Module 6: Test et débogage des applications Web 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/20486C06.html#P1) |
| **Leçon 1:** | [Test unitaire des composants MVC](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C06.html#P2) |
| **Leçon 2:** | [Mettre en œuvre une stratégie de gestion des exceptions](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C06.html#P3) |
| **Laboratoire:** | [Test et débogage des applications Web 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/20486C06.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/20486C06.html#P5) |

# **Aperçu du module**

Les systèmes logiciels tels que les applications Web sont complexes et nécessitent plusieurs composants, souvent écrits par différents développeurs, pour fonctionner ensemble. Des hypothèses incorrectes, une compréhension inexacte, des erreurs de codage et de nombreuses autres sources peuvent créer des bogues qui entraînent des exceptions ou un comportement inattendu. Pour améliorer la qualité de votre application Web et créer une expérience utilisateur satisfaisante, vous devez identifier les bogues de n'importe quelle source et les éliminer. Traditionnellement, la plupart des tests ont été effectués à la fin d'un projet de développement. Cependant, il est récemment devenu largement admis que les tests tout au long du cycle de vie du projet améliorent la qualité et garantissent qu'il n'y a pas de bogues dans les logiciels de production. Vous devez comprendre comment exécuter des tests sur de petits composants de votre application Web pour vous assurer qu'ils fonctionnent comme prévu avant de les assembler dans une application Web complète. Vous devez également savoir comment utiliser les outils disponibles dans Microsoft Visual Studio 2017 pour déboguer et suivre les exceptions lorsqu'elles surviennent.

### **Objectifs**

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

|  |  |
| --- | --- |
| • | Exécutez des tests unitaires sur les composants Model – View – Controller (MVC), tels que les classes de modèle, et localisez les bogues potentiels. |
| • | Créez une application MVC qui gère les exceptions de manière fluide et robuste. |

# Leçon 1: Test unitaire des composants MVC

Un test unitaire est une procédure qui instancie un composant que vous avez écrit, appelle un aspect des fonctionnalités du composant et vérifie que le composant répond correctement selon la conception. Dans la programmation orientée objet, les tests unitaires instancient généralement une classe et appellent l'une de ses méthodes. Dans une application Web MVC, les tests unitaires peuvent instancier des classes de modèle ou des contrôleurs et appeler leurs méthodes et actions. En tant que développeur Web, vous devez savoir comment créer un projet de test et ajouter des tests au projet de test. Les tests vérifient les aspects individuels de votre code sans s'appuyer sur des serveurs de base de données, des connexions réseau et d'autres infrastructures. En effet, les tests unitaires se concentrent sur le code que vous écrivez.

## **Objectifs de la leçon**

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

|  |  |
| --- | --- |
| • | Décrivez l'utilisation des tests unitaires pour améliorer la qualité du code que vous écrivez. |
| • | Énumérez les principes du développement piloté par les tests (TDD). |
| • | Décrivez comment les composants faiblement couplés augmentent la testabilité et la maintenabilité de votre application. |
| • | Écrivez des tests unitaires sur des classes de modèle, des contrôleurs et des vues dans une application Web MVC. |
| • | Décrivez comment les conteneurs d'inversion de contrôle (IoC) peuvent instancier des contrôleurs et des classes de modèle. |
| • | Exécutez des tests unitaires dans Visual Studio 2017. |
| • | Utilisez un outil de simulation pour automatiser la création de simulation dans les tests unitaires. |

## **Pourquoi effectuer des tests unitaires?**

Il existe trois types de tests que vous pouvez utiliser pour identifier les bogues dans votre application:

|  |  |
| --- | --- |
| • | Tests unitaires:Les tests unitaires vérifient les petits aspects de la fonctionnalité. Par exemple, un test unitaire peut vérifier le type de retour d'une seule méthode. En définissant de nombreux tests unitaires pour votre projet, vous pouvez vous assurer qu'ils couvrent tous les aspects fonctionnels de votre application Web. |
| • | Tests d'intégration:Les tests d'intégration vérifient comment deux composants ou plus fonctionnent ensemble. Ils peuvent être utilisés pour vérifier comment deux classes ou plus interagissent les unes avec les autres. Les tests d'intégration peuvent également être utilisés pour vérifier comment l'ensemble de l'application Web, y compris la base de données et les services Web externes, fonctionne pour fournir du contenu. |
| • | Tests d'acceptation:Les tests d'acceptation se concentrent sur une exigence fonctionnelle ou technique qui doit travailler pour que les parties prenantes acceptent la candidature. À l'instar des tests d'intégration, les tests d'acceptation testent généralement plusieurs composants fonctionnant ensemble. |

**Noter:**Les tests unitaires sont importants car ils peuvent être définis au début du développement. Les tests d'intégration et d'acceptation sont généralement exécutés plus tard, lorsque plusieurs composants sont en voie d'achèvement.

**Qu'est-ce qu'un test unitaire?**

Un test unitaire est une procédure qui vérifie un aspect spécifique de la fonctionnalité. Plusieurs tests unitaires peuvent être effectués pour une seule classe et même pour une seule méthode dans une classe. Par exemple, vous pouvez écrire un test qui vérifie que la méthode Validate renvoie toujours une valeur booléenne. Vous pouvez écrire un deuxième test qui vérifie que lorsque vous envoyez une chaîne à la méthode Validate, la méthode renvoie une valeur vraie. Vous pouvez assembler de nombreux tests unitaires en une seule classe appelée appareil de test. Souvent, un montage de test contient tous les tests d'une classe spécifique. Par exemple, ProductTestsFixture peut inclure tous les tests qui vérifient les méthodes de la classe Product.

Un test unitaire unique se compose généralement de code qui s'exécute en trois phases:

|  |  |
| --- | --- |
| 1. | Organiser. Dans cette phase, le test crée une instance de la classe qu'il testera. Il attribue également toutes les propriétés requises et crée tous les objets requis pour terminer le test. Seuls les propriétés et objets essentiels au test sont créés. |
| 2. | Acte. Dans cette phase, le test exécute la fonctionnalité qu'il doit vérifier. Habituellement, dans la phase Act, le test appelle une seule procédure et stocke le résultat dans une variable. |
| 3. | Affirmer. Dans cette phase, le test vérifie le résultat par rapport au résultat attendu. Si le résultat correspond à ce qui était attendu, le test réussit. Sinon, le test échoue. |

**Comment les tests unitaires aident-ils à diagnostiquer les bogues?**

Étant donné que les tests unitaires vérifient un aspect petit et spécifique du code, il est facile de diagnostiquer le problème lorsque les tests échouent. Les tests unitaires fonctionnent généralement avec une seule classe et isolent la classe des autres classes dans la mesure du possible. Si d'autres classes sont essentielles, le plus petit nombre de classes est créé dans la phase d'arrangement. Cette approche vous permet de résoudre rapidement les problèmes car le nombre de sources potentielles d'un bogue est faible.

