Module 3: Développement de modèles 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/20486C03.html#P1) |
| **Leçon 1:** | [Création de modèles MVC](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C03.html#P2) |
| **Leçon 2:** | [Travailler avec des données](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C03.html#P3) |
| **Laboratoire:** | [Développement de modèles 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/20486C03.html#P4) |
|  | [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/20486C03.html#P5) |

# **Aperçu du module**

La plupart des applications Web doivent interagir avec différents types de données ou d'objets. Une application de commerce électronique, par exemple, permet de gérer les produits, les paniers d'achat, les clients et les commandes. Une application de réseau social peut aider à gérer les utilisateurs, les mises à jour de statut, les commentaires, les photos et les vidéos. Un blog est utilisé pour gérer les entrées de blog, les commentaires, les catégories et les balises. Lorsque vous écrivez une application Web MVC, vous créez un modèle MVC pour modéliser les données de votre application Web. Dans ce modèle, vous créez une classe de modèle pour chaque type d'objet. La classe de modèle décrit les propriétés de chaque type d'objet et peut inclure une logique métier correspondant aux processus métier. Par conséquent, le modèle est un élément fondamental dans une application MVC et un bon point de départ lorsque vous écrivez du code. Les modèles interagissent avec la banque d'informations qui sous-tend votre site, qui est généralement une base de données. Par conséquent, vous devez comprendre les techniques et technologies d'accès aux données pour écrire des modèles. Dans ce module, vous verrez comment créer le code pour les modèles et accéder aux données en utilisant Entity Framework et LINQ.

### **Objectifs**

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

|  |  |
| --- | --- |
| • | Ajoutez un modèle à une application MVC et écrivez-y du code pour implémenter la logique métier. |
| • | Créez une nouvelle base de données SQL Azure pour stocker les données de l'application Web. |

# Leçon 1: Création de modèles MVC

Un modèle MVC est une collection de classes .NET Framework. Lorsque vous créez une classe de modèle, vous définissez les propriétés et les méthodes qui conviennent au type d'objet décrit par la classe de modèle. Vous pouvez décrire ces propriétés dans le code afin que MVC puisse déterminer comment les rendre dans une page Web et comment valider l'entrée utilisateur. Vous devez savoir comment créer et décrire des modèles et comment modifier la manière dont MVC crée des instances de classe de modèle lorsqu'il exécute votre application Web.

## **Objectifs de la leçon**

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

|  |  |
| --- | --- |
| • | Décrivez comment créer des modèles MVC et développer une logique métier. |
| • | Utilisez les annotations d'affichage et de modification des données pour attribuer des attributs aux vues et aux contrôleurs. |
| • | Validez l'entrée utilisateur avec des annotations de données. |
| • | Décrivez les classeurs de modèles. |
| • | Décrivez l'extensibilité du modèle. |
| • | Ajoutez un modèle à une application Web MVC 5. |

## **Développer des modèles**

Chaque site Web présente des informations sur différents types d'objets. Dans votre application Web, vous devez définir des classes de modèle pour ces objets. Lorsque vous implémentez une exigence fonctionnelle dans votre application Web, vous commencez généralement par créer une classe de modèle. La classe de modèle sera probablement révisée lorsque vous créez le contrôleur et les vues pour le modèle, puis plus tard au cours des itérations qui se produisent pendant le projet. Si vous suivez le modèle de développement Agile ou Extreme Programming, vous commencez par une simple compréhension de la classe, peut-être son nom et quelques propriétés. Ensuite, vous discutez avec les utilisateurs et ajoutez des détails à la classe de modèle planifiée avec l'ensemble complet de propriétés et ses relations avec d'autres classes. Lors du développement du modèle, vous pouvez vous référer à des cas d'utilisation ou des user stories pour vous assurer que ces détails du modèle sont corrects.

**Exemple de classe de modèle**

Photo de classe publique

{

public int PhotoID {get; ensemble; }

chaîne publique Title {get; ensemble; }

octet public [] PhotoFile {obtenir; ensemble; }

chaîne publique Description {get; ensemble; }

public DateTime CreatedDate {get; ensemble; }

chaîne publique Owner {get; ensemble; }

public virtual ICollection <Commentaire> Commentaires {get; ensemble; }

}

Notez que la classe de modèle n'hérite d'aucune autre classe. Notez également que vous avez créé des propriétés publiques pour chaque propriété du modèle et que vous avez inclus le type de données, tel qu'un entier ou une chaîne, dans la déclaration. Vous pouvez créer des propriétés en lecture seule en omettant l'ensemble; mot-clé.

La classe Photo comprend une propriété appelée Commentaires. Ceci est déclaré comme une collection d'objets Comment et implémente un côté de la relation entre les photos et les commentaires.

**Classe de modèle de commentaire**

Commentaire de classe publique

{

public int CommentID {get; ensemble; }

public int PhotoID {get; ensemble; }

chaîne publique UserName {get; ensemble; }

public string Subject {get; ensemble; }

chaîne publique Body {get; ensemble; }

Photo virtuelle publique Photo {get; ensemble; }

}

Notez que la classe Comment inclut une propriété PhotoID. Cette propriété stocke l'ID de la photo sur laquelle l'utilisateur a commenté et lie le commentaire à une seule photo. Notez également que la classe Comment inclut une propriété Photo, qui renvoie l'objet Photo auquel le commentaire se rapporte. Ces propriétés implémentent l'autre côté de la relation entre les photos et les commentaires. Les instances des classes de modèle sont généralement créées dans une action de contrôleur et transmises à une vue à afficher.

