette démonstration, vous verrez comment créer une vue et utiliser les aides HTML pour y rendre les contrôles. Vous verrez également comment les exigences de validation, qui ont été définies avec des annotations de données dans la classe de modèle, prennent effet dans une vue.

### **Étapes de démonstration**

|  |  |
| --- | --- |
| • | Vous trouverez les étapes dans la section «Leçon 2: Utilisation des HTML Helpers» sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD05_DEMO.md>. |

# Leçon 3: Réutilisation du code dans les vues

Dans une application Web, vous affichez fréquemment des blocs similaires de contenu HTML dans différentes pages Web. Par exemple, dans une application Web de commerce électronique, vous souhaiterez peut-être afficher les produits les plus populaires sur la page d'accueil, sur la page d'accueil du catalogue de produits, en haut de la page de recherche de produits et peut-être à d'autres emplacements. Pour rendre ce contenu HTML dans une application MVC, vous pouvez copier et coller du code d'une vue Razor dans d'autres. Cependant, pour modifier l'affichage des meilleurs produits plus tard dans le projet, vous devrez alors apporter des modifications identiques à de nombreux endroits. Une meilleure pratique consiste à utiliser une vue partielle. Une vue partielle restitue une section de contenu HTML qui peut être insérée dans plusieurs autres vues au moment de l'exécution. Étant donné que la vue partielle est un fichier unique qui est réutilisé à plusieurs emplacements, lorsque vous implémentez une modification dans un seul emplacement, les modifications sont mises à jour dans d'autres emplacements de l'application Web. Les vues partielles augmentent la gérabilité des applications Web MVC et facilitent une présentation cohérente du contenu dans une application Web.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez comment les vues partielles vous permettent de réutiliser le code Razor. |
| • | Créez des vues partielles dans une application Web MVC 5. |
| • | Utilisez des vues partielles dans une application Web MVC 5. |
| • | Déterminez quand utiliser des vues partielles. |

## **Création de vues partielles**

Vous pouvez utiliser des vues partielles pour afficher du HTML identique ou du HTML similaire à différents emplacements de votre application Web. Pensez à une application Web dans laquelle les utilisateurs peuvent commenter des articles. Dans la vue de l'article, vous souhaitez afficher les commentaires de cet article spécifique à la fin de la page. Vous souhaitez un affichage similaire sur la page d'accueil, mais vous souhaitez afficher les commentaires les plus populaires sur n'importe quel article. Vous pouvez créer une vue partielle de commentaires unique pour ce scénario comme suit:

|  |  |
| --- | --- |
| • | Commencez par créer une vue partielle qui affiche une collection de commentaires au format requis. |
| • | Ensuite, créez une action de contrôleur qui récupère les commentaires les plus populaires, puis les transmet à la vue partielle. |
| • | Ensuite, créez une autre action de contrôleur qui utilise l'ID d'article pour rechercher des commentaires spécifiques à cet article. L'action du contrôleur transmet ensuite cette collection de commentaires à la vue partielle. |
| • | Enfin, appelez les actions appropriées depuis la page d'accueil et la vue de l'article. |

Étant donné que la vue partielle restitue une collection de commentaires, vous pouvez la réutiliser dans diverses situations en lui passant différentes collections à partir des actions du contrôleur.

**Création et attribution de noms à des vues partielles**

Dans Visual Studio, vous pouvez créer une vue partielle à l'aide de la boîte de dialogue Ajouter une vue, de la même manière que pour créer toute autre vue. Assurez-vous de cocher la case Créer en tant que vue partielle. Étant donné que les vues partielles rendent une petite section de HTML plutôt qu'une page Web complète, Visual Studio n'ajoute pas <head>, <body> et d'autres balises à la vue partielle. Par convention, les vues partielles sont nommées préfixées par un caractère de soulignement, par exemple, \_CommentsList.cshtml. Cette convention est facultative mais est souvent utile pour distinguer les vues partielles dans l'Explorateur de solutions.

Des vues partielles sont souvent créées dans le dossier / Views / Shared de votre site. Les vues du dossier Shared sont accessibles à de nombreux contrôleurs, tandis que les vues du dossier / Views / Comment ne sont accessibles qu'à partir du CommentController.

**Vues partielles fortement typées et dynamiques**

Comme pour les autres vues, vous pouvez créer des vues partielles fortement typées si vous êtes certain que la vue partielle affichera toujours la même classe de modèle. Visual Studio peut fournir les commentaires IntelliSense les plus informatifs et la vérification des erreurs pour les vues partielles fortement typées, par rapport à d'autres types de vues. Une vue fortement typée a la déclaration de la directive @model en haut du fichier. Vous pouvez également créer une vue partielle dynamique si vous n'êtes pas certain que la vue partielle affichera toujours la même classe de modèle. Vous pouvez créer des vues partielles dynamiques en omettant la déclaration de directive @model.