Les tests unitaires doivent vérifier le code que vous écrivez et non les infrastructures sur lesquelles le système de production s'appuiera. Par exemple, les tests unitaires doivent s'exécuter sans se connecter à une base de données ou à un service Web réel, car des problèmes de réseau ou des pannes de service peuvent entraîner une panne. En utilisant cette approche, vous pouvez distinguer les bogues qui proviennent du code, qui doit être corrigé en modifiant le code, des bogues qui résultent de défaillances d'infrastructure, qui doivent être corrigés en changeant le matériel, en reconfigurant les serveurs Web, en reconfigurant les chaînes de connexion ou en effectuant d'autres configurations changements.

**Tests unitaires automatisés**

Il est important de comprendre que les tests unitaires, une fois définis, peuvent être réexécutés rapidement et facilement tout au long du cycle de vie du projet. En fait, Microsoft Visual Studio réexécute les tests automatiquement chaque fois que vous exécutez un projet avec débogage. Vous pouvez également lancer manuellement des tests à tout moment.

Ceci est important car un nouveau code peut provoquer des bogues à n'importe quelle étape du processus de développement. À titre d'exemple, considérons un projet qui se déroule en trois phases. Un test unitaire défini pour la phase 1 permet de mettre en évidence les problèmes des phases 2 et 3 qui pourraient autrement passer inaperçus:

|  |  |
| --- | --- |
| 1. | La phase 1. La classe ProductController est définie et le test unitaire Test\_Index\_Action est écrit. Le test vérifie que l'action Index fonctionne avec un paramètre entier. Lorsque vous appelez l'action Index avec un entier, l'action renvoie une collection qui comprend le même nombre d'objets Product. Le test réussit. |
| 2. | Phase 2. Un développeur modifie l'action Index afin qu'elle renvoie une vue partielle. Le développeur renomme l'action \_Index pour se conformer à la convention de dénomination de vue partielle de l'équipe. Le test échoue car le nom a changé. Le développeur modifie le test pour qu'il appelle l'action renommée et que le test réussisse. |
| 3. | Phase 3. Un développeur modifie l'action Index en écrivant une autre requête LINQ (Language Integrated Query) pour implémenter une nouvelle exigence fonctionnelle. Cependant, le développeur fait une erreur dans la syntaxe LINQ. L'action Index renvoie désormais zéro produit quel que soit l'entier passé en paramètre. Le test échoue. |

Au cours de la phase 2, l'échec du test est survenu parce que le nom de l'action avait changé et non à cause d'une erreur dans le code d'action. La solution consistait à modifier le nom de l'action appelée dans la méthode de test, mais cela peut rappeler aux développeurs de modifier les appels aux actions renommées dans toute la solution.

Dans la phase 3, l'échec du test est survenu parce que le développeur a mal compris LINQ ou a fait une erreur de frappe dans la syntaxe LINQ. Le test unitaire a mis en évidence l'erreur dès qu'elle s'est produite. Cela aide le développeur à se concentrer sur une solution pour résoudre l'échec du test et à s'assurer que la nouvelle fonctionnalité ne provoque pas d'échecs dans le code écrit plus tôt dans le projet.

**Question**: Les membres du conseil veulent que vous vous assuriez que votre application Web calcule correctement la taxe de vente sur chaque produit du catalogue. S'agit-il d'un exemple de test unitaire, d'un test d'intégration ou d'un test d'acceptation?

## **Principes du développement piloté par les tests**

Vous pouvez utiliser des tests unitaires dans n'importe quelle méthodologie de développement, y compris les projets en cascade, les projets itératifs et les projets Agile pour repérer les bogues potentiels et améliorer la qualité de l'application finale. Tout projet peut bénéficier de tests unitaires, quelle que soit la méthodologie de développement utilisée dans le projet. Cependant, une méthodologie de développement spécifique, appelée Test Driven Development (TDD), place les tests unitaires au centre des pratiques de travail. Le TDD est souvent décrit comme une méthodologie de développement distincte. Certains auteurs considèrent qu'il s'agit d'un principe fondamental de la méthodologie Extreme Programming.

**Ecrire le test, réussir le test, refactoriser**

Dans TDD, les développeurs répètent le cycle court suivant pour implémenter une exigence fonctionnelle:

|  |  |
| --- | --- |
| 1. | Écrivez le test. Le développeur commence par créer et coder le test unitaire. Cette étape nécessite une compréhension complète de l'exigence fonctionnelle, qui peut être obtenue à partir de cas d'utilisation ou de user stories. Étant donné que le développeur n'a écrit aucun code dans l'application, le test échoue. |
| 2. | Passer le test. Le développeur écrit du code simple et rapide dans l'application afin qu'il réussisse le test. Lors de la première itération, ce code est souvent inélégant et peut inclure de fausses hypothèses telles que des valeurs codées en dur. |
| 3. | Refactoriser. Le développeur nettoie le code de l'application, supprime le code en double, supprime les valeurs codées en dur, améliore la lisibilité et apporte d'autres améliorations techniques. Le développeur réexécute le test pour s'assurer que la refactorisation n'a pas provoqué d'échec. |

Le cycle se répète ensuite. À chaque itération, le développeur ajoute un petit nouvel élément de la fonctionnalité finale avec un test correspondant.

Il est important que le code de vos tests soit traité comme du code de production. Il doit être bien pensé et facile à lire afin que les autres développeurs puissent comprendre le test et diagnostiquer rapidement tout échec de test.

**Principes de développement piloté par les tests**

Le TDD est différent des approches traditionnelles de développement d'applications. Pour utiliser le TDD efficacement, vous devez comprendre ses principes fondamentaux:

|  |  |
| --- | --- |
| • | Écrire des tests avant le code. Dans le cycle de développement TDD, le test est créé à la première étape avant tout code dans l'application. Cela signifie que le test doit échouer la première fois qu'il est exécuté. Vous pouvez comprendre le test comme une spécification de la fonctionnalité que vous créez. En écrivant d'abord le test, vous vous assurez de commencer par une compréhension approfondie du problème que vous essayez de résoudre. |
| • | Déplacez-vous par petites étapes. En décomposant une grande application en petits éléments de fonctionnalité, vous pouvez améliorer la productivité des développeurs. Vous pouvez le faire en donnant à un développeur un petit problème à résoudre à chaque itération. Le développeur peut résoudre rapidement le problème simple et comprendre toutes les circonstances possibles dans lesquelles son code s'exécute. |
| • | N'écrivez que suffisamment de code pour réussir le test. À chaque itération, ne soyez pas tenté d'ajouter du code supplémentaire qui n'est pas lié au test. Par exemple, si vous savez que les utilisateurs appelleront une action avec d'autres paramètres que ceux du test, n'écrivez pas de code pour gérer ces paramètres. Au lieu de cela, lors de la prochaine itération, écrivez un deuxième test pour ces paramètres. Ensuite, écrivez et refactorisez ce code. |

Les développeurs peuvent se référer aux tests comme exemples d'utilisation d'une classe ou d'une méthode. Cela peut augmenter la productivité des développeurs, car les développeurs peuvent afficher le code qui illustre la méthode ou la classe.

**Noter:**Dans le laboratoire, vous utiliserez certains principes TDD lors du développement de la solution. Cependant, le laboratoire ne se conforme pas à une lecture stricte des méthodes TDD.