**Instanciation d'une photo dans une action de contrôleur**

Photo newPhoto = nouvelle photo ();

newPhoto.Title = "Ceci est une photo d'exemple";

newPhoto.Owner = User.Identity.Name;

newPhoto.CreatedDate = DateTime.Today;

return View ("DisplayView", newPhoto);

**Rendu d'une propriété à partir d'une classe de modèle**

<div id = "photo-title">

@ Model.Title

</div>

## **Utilisation des annotations d'affichage et de modification des données sur les propriétés**

Les classes de modèle spécifient généralement trois attributs pour chaque propriété:

|  |  |
| --- | --- |
| • | Le nom de la propriété, par exemple, Titre |
| • | Le type de données de la propriété, par exemple, String |
| • | Les niveaux d'accès de la propriété, par exemple, les mots-clés get et set pour indiquer l'accès en lecture et en écriture |

En outre, à l'aide d'attributs, vous pouvez fournir des métadonnées supplémentaires pour décrire les propriétés à ASP.NET MVC. Le moteur d'exécution MVC utilise ces métadonnées pour déterminer comment rendre chaque propriété dans les vues pour l'affichage et la modification. Ces attributs sont appelés annotations d'affichage et de modification.

Par exemple, les noms de propriété en C # ne peuvent pas contenir d'espaces. Sur une page Web rendue, vous souhaiterez souvent inclure des espaces dans une étiquette de propriété. Par exemple, vous souhaiterez peut-être rendre une propriété appelée CreatedDate avec le libellé Date de création. Pour fournir à MVC ces informations, vous pouvez utiliser l'annotation DisplayName.

**Définition de l'annotation DisplayName**

[DisplayName ("Date de création")]

public CreatedDate {get; ensemble; }

Si vous disposez d'une propriété DateTime, vous pouvez utiliser les annotations d'affichage et de modification des données pour indiquer à MVC le format dans lequel vous souhaitez afficher la propriété.

**Définition du DataType et du DisplayFormat**

[DisplayName ("Date de création")]

[DataType (DataType.DateTime)]

[DisplayFormat (DataFormatString = "{0: MM / jj / aa}", ApplyFormatInEditMode = true)]

public DateTime CreatedDate {get; ensemble; }

Toutes les annotations de données fournies avec ASP.NET MVC 5 sont incluses dans l'espace de noms System.ComponentModel.DataAnnotations.

**Question:**Dans le code sur la diapositive, comment pouvez-vous reconnaître l'affichage et modifier les annotations et les distinguer des autres codes?

## **Validation de l'entrée utilisateur avec des annotations de données**

Vous pouvez utiliser des annotations de données dans les modèles MVC pour définir des critères de validation pour l'entrée utilisateur. La validation d'entrée est le processus par lequel MVC vérifie les données fournies par un utilisateur ou une requête Web pour s'assurer qu'elles sont au bon format. L'exemple suivant montre un formulaire de page Web qui recueille des informations auprès de l'utilisateur:

|  |  |
| --- | --- |
| • | Nom. Ceci est une entrée requise. L'utilisateur doit saisir certaines données dans ce champ. |
| • | Hauteur. Ce doit être un entier compris entre 0 et 400. |
| • | Adresse e-mail. Ceci est une entrée requise. La valeur saisie doit être une adresse e-mail valide. |

Dans l'exemple suivant, lorsque l'utilisateur soumet le formulaire, vous voulez que MVC crée une nouvelle instance du modèle Person et l'utilise pour stocker les données dans la base de données. Cependant, vous voulez vous assurer que les données sont valides avant d'être stockées.

**Utilisation des annotations de données de validation**

Personne de classe publique

{

public int PersonID {get; ensemble; }

[Obligatoire]

chaîne publique Name {get; ensemble; }

[Plage (0, 400)]

public int Hauteur {get; ensemble; }

[Obligatoire]

[RegularExpression (". + \\ @. + \\ .. +")]

chaîne publique EmailAddress {get; ensemble; }

}

Les annotations Required, Range, StringLength et RegularExpression implémentent la validation d'entrée dans MVC. Si les utilisateurs n'entrent pas de données qui satisfont aux critères spécifiés pour chaque propriété, la vue affiche un message d'erreur standard qui invite l'utilisateur à entrer les données correctes. Dans l'exemple précédent, vous pouvez voir que l'utilisateur doit entrer un nom et une hauteur entre 0 et 400. Pour la propriété EmailAddress, l'utilisateur doit entrer une valeur qui correspond à l'expression régulière. L'expression régulière de l'exemple est une expression simple qui nécessite un symbole @ et un point.

**Définition d'un message d'erreur de validation**

[Obligatoire (ErrorMessage = "Veuillez saisir un nom.")]

chaîne publique Name {get; ensemble; }

**Noter:**Vous verrez comment vous assurer que le message d'erreur de validation est affiché dans une vue.

**Lecture supplémentaire:**Pour plus d'exemples d'annotations de données de validation, voir

<http://go.microsoft.com/fwlink/?LinkID=288950&clcid=0x409>

**Lecture supplémentaire:**Pour plus d'informations sur les expressions régulières que vous pouvez utiliser pour vérifier les entrées utilisateur, consultez<http://go.microsoft.com/fwlink/?LinkID=288951&clcid=0x409>

**Question**: Vous voulez vous assurer que les utilisateurs entrent un mot de passe de plus de six caractères. Comment procéder en utilisant une annotation de données de validation?

## **Que sont les classeurs de modèles?**

Un classeur de modèle est un composant d'une application ASP.NET MVC qui crée une instance d'une classe de modèle, en fonction des données envoyées dans la demande à partir du navigateur Web. ASP.NET MVC comprend un classeur de modèles par défaut qui répond aux besoins de la plupart des applications Web. Cependant, vous devez savoir comment le classeur de modèle par défaut fonctionne avec d'autres composants pour l'utiliser correctement. De plus, vous pouvez choisir de créer un classeur de modèles personnalisé pour les situations avancées.