**Une vue partielle fortement typée**

@model IEnumerable <Adworks.Models.Comment>

<ul>

@foreach (élément var dans le modèle) {

<li>

Sujet: @ Html.DisplayFor (modelItem => item.Subject)

</li>

}

</ul>

**Question:**Vous souhaitez créer une vue partielle qui affiche une liste de commentaires. La vue partielle des commentaires sera appelée par des actions dans le PhotoController et le HomeController. Dans quel dossier devez-vous créer le fichier de vue partielle?

## **Utilisation des vues partielles**

En plus de savoir comment créer des vues partielles, vous devez savoir comment les utiliser dans plusieurs autres vues de votre site. Vous pouvez réutiliser des vues partielles à l'aide des aides HTML Html.Partial () et Html.Action ().

**L'assistant Html.Partial ()**

Vous pouvez utiliser l'assistant Html.Partial () pour rendre une vue partielle dans un autre fichier de vue. MVC insère le HTML que la vue partielle rend dans la vue parente et renvoie la page Web complète au navigateur. Le premier paramètre est le nom du fichier de vue partielle sans son extension de fichier. Le deuxième paramètre facultatif est un objet de modèle pour le rendu de la vue partielle. Il s'agit du même objet de modèle que celui utilisé par la vue parent car les autres objets de modèle ne sont pas disponibles dans ce contexte.

**Utilisation de l'assistant Html.Partial ()**

@ Html.Partial ("\_ Commentaires")

N'oubliez pas que vous pouvez utiliser les collections ViewBag et ViewData pour transmettre des données entre la méthode d'action et la vue. Une vue parent partagera les mêmes objets ViewBag ou ViewData avec la vue partielle. C'est un bon moyen de partager des données entre l'action du contrôleur, la vue et la vue partielle.

**L'assistant Html.Action ()**

Vous pouvez passer un objet de modèle de la vue parent à une vue partielle lorsque vous utilisez Html.Partial (). Cependant, la vue parente ne peut transmettre que le même objet de modèle disponible dans la vue parente. Parfois, vous souhaiterez peut-être utiliser un objet de modèle différent de la vue parente, souvent d'une classe de modèle différente, avec la vue partielle. Dans de tels cas, vous pouvez utiliser l'assistant Html.Action (). Cela appelle une méthode d'action qui renvoie la vue partielle correcte. La méthode d'action peut créer une instance de n'importe quelle classe de modèle.

Considérez une vue Article qui affiche le texte et le titre d'un objet Article. Vous souhaitez afficher les commentaires à la fin de cette vue dans une vue partielle. Vous pouvez utiliser l'assistant Html.Action () pour appeler une action dans le contrôleur Commentaires qui renvoie une telle vue partielle. La vue partielle peut fonctionner avec une collection d'objets Comment.

**Utilisation de l'assistant Html.Action ()**

@ Html.Action ("\_ CommentsForArticle", "Comment", new {ArticleID = Model.ID})

**Question:**Vous souhaitez afficher les avis des utilisateurs d'un produit sur chaque page de produit. Vous avez créé une vue partielle \_ReviewsForProduct. Souhaitez-vous utiliser Html.Partial () ou Html.Action () pour rendre cette vue?

## **Discussion: Scénarios de vue partielle**

Tenez compte des scénarios suivants lors de la création d'une application Web:

|  |  |
| --- | --- |
| • | Vous souhaitez afficher les dix derniers commentaires dans l'application Web. Lorsque les utilisateurs cliquent sur le lien Derniers commentaires sur la page d'accueil, ils verront une nouvelle page Web avec les commentaires affichés. |
| • | Vous souhaitez afficher les commentaires d'un article dans la vue Article. Vous souhaitez également afficher les commentaires d'un produit dans la vue Produit. Il existe des classes ArticleComment et ProductComment distinctes dans votre modèle, mais elles ont des propriétés similaires. |
| • | Vous disposez d'une application Web de partage de photos et vous souhaitez afficher une galerie d'images miniatures à la fois dans la vue AllPhotos et dans la vue Index de la page d'accueil. La vue AllPhotos doit afficher tous les objets Photo, mais la vue Index de la page d'accueil ne doit afficher que les trois photos les plus récentes téléchargées. |

Pour chaque scénario, discutez avec la classe:

|  |  |
| --- | --- |
| • | Que vous ayez besoin de créer une vue ou une vue partielle. |
| • | Que vous ayez besoin de créer une vue fortement typée ou une vue dynamique. |
| • | Si vous décidez de créer une vue partielle, si vous devez utiliser l'assistant HTML Html.Partial () ou Html.Action () dans la vue parent. |