**Question:**Vous avez écrit un contrôleur pour la classe de modèle Comment. Vous écrivez un test unitaire qui vérifie que l'action Index renvoie une collection d'objets Comment. Vous êtes-vous conformé aux principes TDD?

## **Écriture de composants MVC faiblement couplés**

Une application faiblement couplée est une application dans laquelle chaque composant nécessite peu ou pas de détails sur les autres composants du système. Dans la programmation orientée objet, par exemple, deux classes peuvent être décrites comme faiblement couplées si une classe peut appeler des méthodes sur l'autre classe sans aucun code spécifique à l'autre classe. Lorsque les composants du système sont faiblement couplés de cette manière, il est facile de remplacer une classe par une autre implémentation de la même fonctionnalité. Les composants faiblement couplés sont essentiels pour des tests unitaires approfondis, car les classes qui traitent des données réelles, telles que les données d'une base de données, peuvent être facilement remplacées par des classes qui traitent des données de test.

**Utilisation du couplage lâche dans les tests**

Une application faiblement couplée est facile à tester car vous pouvez simplifier les tests en remplaçant une instance entièrement fonctionnelle d'une classe par une instance simplifiée spécialement conçue pour le test. Le remplacement des classes dans les tests de cette manière ne peut être effectué que lorsque les composants sont faiblement couplés. Les instances de remplacement utilisées pour les tests unitaires sont appelées tests doubles ou faux. Un test double ou faux comprend juste assez de code et de propriétés pour réussir le test pertinent et prouver la fonctionnalité.

**Autres avantages de l'accouplement lâche**

Un couplage lâche présente d'autres avantages que les tests. Le couplage lâche facilite le remplacement de composants simples par des composants plus sophistiqués. Par exemple, dans la première version d'une application, vous pouvez écrire une classe qui calcule une moyenne arithmétique simple. Dans la version deux, vous pouvez remplacer cette classe par une nouvelle classe qui calcule une moyenne pondérée basée sur un algorithme plus complexe. Si les composants sont faiblement couplés, vous pouvez effectuer ce remplacement sans modifier aucun code en dehors des classes de moyenne.

**Utilisation d'interfaces pour un couplage lâche**

Dans la programmation orientée objet, une interface définit un ensemble de propriétés et de méthodes. Toute classe qui implémente cette interface doit implémenter toutes les propriétés et méthodes qu'elle définit au minimum. Cela crée un couplage lâche car vous n'avez pas besoin de spécifier une classe dans le code. Au lieu de cela, vous pouvez spécifier n'importe quelle implémentation d'une interface particulière.

**Couplage lâche dans une application Web MVC**

Pour montrer comment un couplage lâche peut être utilisé dans une application Web MVC, envisagez le scénario suivant:

Vous écrivez une application Web MVC qui affiche un catalogue de produits et permet aux utilisateurs de cliquer sur un produit pour afficher tous les détails. Sur la page des détails du produit, les utilisateurs peuvent afficher et ajouter des avis sur ce produit. Lors du test, vous voulez vous assurer que la propriété Product.Reviews, qui est une collection d'objets Review, n'inclut que les avis avec la bonne valeur ProductID.

Rappelez-vous que les tests unitaires ne doivent pas s'appuyer sur une infrastructure telle que des connexions réseau ou des serveurs de base de données, mais uniquement sur du code de test. En revanche, en production, l'application obtiendra tous les avis produits à partir d'une base de données. Pour répondre aux besoins de test et de production, vous pouvez créer les composants suivants:

|  |  |
| --- | --- |
| • | Une interface IProduct. Cela inclut une définition de la propriété Reviews, ainsi que d'autres propriétés et méthodes. Cette interface est connue sous le nom de référentiel. |
| • | Une classe de produit. Il s'agit de l'implémentation du référentiel IProduct que l'application utilise en production. Lorsque l'utilisateur appelle la propriété Avis, tous les avis du produit actuel sont obtenus à partir de la base de données. |
| • | Une classe TestProduct. Il s'agit de l'implémentation double ou fausse de test du référentiel IProduct utilisé par le test. Le test configure un objet TestProduct et de faux avis avec différentes valeurs ProductID. Le test appelle la propriété TestProduct.Reviews et vérifie que seules les bonnes critiques sont renvoyées. |

Notez que, dans le test, le TestProduct double utilise des données en mémoire configurées pendant la phase d'arrangement. Il n'interroge pas la base de données. Par conséquent, vous pouvez tester votre code sans vous fier à la connexion réseau à la base de données ou au serveur de base de données lui-même. Cette approche garantit également que le test s'exécute rapidement, ce qui est important car les tests lents découragent les développeurs d'exécuter des tests régulièrement.

**Question:**Dans un test, vous créez une fausse collection d'objets BlogEntry. S'agit-il d'un exemple d'interface ou d'un double test?

## **Ecriture de tests unitaires pour les composants MVC**

Le modèle de programmation Microsoft ASP.NET MVC 5 est facile à intégrer avec les principes des tests unitaires et du TDD en raison de sa séparation des préoccupations en modèle, contrôleurs et vues. Les modèles sont simples à tester car ce sont des classes indépendantes que vous pouvez instancier et configurer pendant la phase d'arrangement d'un test. Les contrôleurs sont des classes simples que vous pouvez tester, mais il est complexe de tester des contrôleurs avec des données en mémoire, plutôt que d'utiliser une base de données. Pour tester les contrôleurs avec des données en mémoire, vous créez une double classe de test, également appelée faux référentiel. Les objets de cette classe peuvent être remplis avec des données de test en mémoire sans interroger une base de données. Vous devez comprendre comment écrire des doubles de test et comment écrire un test typique pour les classes MVC.

**Ajouter et configurer un projet de test**

Dans Microsoft Visual Studio 2017, vous pouvez tester un projet d'application Web MVC en ajoutant un nouveau projet à la solution, basé sur le modèle de projet de test unitaire. Vous devez ajouter des références au projet de test afin que le code de test puisse accéder aux classes dans le projet d'application Web MVC. Vous aurez également besoin d'une référence à System.Web.Mvc et à d'autres espaces de noms de Microsoft .NET Framework.

Dans un projet de test Visual Studio, les fixtures de test sont des classes marquées de l'annotation [TestClass]. Les tests unitaires sont des méthodes marquées de l'annotation [TestMethod]. Les tests unitaires renvoient généralement void mais appellent une méthode de la classe Assert, telle que Assert.AreEqual () pour vérifier les résultats dans la phase de test Assert.

**Noter:**Il existe de nombreux autres frameworks de test disponibles pour les applications Web MVC et vous pouvez choisir celui que vous préférez. Beaucoup de ces frameworks peuvent être ajoutés à l'aide du gestionnaire de packages NuGet disponible dans Visual Studio.

**Classes de modèles de test et logique métier**

Dans MVC, les classes de modèle ne dépendent pas d'autres composants ou de l'infrastructure. Vous pouvez facilement les instancier en mémoire, organiser leurs valeurs de propriété, agir sur elles en appelant une méthode et affirmer que le résultat était comme prévu. Parfois, vous créez une logique métier dans les classes de modèle, auquel cas la phase Act implique l'appel d'une méthode sur la classe de modèle elle-même. Si, en revanche, vous avez créé une couche de logique métier distincte, le code de la phase Act doit appeler une méthode sur la classe d'objet métier et transmettre une classe de modèle.