**Que fait un classeur de modèles?**

Un classeur de modèles garantit que les bonnes données sont envoyées aux paramètres dans une méthode d'action du contrôleur. Cela permet à MVC de créer des instances de classes de modèle qui satisfont la demande de l'utilisateur. Le classeur de modèle par défaut, par exemple, examine à la fois la définition des paramètres d'action du contrôleur et les paramètres de requête pour déterminer les valeurs de requête à transmettre à quel paramètre d'action.

Ce processus de liaison de modèle peut faire gagner beaucoup de temps aux développeurs et éviter de nombreuses erreurs d'exécution inattendues résultant de paramètres incorrects. MVC comprend un classeur de modèle par défaut avec une logique sophistiquée qui transmet correctement les paramètres dans presque tous les cas sans code personnalisé complexe.

**L'invocateur d'action du contrôleur et le classeur de modèle par défaut**

Pour comprendre le processus de liaison de modèle par défaut, considérez la demande suivante d'un navigateur Web:

http://www.adventure-works.com/product/display/45

Cette demande identifie trois aspects:

|  |  |
| --- | --- |
| • | La classe de modèle qui intéresse l'utilisateur. L'utilisateur a demandé un produit. |
| • | L'opération à effectuer sur la classe de modèle. L'utilisateur a demandé que le produit soit affiché. |
| • | Instance spécifique de la classe de modèle. L'utilisateur a demandé que le produit avec l'ID 45 soit affiché. |

La demande est reçue par un objet appelé l'invocateur d'action du contrôleur. L'invocateur d'action de contrôleur du runtime MVC appelle une action de contrôleur et lui transmet les paramètres corrects. Dans l'exemple, l'appelant d'action appelle l'action Afficher dans le contrôleur de produit et transmet l'ID «45» en tant que paramètre à l'action, afin que le bon produit puisse être affiché.

La classe ControllerActionInvoker est l'appelant d'action par défaut. Cet appelant d'action utilise des classeurs de modèle pour déterminer comment les paramètres sont passés aux actions.

**Comment le classeur de modèle par défaut passe les paramètres**

Dans une application MVC par défaut, il n'y a qu'un seul classeur de modèle à utiliser par ControllerActionInvoker. Ce classeur est une instance de la classe DefaultModelBinder. Le classeur de modèle par défaut transmet les paramètres à l'aide de la logique suivante:

|  |  |
| --- | --- |
| 1. | Le classeur examine la définition de l'action à laquelle il doit passer des paramètres. Dans l'exemple, le classeur détermine que l'action nécessite un paramètre entier appelé PhotoID. |
| 2. | Le classeur recherche des valeurs dans la demande qui peuvent être transmises en tant que paramètres. Dans l'exemple, le classeur recherche des entiers car l'action nécessite un entier. Le classeur recherche les valeurs aux emplacements suivants, dans l'ordre:   |  |  | | --- | --- | | une. | Valeurs de formulaire. Si l'utilisateur remplit un formulaire et clique sur un bouton d'envoi, vous pouvez trouver des paramètres dans la collection Request.Form. | | b. | Valeurs d'itinéraire. Selon les itinéraires que vous avez définis dans votre application Web, le classeur de modèles peut être en mesure d'identifier des paramètres dans l'URL. Dans l'exemple d'URL, «45» est identifié comme paramètre par la route MVC par défaut. | | c. | Chaînes de requête. Si la demande utilisateur inclut des paramètres nommés après un point d'interrogation, vous pouvez trouver ces paramètres dans la collection Request.QueryString. | | ré. | Des dossiers. Si la demande de l'utilisateur comprend des fichiers téléchargés, ceux-ci peuvent être utilisés comme paramètres. | |

Notez que s'il existe des valeurs de formulaire et des valeurs d'itinéraire dans la demande, les valeurs de formulaire ont la priorité. Les valeurs de chaîne de requête ne sont utilisées que s'il n'y a aucune valeur de formulaire et aucune valeur d'itinéraire disponible en tant que paramètres.

## **Extensibilité du modèle**

L'architecture MVC a été conçue pour fournir une extensibilité afin que les développeurs puissent adapter l'architecture à des exigences inhabituelles ou uniques. Par exemple, l'invocateur d'action par défaut, ControllerActionInvoker, peut être remplacé par votre propre invocateur d'action si vous souhaitez implémenter votre propre logique d'appel.

Vous pouvez étendre la gestion MVC des modèles MVC de deux manières: créer des annotations de données personnalisées et des classeurs de modèles personnalisés.

**Annotations de données de validation personnalisées**

Vous pouvez utiliser des annotations de données pour indiquer à MVC comment valider les données qu'un utilisateur entre dans un formulaire ou transmet dans des chaînes de requête. Il existe des attributs de validation intégrés dans MVC 5, tels que Required, Range, StringLength et RegularExpression. Ces attributs de validation sont très flexibles. Toutefois, dans certaines situations, telles que les exemples suivants, vous souhaiterez peut-être exécuter du code de validation personnalisé:

|  |  |
| --- | --- |
| • | Exécution d'une vérification du magasin de données. Vous souhaitez vérifier les données saisies par rapport aux données qui ont déjà été stockées dans la base de données ou dans un autre magasin de bases de données. |
| • | Vérifications mathématiques. Vous souhaitez calculer une valeur à partir des données saisies et vérifier que la valeur est valide. |

Dans de telles situations, vous pouvez créer une annotation de données de validation personnalisée. Pour ce faire, vous créez une classe qui hérite de la classe System.ComponentModel.DataAnnotations.ValidationAttribute.

