

Validation avec une couche service

par Matt-k (Traduction)


Date de publication : 24 avril 2009

Dernière mise à jour :

Apprenez à déplacer votre logique de validation en dehors de vos contrôleurs et dans une couche service séparée. Dans ce tutoriel, j'expliquerai comment vous pouvez conserver une nette séparation de vos couches en isolant votre couche service de votre couche contrôleur.

Traduction.....	3
Introduction.....	3
Séparation de code.....	3
Creating a Service Layer.....	4
Dissociation de la couche service.....	6
Conclusion.....	9

Traduction

Cet article est la traduction la plus fidèle possible de l'article original :  **Validating with a Service Layer**

Introduction

L'objectif de ce tutoriel est de décrire une méthode de validation dans une application ASP.NET MVC. Vous apprendrez à déplacer votre logique de validation de vos contrôleurs vers une couche service distincte.

Séparation de code

Quand vous développez une application ASP.NET MVC, vous ne devez pas mettre vos accès logiques à une base de données dans les actions de votre contrôleur. Mélanger le code du contrôleur à ceux des accès bases de données rendra votre code plus difficile à maintenir avec le temps. Il est donc recommandé de mettre les accès aux données dans une couche séparée.

Par exemple, le code ci-dessous contient un simple dépôt nommé ProductRepository. Ce dépôt contient tout le code d'accès aux données nécessaire à l'application. Cet exemple de code inclut aussi l'interface IProductRepository que le répertoire de dépôt implémente.

```
using System.Collections.Generic;
using System.Linq;

namespace MvcApplication1.Models
{
    public class ProductRepository : MvcApplication1.Models.IProductRepository
    {
        private ProductDBEntities _entities = new ProductDBEntities();

        public IEnumerable<Product> ListProducts ()
        {
            return _entities.ProductSet.ToList ();
        }

        public bool CreateProduct (Product productToCreate)
        {
            try
            {
                _entities.AddToProductSet (productToCreate);
                _entities.SaveChanges ();
                return true;
            }
            catch
            {
                return false;
            }
        }
    }

    public interface IProductRepository
    {
        bool CreateProduct (Product productToCreate);
        IEnumerable<Product> ListProducts ();
    }
}
```

Le contrôleur donné en exemple ci-dessous utilise dans ses actions Index() et Create() la couche d'accès aux données créée. Notez que le contrôleur ne contient aucune logique base de données. Créer une couche répertoire de dépôt vous permet de conserver une séparation nette de votre code. Les contrôleurs sont responsables de la logique de flux de contrôle et le répertoire de dépôt assure la logique d'accès aux données.

```
using System.Web.Mvc;
using MvcApplication1.Models;

namespace MvcApplication1.Controllers
{
    public class ProductController : Controller
    {
        private IProductRepository _repository;

        public ProductController():
            this(new ProductRepository()) {}

        public ProductController(IProductRepository repository)
        {
            _repository = repository;
        }

        public ActionResult Index()
        {
            return View(_repository.ListProducts());
        }

        //
        // GET: /Product/Create

        public ActionResult Create()
        {
            return View();
        }

        //
        // POST: /Product/Create

        [AcceptVerbs(HttpVerbs.Post)]
        public ActionResult Create([Bind(Exclude="Id")] Product productToCreate)
        {
            _repository.CreateProduct(productToCreate);
            return RedirectToAction("Index");
        }
    }
}
```

Creating a Service Layer

Donc, le flux de contrôle de l'application est géré par le contrôleur, et le répertoire de dépôt s'occupe des accès aux données. Dans ce cas, où mettre la logique de validation? Une option consiste à placer la logique de validation dans une couche service.

Une couche service dans une application ASP.NET MVC est une couche supplémentaire qui gère les communications entre le contrôleur and et la couche d'accès aux données. La couche service contient la logique business. Et en particulier, elle s'occupe de la logique de validation.