**Test d'une classe de modèle**

[Méthode d'essai]

public void Test\_Product\_Reviews ()

{

// Arranger la phase

Test de produitProduit = nouveau produit ();

testProduct.Type = "CompleteBike";

// Phase d'action

var result = testProduct.GetAccessories ();

// Phase d'affirmation

Assert.AreEqual (typeof (IEnumerable (BikeAccessory)), result.GetType ());

}

**Créer des interfaces de référentiel**

Un référentiel est une interface qui définit les propriétés et les méthodes que MVC peut utiliser pour stocker des données dans une application Web. Il est courant de créer une interface de référentiel unique pour l'ensemble de votre application Web.

**Une interface de référentiel**

interface publique IWebStoreContext

{

Produits <Product> IQueryable {get; }

T Ajoutez <T> (entité T) où T: classe;

Product FindProductById (ID int);

T Supprimer <T> (entité T) où T: classe;

}

**Implémenter et utiliser un référentiel dans l'application**

L'interface du référentiel définit les méthodes d'interaction avec les données mais ne fixe pas la manière dont ces données seront définies et stockées. Vous devez fournir deux implémentations du référentiel:

|  |  |
| --- | --- |
| • | Une implémentation du référentiel à utiliser dans la production et le développement. Cette implémentation utilisera les données de la base de données ou d'un autre mécanisme de stockage. |
| • | Une implémentation du référentiel à utiliser dans les tests. Cette implémentation utilisera les données de la mémoire définie pendant la phase d'arrangement de chaque test. |

**Implémentation d'un référentiel dans le projet MVC**

classe publique WebStoreContext: DbContext, IWebStoreContext

{

public DbSet <Product> Products {get; ensemble; }

IQueryable <Product> IWebStoreContext.Products {get {return Photos; }}

T IWebStoreContext.Add <T> (entité T)

{

return Set <T> () .Add (entité);

}

Produit IWebStoreContext.FindProductById (ID int)

{

return Set <Produit> () .Find (ID);

}

T IWebStoreContext.Delete <T> (entité T)

{

return Set <T> () .Remove (entité);

}

}

Notez que la classe WebStoreContext hérite toujours de la classe Entity Framework DbContext mais implémente également l'interface IWebStoreContext que vous venez de créer. En fait, les méthodes d'interface telles que Add (), Delete () et FindProductById () encapsulent simplement les méthodes de la classe DbContext telles que Remove ().

**Mettre en œuvre un double test de référentiel**

La deuxième implémentation de l'interface du référentiel est l'implémentation que vous utiliserez dans les tests unitaires. Cette implémentation utilise des données en mémoire et une collection d'objets à clé pour fonctionner comme un contexte Entity Framework, mais évite de travailler avec une base de données.

**Implémentation d'un faux référentiel**

classe FakeWebStoreContext: IWebStoreContext

{

// Cet objet est une collection à clé que nous utilisons pour simuler un

// contexte du cadre d'entité en mémoire

SetMap \_map = nouveau SetMap ();

Produits publics IQueryable <Product>

{

get {return \_map.Get <Produit> () .AsQueryable (); }

set {\_map.Use <Produit> (valeur); }

}

public T Ajouter <T> (entité T) où T: classe

{

\_map.Get <T> () .Add (entité);

entité de retour;

}

public Product FindProductById (ID int)

{

Article de produit = (à partir de p dans ce produit.

où p.ProductID == ID

sélectionnez p) .Premier ();

retourner l'objet;

}

public T Delete <T> (entité T) où T: classe

{

\_map.Get <T> () .Remove (entité);

entité de retour;

}

class SetMap: KeyedCollection <Type, objet>

{

public HashSet <T> Utiliser <T> (IEnumerable <T> sourceData)

{

var set = new HashSet <T> (sourceData);

if (Contient (typeof (T)))

{

Supprimer (typeof (T));

}

Ajouter (définir);

jeu de retour;

}

public HashSet <T> Get <T> () {return (HashSet <T>) this [typeof (T)]; }

protected override Type GetKeyForItem (élément d'objet)

{

return item.GetType (). GetGenericArguments (). Single ();

}

}

}

**Utilisation du test double pour tester un contrôleur**

Après avoir implémenté une double classe de test pour le référentiel, vous pouvez l'utiliser pour tester un contrôleur dans un test unitaire. Dans la phase d'arrangement du test, créez un nouvel objet à partir de la double classe de test et transmettez-le au constructeur du contrôleur. Le contrôleur utilise cet objet pour son contexte Entity Framework pendant la phase de test Act. Cela vous permet de vérifier les résultats.

**Utilisation d'un test double pour tester un contrôleur**

[Méthode d'essai]

public void Test\_Index\_Model\_Type ()

{

var context = new FakeWebStoreContext ();

context.Products = nouveau [] {

nouveau produit(),

nouveau produit (),

nouveau produit ()

} .AsQueryable ();

var controller = new ProductController (contexte);

var result = controller.Index () as ViewResult;

Assert.AreEqual (typeof (List <Product>), result.Model.GetType ());

}

Notez que le test crée une nouvelle instance de la classe FakeWebStoreContext et y ajoute trois objets Product. Dans ce cas, vous pouvez exécuter le test sans définir de propriétés sur les objets Product. Ce contexte est transmis au constructeur ProductController et le test passe aux phases Act et Assert.

**Question:**Dans quel but utiliseriez-vous un faux référentiel lorsque vous écrivez des tests unitaires sur un contrôleur MVC?

## **Spécification du contexte correct**

Lorsque vous utilisez des doubles de test pour les tests unitaires des contrôleurs MVC, vous devez vous assurer que le contrôleur utilise le contexte correct dans deux scénarios différents: les tests unitaires et à tous les autres moments. Si vous utilisez le code de test double production, par exemple, aucune donnée ne sera stockée dans la base de données. Si vous utilisez le contexte Entity Framework réel pour les tests unitaires, tous les tests échoueront. Le tableau suivant donne plus de détails.

| **Scénario** | **Le contexte** |
| --- | --- |
| Dans les tests unitaires | Dans un test unitaire, vous devez tester le code sans vous fier à l'infrastructure sous-jacente telle que les bases de données. Vous pouvez utiliser une double classe de test qui se comporte comme un contexte Entity Framework, mais utilise des données en mémoire. |
| D'autres fois | Lorsque vous ne testez pas le code unitaire, vous utilisez un contexte Entity Framework réel dans votre code de contrôleur, qui interagit avec une base de données. |

**Utilisation de constructeurs pour spécifier un référentiel**

Vous pouvez spécifier le référentiel correct à utiliser dans un test à l'aide de constructeurs de contrôleurs. Lors d'un test unitaire, appelez un constructeur de contrôleur qui prend une instance de l'interface du référentiel comme paramètre. À d'autres moments, l'usine de contrôleurs MVC appellera le constructeur de contrôleurs sans aucun paramètre.