**Création d'une annotation de données de validation personnalisée**

[AttributeUsage (AttributeTargets.Field)]

classe publique LargerThanValidationAttribute: ValidationAttribute

{

public int MinimumValue {get; ensemble; }

//Constructeur

public LargerThanValidationAttribute (int minimum)

{

MinimumValue = minimum;

}

// Vous devez remplacer la méthode IsValid pour exécuter votre test

public override Boolean IsValid (valeur de l'objet)

{

var valueToCompare = valeur (int);

if (valueToCompare> MinimumValue)

{

retourne vrai;

}

autre

{

retourne faux;

}

}

}

**Utilisation d'une annotation de données de validation personnalisée**

[LargerThanValidationAttribute (18)]

public VoterAge {get; ensemble; }

**Classeurs de modèles personnalisés**

L'invocateur d'action de contrôleur par défaut reçoit les demandes et appelle la bonne action sur le bon contrôleur pour répondre à la demande. L'invocateur d'action par défaut utilise des classeurs de modèle pour identifier les paramètres de la demande et les transmettre tous aux bons paramètres de l'action. Ce mécanisme garantit que les chaînes de requête, les valeurs d'itinéraire, les valeurs de formulaire et les fichiers téléchargés sont disponibles pour l'action à utiliser.

Le classeur de modèles par défaut est sophistiqué et flexible, mais parfois, vous souhaiterez peut-être personnaliser son comportement pour transmettre des paramètres de manière inhabituelle. Vous pouvez le faire en créant un classeur de modèles personnalisé et en l'enregistrant avec le moteur d'exécution MVC.

**Un classeur de modèle personnalisé simple**

public class CarModelBinder: IModelBinder

{

objet public BindModel (ControllerContext controllerContext, ModelBindingContext bindingContext)

{

// Récupère la couleur de la voiture à partir du formulaire de demande

string color = controllerContext.HttpContext.Request.Form ["color"];

// Récupère la marque de la voiture à partir du formulaire de demande

string brand = controllerContext.HttpContext.Request.Form ["brand"];

// Créer une nouvelle instance du modèle de voiture

Voiture newCar = nouvelle voiture ();

newCar.color = couleur;

newCar.brand = marque;

// rendre la voiture

return newCar;

}

}

L'exemple de code suppose que vous avez une classe de modèle dans votre application MVC appelée, Car. Il suppose également que toute demande d'objet Car inclut des valeurs de couleur et de marque dans la collection de formulaires. Cette situation peut être facilement gérée par le classeur de modèles par défaut. Cependant, cet exemple montre comment les classeurs de modèles peuvent localiser des valeurs à partir du contexte de la demande et transmettre ces valeurs aux bonnes propriétés du modèle. Vous pouvez ajouter du code personnalisé pour implémenter des fonctionnalités supplémentaires.

**Question**: Vous voulez vous assurer que lorsqu'un utilisateur saisit une valeur dans la zone Numéro de modèle de voiture lors de l'ajout d'une nouvelle voiture sur le site Web, le texte saisi n'est pas déjà utilisé par une autre voiture dans la base de données. Souhaitez-vous utiliser une annotation de données de validation personnalisée ou un classeur de modèle personnalisé pour cela?

## **Démonstration: comment ajouter un modèle**

Dans cette démonstration, vous verrez comment créer un modèle dans une application Web ASP.NET MVC 5 et ajouter des classes de modèle à l'application Web. Vous verrez également comment ajouter des annotations de données et créer une annotation de données de validation personnalisée.

### **Étapes de démonstration**

|  |  |
| --- | --- |
| • | Vous trouverez les étapes dans la section «Leçon 1: Création de modèles MVC» sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD03_DEMO.md>. |

# Leçon 2: Travailler avec des données

Toutes les applications Web présentent des informations et presque toutes les applications Web nécessitent un magasin de données pour ces informations. En rendant des pages Web à l'aide des données d'un magasin de données, vous pouvez créer une application Web qui change continuellement en réponse aux entrées de l'utilisateur, aux actions administratives et aux événements de publication. Le magasin de données est généralement une base de données, mais d'autres magasins de données sont parfois utilisés. Dans les applications MVC, vous pouvez créer un modèle qui implémente la logique d'accès aux données et la logique métier. Vous pouvez également séparer la logique métier, dans les classes de modèle, de la logique d'accès aux données, dans un référentiel. Un référentiel est une classe qu'un contrôleur peut appeler pour lire et écrire des données depuis et vers un magasin de données. Le .NET Framework inclut les technologies Entity Framework et LINQ, qui rendent le code d'accès aux données très rapide à écrire et simple à comprendre. En plus,

## **Objectifs de la leçon**

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

|  |  |
| --- | --- |
| • | Connectez une application à une base de données pour accéder et stocker des données. |
| • | Décrivez les fonctionnalités d'Entity Framework. |
| • | Utilisez LINQ pour écrire des requêtes de sélection, de filtrage et de regroupement de données. |
| • | Créez des classes de modèle distinctes et des classes de référentiel correspondantes à l'aide du code Entity Framework. |
| • | Expliquez comment accéder aux données dans les modèles et les référentiels. |

## **Connexion à une base de données**

La plupart des sites Web utilisent une base de données pour stocker des données dynamiques. En incluant ces données dans les pages HTML rendues, vous pouvez créer une application Web dynamique avec un contenu qui change fréquemment. Par exemple, vous pouvez fournir des pages Web administratives qui permettent aux employés de l'entreprise de mettre à jour le catalogue de produits et de publier des actualités. Les produits et articles sont stockés dans la base de données. Dès qu'ils sont stockés, les utilisateurs peuvent les consulter et les lire. Les employés n'ont pas besoin de modifier le code HTML ou de republier le site Web pour rendre leurs modifications visibles.

Certains sites Web peuvent stocker des données dans d'autres emplacements, comme un service d'annuaire, mais les bases de données sont le magasin de données le plus utilisé.