Par exemple, le couche service ProductService dans le code ci-dessous a une méthode CreateProduct(). La méthode CreateProduct() appelle ValidateProduct() pour valider un nouveau produit avant de le passer au ProductRepository.

```
using System.Collections.Generic;
using System.Web.Mvc;

namespace MvcApplication1.Models
{
    public class ProductService : MvcApplication1.Models.IProductService
    {

```

```
private ModelStateDictionary _ModelState;
private IProductRepository _repository;

public ProductService(ModelStateDictionary modelState, IProductRepository repository)
{
    _ModelState = modelState;
    _repository = repository;
}

protected bool ValidateProduct(Product productToValidate)
{
    if (productToValidate.Name.Trim().Length == 0)
        _ModelState.AddModelError("Name", "Name is required.");
    if (productToValidate.Description.Trim().Length == 0)
        _ModelState.AddModelError("Description", "Description is required.");
    if (productToValidate.UnitsInStock < 0)
        _ModelState.AddModelError("UnitsInStock", "Units in stock cannot be less than zero.");
    return _ModelState.IsValid;
}

public IEnumerable<Product> ListProducts()
{
    return _repository.ListProducts();
}

public bool CreateProduct(Product productToCreate)
{
    // Validation logic
    if (!ValidateProduct(productToCreate))
        return false;

    // Database logic
    try
    {
        _repository.CreateProduct(productToCreate);
    }
    catch
    {
        return false;
    }
    return true;
}

public interface IProductService
{
    bool CreateProduct(Product productToCreate);
    IEnumerable<Product> ListProducts();
}
}
```

Note contrôleur ProductContrôleur a été modifié dans le code ci-dessous exploiter la couche service plutôt que le répertoire de dépôt. La couche contrôleur "parle" à la couche service. La couche service "parle" à la couche accès aux données. Chaque couche a donc ses propres responsabilités.

```
using System.Web.Mvc;
using MvcApplication1.Models;

namespace MvcApplication1.Controllers
{
    public class ProductController : Controller
    {
        private IProductService _service;

        public ProductController()
        {
            _service = new ProductService(this.ModelState, new ProductRepository());
        }
    }
}
```

```
public ProductController(IProductService service)
{
    _service = service;
}

public ActionResult Index()
{
    return View(_service.ListProducts());
}

//
// GET: /Product/Create

public ActionResult Create()
{
    return View();
}

//
// POST: /Product/Create

[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create([Bind(Exclude = "Id")] Product productToCreate)
{
    if (!_service.CreateProduct(productToCreate))
        return View();
    return RedirectToAction("Index");
}

}
}
```

Notez que le ProductService est instancié dans le constructeur du contrôleur. Lorsque le ProductService est instancié, le dictionnaire d'états du modèle est passé au service. Le ProductService exploite les états du modèle pour renvoyer au contrôleur les erreurs lors de la validation.

Dissociation de la couche service

Nous n'avons donc pas réussi dès le premier coup à séparer le contrôleur de la couche service. Le contrôleur et la couche service communiquent par l'intermédiaire du ModelState. En d'autres mots, la couche service a une dépendance avec une fonctionnalité particulière du Framework ASP.NET MVC.

On veut isoler le plus possible notre couche service de notre contrôleur. En théorie, nous devrions être capable d'utiliser la couche service avec tout type d'application et pas seulement avec les applications ASP.NET MVC. Par exemple, dans le futur, nous voudrions développer une application WPF pour notre application. Nous devons donc trouver un moyen de supprimer la dépendance entre notre couche service et le ModelState d'ASP.NET MVC.