**Utilisation de constructeurs pour spécifier des référentiels**

public class ProductController: Controller

{

contexte IWebStoreContext privé;

// La version sans paramètre du constructeur est utilisée par la fabrique de contrôleurs MVC

public ProductController ()

{

// Instancier un contexte Entity Framework réel

context = nouveau WebStoreContext ();

}

// Ce constructeur est utilisé par les tests unitaires. Ils passent un test double contexte

public ProductController (contexte IWebStoreContext)

{

// Utilise le contexte passé au constructeur

context = Contexte;

}

// Ajoutez des méthodes d'action ici

}

**Utilisation de l'inversion des conteneurs de contrôle**

Vous pouvez également vous assurer que la classe de référentiel correcte est utilisée dans chaque contexte à l'aide d'un conteneur d'inversion de contrôle (IoC). Un conteneur IoC est un framework que vous pouvez ajouter à votre application qui instancie des classes. Vous devez configurer le conteneur IoC afin qu'il puisse identifier le type d'objet à créer lorsque l'application attend une interface spécifique. Par exemple, lorsque du code est utilisé pour créer une nouvelle instance de l'interface IWebStoreContext, le conteneur IoC vérifie sa configuration et constate qu'il doit instancier la classe WebStoreContext, qui est un contexte Entity Framework qui implémente l'interface IWebStoreContext.

**Noter:**Les frameworks IoC sont également appelés frameworks d'injection de dépendances (DI) car ils injectent des classes qui implémentent les interfaces correctes dans les constructeurs de classes.

En utilisant un conteneur IoC, vous évitez d'avoir à créer deux constructeurs pour les classes de contrôleur. Pendant les tests, vous pouvez instancier le test double et le transmettre au constructeur du contrôleur. À d'autres moments, le conteneur IoC instancie le contexte Entity Framework réel et le transmet au constructeur du contrôleur.

**Noter:**Il existe de nombreux conteneurs IoC conçus pour être utilisés avec les applications Web ASP.NET MVC. Ceux-ci remplacent certains composants du framework MVC pour prendre en charge l'IoC. Par exemple, ils remplacent la fabrique de contrôleurs par défaut par une fabrique qui instancie les contrôleurs avec la classe de contexte appropriée. De nombreux conteneurs IoC sont disponibles dans la bibliothèque de packages NuGet, tels que NinJect et StructureMap.

**Question:**Quelle approche est plus faiblement couplée: utiliser des constructeurs pour spécifier le contexte ou utiliser des conteneurs IoC pour spécifier le contexte?

## **Démonstration: comment exécuter des tests unitaires**

Dans cette démonstration, vous verrez comment ajouter un nouveau projet de test à votre solution ASP.NET MVC 5 et comment créer et exécuter un test unitaire simple sur le contrôleur de page d'accueil.

### **Étapes de démonstration**

|  |  |
| --- | --- |
| • | Vous trouverez les étapes dans la section «Leçon 1: Test unitaire des composants MVC» sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD06_DEMO.md>. |

## **Utilisation de cadres simulés**

Lorsque vous écrivez un test unitaire, vous devez, pendant la phase d'arrangement, créer des données de test sur lesquelles le test sera exécuté. En utilisant un objet de test double ou fictif pour fournir des données de test, vous pouvez tester votre code indépendamment des autres classes et des éléments d'infrastructure tels que les bases de données et les connexions réseau sur lesquelles l'application s'appuiera. Vous pouvez créer des doubles de test en codant manuellement leur instanciation, en définissant leurs propriétés et en remplissant des collections. Ces doubles de test sont appelés objets de simulation manuels.

Alternativement, au lieu de créer manuellement des objets fictifs avec votre propre code, vous pouvez utiliser un framework de simulation pour automatiser ce travail. Un cadre de simulation crée automatiquement un objet simulé à l'aide des interfaces que vous spécifiez. Dans le cas des conteneurs IoC, il existe de nombreux frameworks simulés qui peuvent être utilisés pour tester les applications Web MVC. Vous pouvez trouver de nombreux frameworks moqueurs à l'aide du gestionnaire de packages NuGet.

**Noter:**Les frameworks moqueurs pour ASP.NET MVC 5 incluent Moq et NSubstitute. Ces frameworks sont disponibles dans NuGet et il existe des alternatives. Choisissez le cadre qui répond le mieux à vos besoins de test.

**Pourquoi utiliser un framework moqueur?**

Il existe de nombreuses situations dans lesquelles les frameworks moqueurs peuvent considérablement faciliter les tests unitaires. Même pour les tests simples, les frameworks moqueurs réduisent la quantité de code d'installation que vous devez développer. Une fois que vous vous êtes familiarisé avec le cadre de simulation que vous choisissez et que vous aurez appris à écrire le code d'arrangement pour celui-ci, vous commencerez à gagner du temps. Dans des situations plus complexes, telles que les suivantes, les frameworks moqueurs présentent de grands avantages:

|  |  |
| --- | --- |
| • | Création de plusieurs objets fictifs d'un même type. Vous devriez essayer de garder chaque test unitaire aussi simple que possible, mais inévitablement, certains tests nécessitent de nombreux objets fictifs pour fonctionner. Par exemple, si vous testez une méthode qui sous-tend la fonctionnalité de pagination dans votre application Web, vous devez créer suffisamment d'objets fictifs pour remplir plusieurs pages. Les frameworks moqueurs peuvent automatiser la création de plusieurs objets fictifs du même type. |
| • | Objets moqueurs avec plusieurs interfaces. Dans certains tests, où il existe plusieurs dépendances entre des classes de types différents, vous devez créer de nombreux objets fictifs de différentes classes. Dans de telles situations, il est fastidieux de coder manuellement de nombreux objets fictifs. Les frameworks moqueurs peuvent aider en générant automatiquement les objets à partir des interfaces que vous spécifiez. |

Dans chaque test unitaire, vous vous intéressez à un petit domaine spécifique de fonctionnalités. Certaines propriétés et méthodes d'une interface seront cruciales pour votre test, tandis que d'autres ne seront pas pertinentes. Un cadre de simulation vous permet de spécifier des propriétés et des méthodes non pertinentes dans un test donné. Lorsque le framework crée un objet fictif pour votre test, il crée des stubs pour les propriétés et méthodes non pertinentes. Un stub est une implémentation simple avec peu de code. De cette façon, le cadre de simulation vous évite d'avoir à implémenter laborieusement toutes les exigences de l'interface.

**Question:**Quelle est la différence entre un conteneur IoC et un framework de simulation?

# Leçon 2: Mettre en œuvre une stratégie de gestion des exceptions

Des événements inattendus sont susceptibles de se produire de temps en temps dans tout système complexe, y compris les applications Web MVC. Parfois, de tels événements d'exécution inattendus provoquent une erreur. Lorsque cela se produit, ASP.NET ou .NET Framework génère une exception, qui est un objet que vous pouvez utiliser pour diagnostiquer et résoudre l'erreur. L'exception contient des informations que vous pouvez utiliser pour diagnostiquer le problème. Les exceptions qui ne sont pas gérées dans votre code entraîneront l'arrêt de l'application Web et l'affichage d'un message d'erreur à l'utilisateur. En tant que développeur Web, vous devez savoir comment détecter, gérer et lever des exceptions, et identifier la cause du problème. Visual Studio fournit une large gamme d'outils pour déboguer les exceptions et améliorer la robustesse de votre code. Vous pouvez également consigner les exceptions dans les bases de données et autres magasins, et utiliser le.