**ADO.NET et bases de données**

Lorsque vous créez des applications .NET Framework, y compris des applications Web MVC, vous pouvez utiliser la technologie ADO.NET pour accéder aux bases de données. Les classes ADO.NET sont contenues dans l'espace de noms System.Data. ADO.NET prend en charge un large éventail de bases de données en utilisant différents fournisseurs de données. Par exemple:

|  |  |
| --- | --- |
| • | Microsoft SQL Server. Il s'agit d'un serveur de base de données de premier plan de Microsoft. ADO.NET inclut le fournisseur SqlClient pour toutes les bases de données SQL Server. |
| • | Microsoft SQL Server Express. Il s'agit d'une version gratuite de SQL Server qui comprend un large éventail de fonctionnalités de base de données et est très flexible. Certaines fonctionnalités avancées ne sont pas possibles avec SQL Express. Le fournisseur SqlClient est utilisé pour SQL Express. |
| • | OLE DB. Il s'agit d'une norme à laquelle adhèrent de nombreuses bases de données différentes. ADO.NET inclut le fournisseur OleDb pour toutes les bases de données OLE DB. |
| • | ODBC. Il s'agit d'une autre norme plus ancienne à laquelle adhèrent de nombreuses bases de données différentes. ADO.NET inclut le fournisseur Odbc pour toutes les bases de données ODBC. En général, vous devez utiliser un fournisseur OLE DB, s'il est disponible, au lieu d'un fournisseur ODBC. |

Vous pouvez également utiliser des fournisseurs ADO.NET tiers pour accéder à d'autres bases de données.

**Bases de données cloud**

La base de données d'une application Web est généralement située sur le même serveur que l'application Web elle-même ou sur un serveur de base de données dédié sur le même site physique. Cependant, si vous disposez d'une connexion Internet rapide et fiable, vous pouvez envisager d'utiliser une base de données cloud. De plus, si vous avez choisi d'héberger une application Web dans le cloud, une base de données cloud est une solution de stockage logique. Le service de base de données cloud Microsoft s'appelle SQL Database et fait partie de Microsoft Azure.

Microsoft Azure SQL Database présente les avantages suivants:

|  |  |
| --- | --- |
| • | Les bases de données fonctionnent dans les centres de données Microsoft avec la meilleure connectivité et fiabilité. |
| • | Microsoft garantit jusqu'à 99% de disponibilité. |
| • | Vous n'avez pas besoin de créer et de maintenir vos propres serveurs de base de données ou d'employer des administrateurs de base de données. |
| • | Vous pouvez faire évoluer les bases de données très facilement. |
| • | Vous ne payez que pour les données que vous utilisez et diffusez. |

Vous pouvez utiliser Microsoft Azure SQL Database avec ADO.NET à l'aide du fournisseur SqlClient.

**Connexion d'une application Web MVC à une base de données**

Pour utiliser ADO.NET et vous connecter à une base de données, vous devez ajouter deux éléments à votre application:

|  |  |
| --- | --- |
| • | Ajoutez une référence à l'espace de noms System.Data. |
| • | Ajoutez une chaîne de connexion au fichier Web.config. Cette chaîne spécifie le fournisseur, l'emplacement de la base de données, les propriétés de sécurité et d'autres propriétés en fonction du fournisseur. |

**Connexion à SQL Express**

<connectionStrings>

<add name = "PhotoSharingDB"

connectionString = "Source de données =. \ SQLEXPRESS; Catalogue initial = PhotoSharingDB;" +

"Sécurité intégrée = SSPI"

providerName = "System.Data.SqlClient" />

</connectionStrings>

**Connexion à la base de données Microsoft Azure SQL**

<connectionStrings>

<add name = "PhotoSharingDB"

connectionString = "Serveur = tcp: exemple.database.windows.net, 1433; Base de données = PhotoSharingDB;" +

" ID utilisateur = Admin @ exemple ; Mot de passe = Pa $$ w0rd; Trusted\_Connection = False;" +

"Encrypt = True; Délai d'expiration de la connexion = 30; PersistSecurityInfo = true"

providerName = "System.Data.SqlClient" />

</connectionStrings>

La balise <connectionStrings> doit apparaître dans la balise <configuration>, après la balise <configSections>.

## **Le cadre d'entité**

Les développeurs écrivent du code qui fonctionne avec des classes et des objets. En revanche, les bases de données stockent les données dans des tables avec des colonnes et des lignes. Les administrateurs de base de données créent et analysent des bases de données en exécutant des requêtes Transact-SQL. Vous pouvez choisir de créer et d'exécuter des requêtes Transact-SQL dans votre code ADO.NET. Cependant, ADO.NET inclut Entity Framework - cette technologie vous permet de lire et d'écrire des données vers et à partir d'une base de données à l'aide de classes et d'objets.

Entity Framework est un framework ORM (Object Relational Mapping). Un framework ORM mappe les tables et colonnes trouvées dans une base de données sur des objets et leurs propriétés que vous pouvez appeler à partir du code .NET.

**Flux de travail Entity Framework**

La façon dont vous utilisez Entity Framework dans votre application dépend de la manière dont vous souhaitez créer votre base de données. Plusieurs flux de travail Entity Framework sont disponibles:

|  |  |
| --- | --- |
| • | La base de données d'abord. Vous pouvez utiliser le flux de travail basé sur la base de données lorsque vous disposez d'une base de données préexistante ou si vous préférez créer une nouvelle base de données en définissant une table, des colonnes, des vues et d'autres objets de schéma de base de données. Dans ce flux de travail, Entity Framework examine la base de données et crée un fichier XML avec une extension .edmx appelé fichier de modèle. Le fichier modèle décrit les classes avec lesquelles vous pourrez travailler, dans le code. Vous pouvez ajuster le modèle à l'aide d'un concepteur dans Visual Studio, puis en écrivant du code sur les classes Entity Framework. |
| • | Modèle d'abord. Vous pouvez utiliser le flux de travail basé sur le modèle lorsque vous ne disposez pas encore d'une base de données et que vous préférez concevoir votre modèle à l'aide d'un outil de modélisation ORM. Dans ce flux de travail, vous créez le fichier .edmx dans le concepteur Visual Studio, puis vous écrivez du code sur les classes de modèle générées par le concepteur. Lorsque vous exécutez l'application, Entity Framework crée les tables et colonnes de base de données pour prendre en charge le modèle. Dans ce flux de travail, Visual Studio peut également créer une chaîne de connexion pour vous et l'insérer dans Web.config, en fonction de la base de données que vous spécifiez. |
| • | Code d'abord. Vous pouvez utiliser le flux de travail basé sur le code si vous préférez créer un modèle en écrivant des classes .NET Framework. Dans ce flux de travail, il n'y a pas de fichier de modèle. Au lieu de cela, vous créez des classes de modèle en C # ou Visual Basic. Lorsque vous exécutez l'application, si la base de données n'existe pas, Entity Framework crée la base de données. |

**Ajout d'un contexte de structure d'entité**

Lorsque vous utilisez Entity Framework dans le flux de travail code-first, vous devez vous assurer que l'infrastructure crée la base de données et les tables appropriées pour stocker vos classes de modèle. Pour ce faire, créez une classe qui hérite de la classe Entity Framework DbContext. Vous utiliserez cette classe dans les contrôleurs lorsque vous souhaitez manipuler des données dans la base de données. Dans cette classe, ajoutez une propriété DbSet <> pour chaque table de base de données que vous souhaitez que Entity Framework crée dans la nouvelle base de données.

**Une classe de contexte Entity Framework**

classe publique PhotoSharingDB: DbContext

{

public DbSet <Photo> Photos {get; ensemble; }

public DbSet <Commentaire> Commentaires {get; ensemble; }

}

Dans l'exemple précédent, Entity Framework recherche une chaîne de connexion avec le nom, PhotoSharingDB, pour correspondre au nom de la classe DbContext. Entity Framework crée la base de données à l'emplacement fourni par la chaîne de connexion et crée deux tables dans la nouvelle base de données:

|  |  |
| --- | --- |
| • | **Photos:**Cette table aura des colonnes qui correspondent aux propriétés de la classe de modèle Photo. |
| • | **Commentaires:**Cette table aura des colonnes qui correspondent aux propriétés de la classe de modèle Comment. |

**Question:**Vous disposez d'un diagramme Microsoft Visio, créé par un analyste métier, qui montre toutes les classes de modèle de votre application Web et leurs relations. Vous souhaitez recréer ce diagramme dans Visual Studio. Quel flux de travail Entity Framework devez-vous utiliser?

## **Utilisation d'un contexte Entity Framework**

Maintenant que vous avez défini le contexte Entity Framework et les classes de modèle, vous pouvez les utiliser dans les contrôleurs MVC pour transmettre des données aux vues à afficher.

**Utilisation d'un contexte Entity Framework dans un contrôleur**

public class PhotoController: Controller

{

// Création d'une référence à la classe de contexte Entity Framework

private PhotoSharingDB db = nouveau PhotoSharingDB ();

// Cette action récupère toutes les photos de la base de données et les transmet à la vue Index

Index ActionResult public ()

{

return View ("Index", db.Photos.ToList ());

}

// Cette action obtient une photo avec un identifiant particulier et la transmet à la vue Détails

public ActionResult Details (int id = 0)

{

Photo photo = db.Photos.Find (id);

si (photo == null)

{

return HttpNotFound ();

}

retour Vue ("Détails", photo);

}

}

}

**Utilisation d'initialiseurs pour remplir des bases de données**

Si vous utilisez le flux de travail code-first ou model-first, Entity Framework crée la base de données la première fois que vous exécutez l'application et accédez aux données. La base de données reste vide si vous avez créé le schéma de base de données mais ne l'avez pas rempli avec des lignes de données.

Vous pouvez utiliser une classe d'initialisation pour remplir la base de données avec des exemples de données. Cette technique garantit qu'il existe des exemples de données avec lesquels travailler pendant le développement.

**Un exemple d'initialiseur Entity Framework**

classe publique PhotoSharingInitializer: DropCreateDatabaseAlways <PhotoSharingDB>

{

// Remplace la méthode Seed pour remplir la base de données

protection de remplacement void Seed (contexte PhotoSharingDB)

{

// Créer une liste d'objets photo

var photos = nouvelle liste <Photo>

{

Nouvelle photo {

Title = "Ma première photo",

Description = "Ceci fait partie des exemples de données",

UserName = "Fred"

},

Nouvelle photo {

Title = "Ma deuxième photo",

Description = "Ceci fait partie des exemples de données",

UserName = "Sue"

}

};

// Ajout de la liste des photos à la base de données et sauvegarde des modifications

photos.ForEach (s => context.Photos.Add (s));

context.SaveChanges ();

}

}

**Exécution de l'initialiseur dans Global.asax**

protected void Application\_Start ()

{

// Amorcez la base de données avec des exemples de données pour le développement. Ce code doit être supprimé pour la production.

Database.SetInitializer <PhotoSharingDB> (nouveau PhotoSharingInitializer ());

}

**Question:**Vous avez créé une classe de contexte Entity Framework dans votre modèle, ajouté une initialisation et appelé Database.SetInitializer () à partir de Global.asax. Lorsque vous exécutez l'application, aucune base de données n'est créée et aucun objet modèle n'est affiché sur les pages Web. Qu'as-tu oublié de faire?

## **Utilisation de LINQ to Entities**

Language Integrated Query (LINQ) est un ensemble d'extensions de Visual C # et Visual Basic qui vous permettent d'écrire des expressions de requête complexes. Vous pouvez utiliser ces expressions pour extraire des données de bases de données, d'objets énumérables, de documents XML et d'autres sources de données. Les expressions sont similaires aux requêtes Transact-SQL, mais utilisent des mots clés C # ou VB afin que vous puissiez obtenir la prise en charge d'IntelliSense et la vérification des erreurs dans Visual Studio.