Dans le code ci-dessous, la couche service a été modifiée afin de ne plus utiliser le ModelState. A la place, elle utilise une classe qui implémente l'interface IValidationDictionary.

```
using System.Collections.Generic;

namespace MvcApplication1.Models
{
    public class ProductService : IProductService
    {
        private IValidationDictionary _validationDictionary;
        private IProductRepository _repository;

        public ProductService(IValidationDictionary validationDictionary, IProductRepository repository)
        {

```

```
_validationDictionary = validationDictionary;
_repository = repository;
}

protected bool ValidateProduct(Product productToValidate)
{
    if (productToValidate.Name.Trim().Length == 0)
        _validationDictionary.AddError("Name", "Name is required.");
    if (productToValidate.Description.Trim().Length == 0)
        _validationDictionary.AddError("Description", "Description is required.");
    if (productToValidate.UnitsInStock < 0)
        _validationDictionary.AddError("UnitsInStock", "Units in stock cannot be less than zero.");
    return _validationDictionary.IsValid;
}

public IEnumerable<Product> ListProducts()
{
    return _repository.ListProducts();
}

public bool CreateProduct(Product productToCreate)
{
    // Validation logic
    if (!ValidateProduct(productToCreate))
        return false;

    // Database logic
    try
    {
        _repository.CreateProduct(productToCreate);
    }
    catch
    {
        return false;
    }
    return true;
}

}

public interface IProductService
{
    bool CreateProduct(Product productToCreate);
    System.Collections.Generic.IEnumerable<Product> ListProducts();
}
}
```

L'interface IValidationDictionary est définie ci-dessous. Elle contient une seule méthode et une propriété.

```
namespace MvcApplication1.Models
{
    public interface IValidationDictionary
    {
        void AddError(string key, string errorMessage);
        bool IsValid { get; }
    }
}
```

La classe suivante, nommée ModelStateWrapper, implémente notre interface IValidationDictionary. Il est possible d'instancier la classe ModelStateWrapper en passant un dictionnaire d'états du modèle au constructeur de la classe.

```
using System.Web.Mvc;

namespace MvcApplication1.Models
{
```

```

public class ModelStateWrapper : IValidationDictionary
{
    private ModelStateDictionary _modelState;

    public ModelStateWrapper(ModelStateDictionary modelState)
    {
        _modelState = modelState;
    }

    #region IValidationDictionary Members

    public void AddError(string key, string errorMessage)
    {
        _modelState.AddModelError(key, errorMessage);
    }

    public bool IsValid
    {
        get { return _modelState.IsValid; }
    }

    #endregion
}

```

Enfin, nous devons modifier le contrôleur pour qu'il utilise la classe ModelStateWrapper lors de l'instanciation de la couche service.

```

using System.Web.Mvc;
using MvcApplication1.Models;

namespace MvcApplication1.Controllers
{
    public class ProductController : Controller
    {
        private IProductService _service;

        public ProductController()
        {
            _service = new ProductService(new ModelStateWrapper(this.ModelState), new ProductRepository());
        }

        public ProductController(IProductService service)
        {
            _service = service;
        }

        public ActionResult Index()
        {
            return View(_service.ListProducts());
        }

        //
        // GET: /Product/Create

        public ActionResult Create()
        {
            return View();
        }

        //
        // POST: /Product/Create

        [AcceptVerbs(HttpVerbs.Post)]
        public ActionResult Create([Bind(Exclude = "Id")] Product productToCreate)
        {

```

```
if (!_service.CreateProduct(productToCreate))
    return View();
return RedirectToAction("Index");
}
}
```

En exploitant notre interface `IValidationDictionary` et la classe `ModelStateWrapper`, nous avons pu complètement isoler notre couche service de notre contrôleur. Le couche service ne possède plus de dépendances avec le `ModelState`. Vous pouvez passer n'importe quelle classe qui implémente l'interface `IValidationDictionary` à votre couche service. Par exemple, une application WPF peut implémenter l'interface `IValidationDictionary` avec une simple classe des Collections.

Conclusion

Le but de ce tutoriel était de discuter d'une approche pour créer une logique validation dans une application ASP.NET MVC. Ainsi, vous avez appris comment déplacer toute la logique de validation en dehors de vos contrôleurs et dans une couche service séparée. Vous avez aussi appris comment isoler la couche service de votre contrôleur en développant une classe `ModelStateWrapper`.