## **Objectifs de la leçon**

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

|  |  |
| --- | --- |
| • | Expliquez comment déclencher et intercepter des exceptions. |
| • | Expliquez comment configurer la gestion des exceptions à l'aide de Web.config. |
| • | Décrivez comment consigner les exceptions. |

## **Lever et attraper des exceptions**

Une erreur est un événement d'exécution inattendu qui empêche une application de terminer une opération. Lorsqu'une ligne de code provoque une erreur, ASP.NET ou le Common Language Runtime crée une exception. Cette exception est un objet d'une classe qui hérite de la classe de base System.Exception. Il existe de nombreuses classes d'exception. Souvent, la classe d'objets identifie ce qui n'a pas fonctionné. Par exemple, s'il existe une ArgumentNullException, cela indique qu'une valeur Null a été envoyée à une méthode qui n'accepte pas une valeur Null comme argument.

**Exceptions non gérées**

Lorsqu'une exception n'est pas explicitement gérée par une application, l'application s'arrête et l'utilisateur voit un message d'erreur. Dans les applications ASP.NET MVC, ce message d'erreur se présente sous la forme d'une page Web. Vous pouvez remplacer les pages d'erreur par défaut d'ASP.NET pour afficher vos propres informations d'erreur aux utilisateurs.

Si une exception non gérée se produit lors du débogage de l'application dans Visual Studio, l'exécution s'interrompt sur la ligne qui a généré l'exception. Vous pouvez utiliser les outils de débogage de Visual Studio pour rechercher ce qui n'a pas fonctionné, isoler le problème et déboguer votre code.

Parfois, vous pouvez également soulever vos propres exceptions comme forme de communication entre les parties de votre application. Par exemple, considérons une application qui affiche des produits sur un site Web. Dans le référentiel, vous implémentez une méthode FindProduct qui renvoie le produit avec un nom correspondant à la chaîne de recherche transmise en tant qu'attribut. Si la méthode FindProduct ne localise pas un produit avec le nom spécifié, vous souhaiterez peut-être lever une exception. Cette approche permet au code du contrôleur, qui appelle la méthode FindProduct, de gérer l'événement selon lequel aucun produit n'est trouvé. Vous pouvez utiliser le mot clé throw pour générer des erreurs dans le code C #.

**Attraper des erreurs avec les blocs Try / Catch**

Le moyen le plus courant pour intercepter une exception, qui fonctionne dans n'importe quel code .NET Framework, consiste à utiliser le bloc try / catch. Le code du bloc try est exécuté. Si l'un de ces codes génère une exception, le type d'exception est vérifié par rapport au type déclaré dans le bloc catch. Si le type correspond ou est d'un type dérivé du type déclaré dans le bloc catch, le code du bloc catch s'exécute. Vous pouvez utiliser le code du bloc catch pour obtenir des informations sur ce qui n'a pas fonctionné et résoudre la condition d'erreur.

**Un simple bloc Try / Catch**

essayer

{

Produit produit = FindProductFromComment (commentaire);

}

catch (ArgumentNullException ex)

{

Console.WriteLine ("L'objet de commentaire était nul. Message d'erreur:" + ex.Message);

}

Dans les contrôleurs MVC, il existe deux méthodes pour intercepter les exceptions: la méthode OnException et l'annotation [HandleError].

**Attraper les exceptions de contrôleur dans OnException**

La méthode OnException est définie sur la classe Controller de base, vous pouvez donc la remplacer dans n'importe quel contrôleur MVC. Lorsque vous substituez la méthode OnException dans un contrôleur personnalisé, MVC exécute la méthode chaque fois qu'une exception se produit et n'est pas gérée dans un bloc try / catch. En utilisant cette approche, vous pouvez gérer les erreurs n'importe où dans votre contrôleur sans ajouter de nombreux blocs try / catch dans tout le code.

Dans la méthode OnException, vous pouvez intercepter des types spécifiques d'exceptions en vérifiant la propriété ExceptionContext.Exception. Votre code doit définir la propriété context.Result pour afficher une vue à l'utilisateur. Si vous ne définissez pas cette propriété, le navigateur affiche une page Web vierge à l'utilisateur.

**Utilisation d'OnException pour détecter les erreurs du contrôleur**

protected override void OnException (contexte ExceptionContext)

{

// Attraper une exception d'opération non valide

if (filterContext.Exception est InvalidOperationException)

{

var model = new HandleErrorInfo (context.Exception, controllerName, actionName);

var result = new ViewResult

{

ViewName = "Erreur",

ViewData = new ViewDataDictionary <HandleErrorInfo> (modèle),

// Enregistrer passe les données temporaires actuelles à la vue Erreur, car elle contient souvent

// informations de diagnostic

TempData = context.Controller.TempData

};

context.Result = résultat;

}

}

**Attraper des exceptions de contrôleur avec HandleError**

En utilisant l'annotation [HandleError], vous pouvez utiliser un attribut sur des méthodes d'action individuelles ou sur la classe de contrôleur elle-même, auquel cas l'annotation [HandleError] détecte les erreurs dans n'importe quelle action. Lorsque vous utilisez l'annotation [HandleError] sans propriétés, toutes les exceptions sont interceptées et envoyées à une vue MVC appelée Error.cshtml. Cette vue d'erreur par défaut doit être créée dans le dossier / View / Shared.

Vous pouvez utiliser deux propriétés pour contrôler le comportement de [HandleError]:

|  |  |
| --- | --- |
| • | Type d'exception. Par défaut, l'annotation [HandleError] intercepte toutes les exceptions. Si vous définissez un type plus spécifique avec la propriété ExceptionType, seules les exceptions de ce type spécifique seront interceptées. |
| • | Vue. Par défaut, l'annotation [HandleError] redirige les utilisateurs vers la vue Error.cshtml. Vous pouvez spécifier un autre fichier de vue à l'aide de la propriété View. En définissant les propriétés ExceptionType et View, vous pouvez transférer des exceptions de types spécifiques vers des vues conçues spécifiquement pour les afficher. |

**Utilisation de l'annotation HandleError**

[HandleError (ExceptionType = typeof (NotImplementedException), View = "NotImplemented")]

[HandleError]

Index ActionResult public ()

{

// Placez le code d'action ici

}

**Question:**Vous utilisez l'annotation [HandleError] pour intercepter les exceptions dans votre application Web. Cependant, vous vous rendez compte que [HandleError] intercepte toutes les exceptions, ce qui rend difficile pour vous d'isoler un problème spécifique. Que pouvez-vous faire pour restreindre les exceptions que l'annotation [HandleError] détecte?