**Qu'est-ce que LINQ to Entities?**

LINQ to Entities est la version de LINQ qui fonctionne avec Entity Framework. LINQ to Entities vous permet d'écrire des requêtes complexes et sophistiquées pour localiser des données spécifiques, joindre des données à partir de plusieurs objets, mettre à jour des données et effectuer d'autres actions sur des objets à partir d'un contexte Entity Framework. Si vous utilisez Entity Framework, vous pouvez écrire des requêtes LINQ partout où vous avez besoin d'une instance spécifique d'une classe de modèle, d'un ensemble d'objets ou pour des besoins d'application plus complexes. Vous pouvez écrire des requêtes LINQ dans la syntaxe de requête, qui ressemble à la syntaxe SQL ou à la syntaxe de méthode, dans laquelle des opérations telles que «sélectionner» sont appelées en tant que méthodes sur des objets.

**Exemples de requêtes LINQ**

**Utilisation de LINQ pour obtenir des photos récentes**

// Cette liste stockera les photos qui sont retournées

Liste des photos <Photo>;

// Ceci est le contexte Entity Framework

Contexte PhotoSharingDB = nouveau PhotoSharingDB ();

si (nombre == 0)

{

// Si un nombre de photos n'est pas spécifié, nous obtiendrons toutes les photos de la base de données

// Cet exemple est dans la syntaxe de la méthode.

photos = context.Photos.ToList ();

}

autre

{

// Le nombre spécifie le nombre des photos les plus récentes demandées par l'utilisateur

// Utilisez une requête LINQ avec à la fois la syntaxe de requête et de méthode pour les obtenir à partir de la base de données

photos = (de p en contexte.

orderby p. crééDate décroissante

sélectionnez p) .Take (nombre) .ToList ();

}

**Lecture supplémentaire:**Pour lire de nombreux autres exemples de requêtes LINQ, consultez<http://go.microsoft.com/fwlink/?LinkID=288954&clcid=0x409>

## **Démonstration: Comment utiliser le code Entity Framework**

Dans cette démonstration, vous verrez comment ajouter une chaîne de connexion, un contexte Entity Framework et un initialiseur à l'application Web.

### **Étapes de démonstration**

|  |  |
| --- | --- |
| • | Vous trouverez les étapes dans la section «Leçon 2: Travailler avec des données» sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD03_DEMO.md>. |

## **Accès aux données dans les modèles et les référentiels**

Dans les applications MVC, vous pouvez placer le code d'accès aux données dans le modèle, avec la logique métier. Cependant, de nombreux architectes logiciels préfèrent séparer ces deux types de code car ils servent des objectifs différents:

|  |  |
| --- | --- |
| • | Logique métier. Ce code définit les objets gérés par l'application Web, leurs propriétés et leurs relations les uns avec les autres. |
| • | Logique d'accès aux données. Ce code définit les opérations nécessaires pour conserver les données dans une base de données. Cela inclut les opérations de création de nouveaux enregistrements, de lecture d'enregistrements, de mise à jour d'enregistrements et de suppression d'enregistrements dans les tables de base de données. Un seul objet de la couche de logique métier peut extraire des données de plusieurs tables de base de données. Cette abstraction est gérée par la logique d'accès aux données. |

MVC ne vous oblige pas à séparer la logique d'accès métier et aux données, et vous pouvez créer des classes de modèle MVC qui implémentent les deux couches. Cela se fait souvent dans des applications Web petites ou simples avec de petites équipes de développement. Dans ces scénarios, les classes Entity Framework sont utilisées directement dans les classes de modèle.

Dans des situations plus complexes, vous devez placer la logique métier dans des classes de modèle MVC et placer la logique d'accès aux données dans des classes dédiées appelées référentiels. Lorsque vous adoptez cette approche, les classes de modèle sont indépendantes de la structure de la base de données et n'incluent pas de code qui dépend des noms de table de base de données, des noms de colonne ou des noms de vue. Cette approche facilite la refonte de la base de données ou la migration vers un magasin de données ou une technologie d'accès aux données différent, sans avoir à recoder l'ensemble de votre application. En utilisant cette approche, vous utilisez Entity Framework dans vos classes de référentiel, mais pas dans vos classes de modèle.

**Comment séparer les classes de modèles et les référentiels**

Si vous choisissez de séparer la logique métier et la logique d'accès aux données, vous devez suivre les étapes suivantes pour chaque classe de modèle:

|  |  |
| --- | --- |
| 1. | Définissez une interface pour la classe de référentiel. Cette interface déclare les méthodes que la classe de référentiel utilise pour lire et écrire des données depuis et vers la base de données. |
| 2. | Créez et écrivez du code pour la classe de référentiel. Cette classe doit implémenter toutes les méthodes d'accès aux données déclarées dans l'interface. |
| 3. | Supprimez tous les codes d'accès aux données de la classe de modèle. |
| 4. | Modifiez la classe de contrôleur pour utiliser la classe de référentiel. Au lieu de cela, créez une instance de la classe de référentiel et appelez ses méthodes pour créer le modèle. |

**Un exemple simple de classe de modèle et de classe de référentiel**

Pour illustrer comment séparer les classes de modèle et les classes de référentiel, les exemples suivants implémentent un commentaire sur une photo.

**La classe de modèle de commentaire**

Commentaire de classe publique

{

public int CommentID {get; ensemble; }

public int PhotoID {get; ensemble; }

chaîne publique CommentText {get; ensemble; }

Photo virtuelle publique Photo {get; ensemble; }

}

**L'interface ICommentRepository**

interface publique ICommentRepository {

ICollection <Comment> GetComments (int PhotoID);

}

**La classe CommentRepository**

public class CommentRepository: ICommentRepository

{

public ICollection <Comment> GetComments (int PhotoID)

{

// Implémentez les appels de framework d'entité ici.