# **Atelier: Développement de vues ASP.NET MVC 5**

### **Scénario**

Vous avez été invité à ajouter les vues suivantes à l'application de partage de photos:

|  |  |
| --- | --- |
| • | Une vue d'affichage pour les objets du modèle photo. Cette vue affichera une seule photo de grande taille, avec le titre, la description, le propriétaire et les propriétés de date de création. |
| • | Une vue Créer pour les objets du modèle Photo. Cette vue permettra aux utilisateurs de télécharger une nouvelle photo dans la galerie et de définir les propriétés du titre et de la description. |
| • | Une vue partielle de la galerie de photos. Cette vue affichera de nombreuses photos dans des tailles de vignettes, avec le titre, le propriétaire et les propriétés de date de création. Cette vue sera utilisée sur la page Web Toutes les photos pour afficher toutes les photos de l'application. De plus, cette vue sera également utilisée sur la page d'accueil pour afficher les trois photos les plus récentes. |

Après avoir ajouté ces trois vues à l'application de partage de photos, vous testerez également le fonctionnement de l'application Web.

### **Objectifs**

Après avoir terminé cet atelier, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Ajoutez des vues Razor à une application MVC et définissez des propriétés telles que l'échafaudage et la liaison de modèle. |
| • | Écrivez à la fois le balisage HTML et le code C # dans une vue à l'aide de la syntaxe Razor. |
| • | Créez une vue partielle et utilisez-la pour afficher le balisage réutilisable. |

##### **Configuration du laboratoire**

Durée estimée: 50 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD05_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD05_LAK.md>.

### **Exercice 1: Ajout d'une vue pour l'affichage de photos**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Créez une nouvelle vue dans l'application Web Partage de photos pour afficher des photos uniques en grande taille. |
| • | Affichez les propriétés d'une photo telles que le titre, la description et la date de création. |

### **Exercice 2: Ajout d'une vue pour les nouvelles photos**

##### **Scénario**

Dans cet exercice, vous allez

|  |  |
| --- | --- |
| • | Créez une vue pour télécharger de nouvelles photos à afficher dans l'application de partage de photos. |
| • | Affichez les propriétés d'une photo, telles que le titre, la description et la date de création. |

### **Exercice 3: Création et utilisation d'une vue partielle**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Ajoutez une action de galerie au contrôleur de photos. |
| • | Ajoutez une vue partielle de galerie de photos. |
| • | Complétez la vue partielle de la galerie de photos. |
| • | Utilisez la vue partielle de la galerie de photos. |

### **Exercice 4: Ajout d'une vue d'accueil et test des vues**

##### **Scénario**

Dans cet exercice, vous allez créer une page d'accueil qui réutilise l'objet de galerie de photos, mais n'affiche que les trois photos les plus récentes.

### **Question (s) de révision**

**Vérifiez vos connaissances**

**Découverte**

**Comment pouvez-vous améliorer l'accessibilité du code HTML rendu par vos vues photo?**

Afficher la solution Réinitialiser

**Vérifiez vos connaissances**

**Découverte**

**Dans le laboratoire, comment vous êtes-vous assuré que leCréervoir pourphotoles objets de modèle pouvaient télécharger des fichiers photo lorsque l'utilisateur cliquait sur leCréerbouton?**

Afficher la solution Réinitialiser

# **Examen du module et points à retenir**

Dans ce module, vous avez vu comment créer l'interface utilisateur de votre application Web en créant des vues avec le moteur de vue Razor. Les vues peuvent afficher les propriétés d'une classe de modèle et permettre aux utilisateurs de créer, lire, mettre à jour et supprimer des objets de cette classe de modèle. Certaines vues affichent de nombreux objets en parcourant une collection d'objets de modèle. Les méthodes HTML Helper intégrées à ASP.NET MVC 5 facilitent le rendu des affichages, des contrôles et des formulaires, et renvoient des messages de validation aux utilisateurs. Les vues partielles peuvent être réutilisées plusieurs fois dans votre application Web pour afficher des sections similaires de HTML sur plusieurs pages.

**Meilleur entrainement:**Utilisez les commentaires Razor, déclarés avec les délimiteurs @ \* \* @, dans toutes vos vues Razor pour aider à expliquer le code de vue aux autres développeurs de votre équipe.

**Meilleur entrainement:**N'utilisez les balises @: et <text> que lorsque Razor interprète mal le contenu en tant que code. Razor a une logique sophistiquée pour distinguer le contenu du code, c'est donc rarement nécessaire.

**Meilleur entrainement:**Utilisez des vues fortement typées dans la mesure du possible, car Visual Studio vous aide à écrire du code correct en affichant des commentaires IntelliSense.

**Problèmes courants et conseils de dépannage**

| **Problème commun** | **Astuce de dépannage** |
| --- | --- |
| Lorsqu'un contrôleur tente d'accéder à une vue partielle, une exception est levée. | Veuillez consulter le contenu du compagnon étudiant pour ce cours. |

### **Question (s) de révision**