## **Configuration de la gestion des exceptions**

Les applications ASP.NET affichent une page d'erreur par défaut aux utilisateurs lorsqu'une exception non gérée se produit dans votre application Web MVC. Il existe deux raisons pour lesquelles vous souhaiterez peut-être modifier la page d'erreur par défaut:

|  |  |
| --- | --- |
| • | Vous souhaitez afficher une page d'erreur de marque et de style qui a la même mise en page que les autres pages de l'application et inclut le logo de l'entreprise. |
| • | Vous voulez éviter d'attirer l'attention sur la technologie qui alimente le site. En informant les utilisateurs malveillants que votre site exécute ASP.NET MVC, vous pouvez les encourager à l'attaquer en utilisant des méthodes qui ont fonctionné dans le passé. |

Pour configurer une page d'erreur personnalisée pour votre application, utilisez le fichier Web.config de niveau supérieur. La page d'erreur personnalisée sera affichée lorsqu'une exception n'est pas gérée par un bloc try / catch ou un attribut [HandleError].

**Modes d'erreur personnalisés**

Pour configurer des erreurs personnalisées, ajoutez l'élément <customErrors> au fichier Web.config. Cet élément doit être un enfant de l'élément <system.web>. Utilisez les attributs suivants:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| • | **mode:**L'attribut mode peut être défini sur l'une des trois valeurs suivantes:   |  |  | | --- | --- | | o | **Désactivé:**Les erreurs non gérées sont affichées sur la page d'erreur ASP.NET par défaut. | | o | **Sur:**Les erreurs non gérées sont affichées dans la page que vous spécifiez dans l'attribut defaultRedirect. | | o | **RemoteOnly:**Dans ce mode, les erreurs non gérées s'affichent différemment pour les navigateurs sur les ordinateurs distants et les navigateurs sur le serveur Web. Dans un site de production, tous les navigateurs sont distants et les erreurs s'affichent sur la page que vous spécifiez dans l'attribut defaultRedirect. Dans les situations de développement, le développeur navigue souvent sur le site sur l'ordinateur local. Lorsqu'une erreur se produit, le développeur est dirigé vers la page d'erreur ASP.NET par défaut, qui comprend des informations de débogage telles que la trace de la pile. Utilisez le mode RemoteOnly sur les ordinateurs de développement. | |
| • | **defaultRedirect:**Cet attribut définit le nom de la vue qui sera utilisée pour afficher les exceptions non gérées. Lorsque vous créez cette vue, placez-la dans le dossier Vues / Partagé. |

**Configuration des erreurs personnalisées**

<system.web>

<customErrors mode = "On" defaultRedirect = "CustomError" />

</system.web>

**Pages d'erreur HTTP**

Certaines erreurs au niveau du protocole de transfert hypertexte (HTTP) ne peuvent pas être interceptées par les vues d'erreurs MVC personnalisées comme spécifié dans Web.config ou l'annotation [HandleError]. Pour de telles erreurs, vous pouvez utiliser l'élément <error> dans Web.config, en tant qu'élément enfant de l'élément <customErrors>.

**Récupération des codes d'erreur HTTP**

<customErrors mode = "On" defaultRedirect = "CustomError">

<error statusCode = "500" redirect = "~ / Error / Error500.html" />

</customErrors>

**Question:**Vous avez désactivé les erreurs personnalisées dans le fichier Web.config. Lorsque vous exécutez votre application sans débogage, une exception est levée dans un bloc try / catch. Quelle page affiche l'exception à l'utilisateur?

## **Journalisation des exceptions**

Les exceptions auxquelles vous êtes confronté pendant le développement peuvent être étudiées et déboguées à l'aide des outils de débogage de Visual Studio. Dans une situation idéale, aucune exception ne se produirait lorsque votre application Web est terminée et déployée pour les utilisateurs sur Internet. Cependant, dans le monde réel, des circonstances imprévues surviennent, entraînant des exceptions. Par exemple, les pannes de base de données, les problèmes de réseau et les erreurs de configuration dans n'importe quelle partie du système peuvent provoquer des exceptions.

Vous avez déjà vu comment présenter une page d'erreur personnalisée de marque et informative aux utilisateurs lorsque des exceptions se produisent. Il est également approprié de consigner les exceptions dans une application Web de production, afin que les administrateurs et les développeurs puissent évaluer l'étendue du problème, supprimer la cause et améliorer la robustesse du code.

**Écriture du code de journalisation des erreurs**

Avant d'écrire du code pour consigner les exceptions dans votre application Web, déterminez où les exceptions doivent être enregistrées. En règle générale, les applications Web consignent les exceptions dans un fichier XML (Extensible Markup Language). Cette approche peut être efficace, en particulier si le fichier texte est conservé sur un lecteur distinct de la base de données de l'application Web et d'autres fichiers critiques fréquemment consultés. Cependant, la taille des fichiers XML peut devenir volumineuse, ce qui les rend difficiles à analyser.

Une approche courante consiste à consigner les erreurs dans les tables d'une base de données. Ces tables peuvent faire partie d'une seule base de données d'application Web, ou vous pouvez les placer dans une base de données dédiée à l'écart des catalogues de produits, des profils utilisateur et d'autres informations. En journalisant les exceptions dans une base de données, vous créez une liste d'exceptions qui peuvent être facilement recherchées et analysées. Vous pouvez également créer une page Web pour présenter ces exceptions enregistrées aux administrateurs afin qu'ils puissent accéder aux détails à partir d'un navigateur.

Vous devez également envisager d'envoyer des e-mails ou d'autres types de messages aux administrateurs et aux développeurs lorsque des exceptions se produisent. Cette approche garantit que les administrateurs et les développeurs sont informés très rapidement d'une erreur. Il ne les oblige pas à examiner les exceptions enregistrées dans la base de données. Souvent, le code de gestion des erreurs dans les applications en direct se connecte à la fois à une base de données et envoie des messages électroniques aux administrateurs et aux développeurs.

**Où écrire le code de journalisation des erreurs**

Lorsque vous décidez où écrire le code qui enregistre les erreurs, vous devez tenir compte du fait que des erreurs peuvent survenir dans presque toutes les parties de votre application. Vous devez choisir une approche qui vous permet d'écrire une fois le code de journalisation des erreurs qui s'exécutera pour toute exception n'importe où dans votre application.

Par exemple, il n'est pas approprié d'écrire du code de journalisation des erreurs dans des blocs try / catch individuels. Si vous faites cela, vous devrez créer un bloc try / catch pour chaque procédure de votre application et écrire un code de journalisation en double dans le bloc catch.

Une approche plus efficace est la suivante:

|  |  |
| --- | --- |
| • | Créez une classe de contrôleur de base personnalisée pour votre application Web. Cette classe hérite de la classe de base System.Web.Mvc.Controller. |
| • | Dans votre classe de contrôleur de base personnalisée, remplacez la méthode OnException (). Placez votre code de journalisation des erreurs dans cette méthode. |
| • | Lorsque vous créez des contrôleurs, héritez de votre classe de contrôleur de base personnalisée au lieu de System.Web.Mvc.Controller. |

De cette façon, vous ne pouvez écrire le code de journalisation des erreurs qu'une seule fois dans la méthode OnExcpetion () remplacée. Cela intercepte les exceptions de l'un des contrôleurs héritiers.