}

}

**La classe CommentsController**

public class CommentairesController: Controller

{

ICommentRepository commentRepository = nouveau CommentRepository;

public ActionResult DisplayCommentsForPhoto (int PhotoID)

{

// Utilisez le référentiel pour obtenir les commentaires

ICollection <Comments> comments = commentRepository.GetComments (PhotoID);

return View ("DisplayComments", commentaires);

}

}

**Noter:**En utilisant l'interface ICommentRepository, le code facilite le remplacement de CommentRespository par une autre implémentation si vous en avez besoin. Cependant, le code CommentController crée toujours un objet CommentRespository. Vous devez modifier l'objet pour effectuer le remplacement.

Dans une architecture encore meilleure, vous pouvez remplacer CommentRepository par une implémentation différente de ICommentRepository sans aucune modification de la classe CommentController. Il s'agit d'une approche extrêmement flexible et adaptable et s'appelle une architecture faiblement couplée.

Les architectures faiblement couplées sont également essentielles pour les tests unitaires. Vous verrez comment créer une architecture faiblement couplée dans le module 6 «Test et débogage des applications Web ASP.NET MVC 5».

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

### **Scénario**

Vous prévoyez de créer et de coder un modèle MVC qui implémente votre plan pour les photos et les commentaires dans l'application de partage de photos Adventure Works. Le modèle doit stocker les données dans une base de données Microsoft Azure SQL et inclure des propriétés qui décrivent les photos, les commentaires et leur contenu. Le modèle doit permettre à l'application de stocker les photos téléchargées, de modifier leurs propriétés et de les supprimer en réponse aux demandes des utilisateurs.

### **Objectifs**

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

|  |  |
| --- | --- |
| • | Créez un nouveau projet ASP.NET MVC 5 dans Visual Studio 2017. |
| • | Ajoutez un nouveau modèle à l'application Web ASP.NET MVC 5 et ajoutez des propriétés au modèle. |
| • | Utilisez les annotations d'affichage et de modification des données dans le modèle MVC pour attribuer des attributs de propriété aux vues et aux contrôleurs. |
| • | Utilisez Visual Studio pour créer une nouvelle base de données Microsoft Azure SQL et vous connecter à la base de données. |
| • | Ajoutez du code Entity Framework aux classes de modèle dans le modèle MVC. |
| • | Utilisez les annotations d'affichage et de modification des données dans le modèle MVC pour attribuer des attributs de propriété aux vues et aux contrôleurs. |

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

Durée estimée: 60 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD03_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD03_LAK.md>.

### **Exercice 1: Création d'un projet MVC et ajout d'un modèle**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Créez une nouvelle application Web MVC 5 dans Visual Studio 2017. |
| • | Ajoutez des classes de modèle à l'application Web. |

### **Exercice 2: Ajout de propriétés aux modèles MVC**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Ajoutez des propriétés aux classes de modèles Photo et Comment. |
| • | Implémentez une relation entre les classes de modèle. |

### **Exercice 3: Utilisation des annotations de données dans les modèles MVC**

##### **Scénario**

Dans cet exercice, vous allez ajouter des annotations de données aux propriétés pour aider l'application Web MVC à les restituer dans des vues et à valider l'entrée utilisateur.

### **Exercice 4: Création d'une nouvelle base de données Microsoft Azure SQL**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Ajoutez le code Entity Framework à l'application de partage de photos en mode code d'abord. |
| • | Créez une nouvelle base de données SQL dans Microsoft Azure. |
| • | Utilisez la base de données SQL pour créer une chaîne de connexion dans l'application. |

### **Exercice 5: Test du modèle et de la base de données**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Ajoutez un contrôleur et des vues à l'application Web MVC. |
| • | Exécutez l'application Web. |

### **Question (s) de révision**

**Vérifiez vos connaissances**

**Découverte**

**Vous créez un site qui collecte des informations auprès des clients pour leurs comptes. Vous voulez vous assurer que les clients saisissent une adresse e-mail valide dans leE-mailbiens. Comment ferais-tu ceci?**

Afficher la solution Réinitialiser

**Vérifiez vos connaissances**

**Découverte**

**Il vous a été demandé de créer un site intranet qui publie une base de données clients, créée par le service commercial, à tous les employés de votre entreprise. Comment créeriez-vous le modèle avec Entity Framework?**

Afficher la solution Réinitialiser

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

Le modèle est le cœur d'une application Web MVC. En effet, les classes de modèle décrivent les informations et les objets gérés par votre application Web. Dans ce module, vous avez vu comment créer votre modèle, définir les relations entre les classes de modèle, décrire comment afficher, modifier et valider les propriétés et comment étendre les capacités de gestion des modèles MVC. Vous avez également vu comment lier des classes de modèle à des tables de base de données à l'aide d'Entity Framework et comment interroger un objet dans le modèle en écrivant du code LINQ.

**Meilleur entrainement:**Si vous souhaitez créer une nouvelle base de données pour une application Web et que vous préférez écrire du code qui décrit votre modèle, utilisez Entity Framework dans le flux de travail code-first pour créer votre modèle et ses classes.

**Meilleur entrainement:**Si vous souhaitez séparer la logique métier de la logique d'accès aux données, créez des classes de modèle et des classes de référentiel distinctes.

**Problèmes courants et conseils de dépannage**

| **Problème commun** | **Astuce de dépannage** |
| --- | --- |
| Le site Web ne peut pas se connecter ou créer une base de données. | Veuillez consulter le contenu du compagnon étudiant pour ce cours. |

### **Question (s) de révision**