**Utilisation d'outils de journalisation tiers**

Étant donné que la journalisation des exceptions est une exigence fonctionnelle très courante pour les applications Web, il existe de nombreuses solutions tierces que vous pouvez choisir si vous ne souhaitez pas écrire votre propre code de journalisation. Beaucoup d'entre eux sont disponibles dans Visual Studio à partir du gestionnaire de packages NuGet.

**Question:**Vous voulez vous assurer que les développeurs peuvent examiner les détails des exceptions qui surviennent lors du débogage. Comment devriez-vous aborder le code de journalisation des erreurs?

# **Atelier: Test et débogage des applications Web ASP.NET MVC 5**

### **Scénario**

L'application de partage de photos en est aux premiers stades de développement. Cependant, des erreurs fréquentes entravent la productivité de l'équipe de développement. Le développeur senior vous conseille d'intercepter les exceptions et autres défauts le plus tôt possible. Vous avez été invité à effectuer des tests unitaires du PhotoController pour vous assurer que tous les scénarios fonctionnent comme prévu et pour éviter des problèmes plus tard dans le cycle de vie de développement de l'application Web. Il vous a également été demandé de vous assurer qu'en cas d'erreurs critiques, les développeurs puissent obtenir des informations techniques utiles.

### **Objectifs**

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

|  |  |
| --- | --- |
| • | Effectuez des tests unitaires des composants d'une application Web MVC. |
| • | Configurez une stratégie de gestion des exceptions pour une application Web MVC. |

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

Durée estimée: 80 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD06_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD06_LAK.md>.

### **Exercice 1: Réalisation de tests unitaires**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| • | Créez un projet de test et écrivez les tests suivants.   |  |  | | --- | --- | | o | **Test\_Index\_Return\_View**: Ce test vérifie que l'action Index renvoie une vue nommée Index. | | o | **Test\_PhotoGallery\_Model\_Type**: Ce test vérifie que l'action \_PhotoGallery transmet une liste énumérable d'objets Photo à la vue partielle \_PhotoGallery. | | o | **Test\_GetImage\_Return\_Type**: Ce test vérifie que l'action GetImage renvoie un fichier et non une vue. | | o | **Test\_PhotoGallery\_No\_Parameter**: Ce test vérifie que lorsque vous appelez l'action \_PhotoGallery sans aucun paramètre, l'action transmet toutes les photos du contexte à la vue partielle \_PhotoGallery. | | o | **Test\_PhotoGallery\_Int\_Parameter**: Ce test vérifie que lorsque vous appelez l'action \_PhotoGallery avec un paramètre entier, l'action transmet le nombre correspondant de photos à l'action \_PhotoGallery. | |
| • | Implémentez un référentiel. |
| • | Refactorisez le PhotoController pour utiliser un référentiel. |
| • | Refactoriser les tests pour utiliser un référentiel fictif. |

**Noter:**Les tests que vous ajoutez à la solution dans cet exercice amélioreront la qualité du code et éviteront les bogues au fur et à mesure du développement. Cependant, cet exercice n'est pas conforme aux principes de TDD car la classe PhotoController existe déjà. Dans TDD, vous devez d'abord créer ces tests et d'autres, puis créer une classe PhotoController qui réussit les tests.

### **Exercice 2: Facultatif - Configuration de la gestion des exceptions**

##### **Scénario**

Maintenant que vous avez développé des tests unitaires pour l'application de partage de photos, vous devez configurer une stratégie de gestion des exceptions pour l'application Web MVC. Cela garantirait que lorsque des exceptions se produisent dans la phase de développement du projet PhotoSharingApplication, le contrôleur, l'action et les messages d'exception sont affichés dans une vue d'erreur MVC personnalisée. Vous devez également implémenter une action d'espace réservé pour l'action Diaporama dans la vue PhotoController. Cette action sera complétée lors d'une itération ultérieure du projet.

Effectuez cet exercice si le temps le permet.

### **Question (s) de révision**

**Vérifiez vos connaissances**

**Découverte**

**Lorsque vous avez exécuté les tests pour la première fois dans l'exercice 1, pourquoiTest\_Index\_Return\_Viewpasser, tandis queTest\_GetImage\_Return\_TypeetTest\_PhotoGallery\_Model\_Typeéchoué?**

Afficher la solution Réinitialiser

**Vérifiez vos connaissances**

**Découverte**

**Dans l'exercice 1, pourquoi tous les tests ont-ils réussi lors de la deuxième exécution?**

Afficher la solution Réinitialiser

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

Dans ce module, vous vous êtes familiarisé avec diverses techniques que vous pouvez utiliser pour éliminer les bogues de votre application Web ASP.NET MVC 5. Cela commence tôt dans la phase de développement du projet, lorsque vous définissez des tests unitaires qui garantissent que chaque méthode et classe de votre application se comporte exactement comme prévu. Les tests unitaires sont automatiquement réexécutés lorsque vous créez l'application afin que vous puissiez être sûr que les méthodes et les classes qui ont fonctionné ne sont pas interrompues par des erreurs de codage ultérieures. Des exceptions surviennent même dans les applications les mieux testées en raison de circonstances imprévues. Vous avez également vu comment garantir que les exceptions sont gérées de manière fluide et comment les journaux d'erreurs sont éventuellement stockés pour une analyse ultérieure.

**Meilleur entrainement:**Si vous utilisez TDD ou Extreme Programming, définissez chaque test avant d'écrire le code qui implémente une exigence. Utilisez le test comme une spécification complète que votre code doit satisfaire. Cela nécessite une compréhension complète de la conception.

**Meilleur entrainement:**Examinez et choisissez un cadre de simulation pour vous aider à créer des objets doubles de test à utiliser dans les tests unitaires. Bien qu'il puisse prendre du temps pour sélectionner le meilleur cadre et apprendre à coder des objets fictifs, votre investissement en temps en vaudra la peine tout au long de la vie du projet.

**Meilleur entrainement:**Ne soyez pas tenté de sauter les tests unitaires lorsque vous êtes pressé par le temps. Cela peut introduire des bogues et des erreurs dans votre système et entraîner plus de temps passé au débogage.

**Problèmes courants et conseils de dépannage**

| **Problème commun** | **Astuce de dépannage** |
| --- | --- |
| Un test unitaire prend beaucoup de temps à s'exécuter ou renvoie une erreur de connexion à une base de données. | Veuillez consulter le contenu du compagnon étudiant pour ce cours. |

### **Question (s) de révision**

**Vérifiez vos connaissances**

**Découverte**

**Vous voulez vous assurer que l'objet PhotoController transmet un seul objet Photo à la vue Affichage, lorsqu'un utilisateur appelle leChercheraction pour un titre de photo existant. Quels tests unitaires devez-vous créer pour vérifier cette fonctionnalité?**

Afficher la solution Réinitialiser

### **Outils**

NinJect, StructureMap. Il s'agit de conteneurs d'inversion de contrôle (IoC), également appelés frameworks d'injection de dépendances (DI). Ils créent des implémentations sans test d'interfaces dans votre application Web.

Moq, NSubstitué. Ce sont des cadres moqueurs. Ils automatisent la création de doubles de test pour les tests unitaires.