In MVC can ViewModels access service layer? What is the best practice?

6 replies

Last post Jul 06, 2015 04:19 PM by Mikesdotnetting

In MVC can ViewModels access service layer? What is the best practice?

Jun 17, 2015 04:35 PM | lax4u

Currently I'm using DI and service locator pattern to get the instance of Service. (Note that Service just generic term I am using, and is nothing but the C# class which calls EF repository and perform data operations. Its NOT WCF service)

Is it okay to have Service instance in ViewModel? If yes, then what's the proper way to pass Service instance to ViewModel?

Approach 1 - Should the controller pass the service instance to ViewModel. In this case service properly get disposed when controller get disposed

Approach 2 - Should the ViewModel get service instance using DI & Service Locator. In this case how service will get disposed? Do I need IDisposable on ViewModel?

BaseController

```
public class BaseController:Controller
{
   private MyDomainService _myDomainServiceInstance = null;

    protected MyDomainService MyDomainServiceInstance
    {
        get
        {
            if (_myDomainServiceInstance == null)
            {
                _myDomainServiceInstance = DefaultServiceLocator.Instance.GetInstance<MyDomainService>();
            }

            return _myDomainServiceInstance;
        }
    }

    protected override void Dispose(bool disposing)
    {
        base.Dispose(disposing);
        if (_myDomainServiceInstance != null)
        {
            _myDomainServiceInstance.Dispose();
        }
    }
}
```

Controller

```
public class MyController:BaseController
{
    public ActionResult DoSomething()
    {
        // Approach 1 - Pass service Instance from base controller
        var model = new SummaryVM(this.MyDomainServiceInstance);
    }
}
```

ViewModel

```
public class SummaryVM
{
    MyDomainService _myDomainService = null;

    public SummaryVM(MyDomainService myDomainService)
    {
        //Approache 1: Controller is passing the service instance
        _myDomainService = myDomainService;
    }

    public SummaryVM()
    {
        //Aprooche 2: Use DI & Service locator pattern to get the instance

        // in this approach how service will get disposed?
        _myDomainService = DefaultServiceLocator.Instance.GetInstance<MyDomainService>();
    }

    public int[] SelectedClients { get; set; }

    public string[] SelectedStates { get; set; }


    public IEnumerable<Clients> PreSelectedClients
    {
        get
        {

            return _myDomainService.GetClients(SelectedClients);
        }
    }
}
```

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jun 18, 2015 02:13 AM | Mikesdotnetting

Conventionally, view models are seen as very lightweight classes that consist of mainly auto-implemented properties required by a view and perhaps some presentation logic. Most people populate them from within the controller. It is the controller's responsibility to prepare the data for the view.

However, if you wanted your view models to access the service layer, you can pretty much bypass the controller and use constructor injection with a DI container:

```
public class SummaryVM
{
    private IDomainService _domainService;

    public SummaryVM(IDomainService domainService)
    {
        _domainService = domainService;
    }
    // properties and methods
}
```

Using the above approach, your controller would not need to know anything about the service layer, which just seems wrong to me.

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jun 18, 2015 12:44 PM | kashyapa

agree with Mike.

The term view model is usually referred to a object which has view concerns in it i.e. some properties that is required by the view to compose the UI. a controller will get the data, then instantiate a ViewModel and pass the VM to the view. you would normally pack your VM with view based logics for e.g. if your domain model had Date property and your view needs a very specific format - you don't change your domain model rather wrap that in a view model and the view model would expose a readonly dateformatted property which would format the date as required by the UI.

nothing stops you from accessing service within your VM - you can always inject a Service to your VM. but you don't generally do that

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jun 18, 2015 05:40 PM | lax4u

Well I do understand why the ViewModel is used and the role it play in MVC architecture. The examples & articles I found are not using real time scenarios. Everyone is saying the same thing, and I do agree that there should be separation of concern between Controller, ViewModel & Service.

However, I could not find a design pattern to populate ViewModel in Controller. If ViewModel has bunch of unrelated properties, meaning there is no DB relationship between these entities, then you cannot load these properties by making one service call(at least not using EF entities). or lets say some properties are populated by making external call.

I certainly do not want to pass Service using DI to ViewModel Constructor. That's because typically you do not implement IDisposable on ViewModel, and because of that you can't control when to dispose the service.

So if the controller has to populate ViewModel, then below is the pseudo code

```
 public class MyController : Controller
    {
        private MyDomainService _service = new MyDomainService();

        public ActionResult DoSomething()
        {
            MyViewModel model = new MyViewModel();



            model.Clients = Mapper.Map<ClientViewModel>(_service.GetClients());
            model.Countries = Mapper.Map<CountryViewModel>(_service.GetCountries());
            model.Users = Mapper.Map<UserViewModel>(_service.GetUsers());
            model.Types = Mapper.Map<TypeViewModel>(_service.GetTypes());
            model.Statuses = Mapper.Map<StatusViewModel>(_service.GetStatuses());

            return View(model);
        }


       protected override void Dispose(bool disposing)
       {
            base.Dispose(disposing);
            _service.Dispose();
       }

}
```

In the example above there are lot of assignment operations in controller. It can also have if..then..else logic to assign viewmodel's properties based on certain conditions. So the controller's logic gets ugly.

so I came up with the following

```
public class ImportSummaryController : BaseImportController
{
        private MyDomainService _service = new MyDomainService();

        public ActionResult Index()
        {
            MyViewModel model = new MyViewModel();
            model.PopulateViewModel(_service);
            return View(model);
        }

       protected override void Dispose(bool disposing)
       {
            base.Dispose(disposing);
            _service.Dispose();
       }
}
```

so here i'm passing service to ViewModel via method, so service can get disposed along with the Controller. But now ViewModel is now making the service call to populate its own properties. I'm not sure its the correct pattern.

I hope im explaining my concern properly.

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jun 19, 2015 02:13 AM | Mikesdotnetting

I use the first pattern. I don't have my services implement IDiposable. I let Garbage Collection take care of them. If the services make any calls to unmanaged resources, I make those calls in using statements. I used to use using statements for the DbContext too, but stopped doing that after I discovered that the default behaviour of the context is to take care of connection management for you: **http://blog.jongallant.com/2012/10/do-i-have-to-call-dispose-on-dbcontext.html (http://blog.jongallant.com/2012/10/do-i-have-to-call-dispose-on-dbcontext.html)** .

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jul 06, 2015 02:53 PM | lax4u

The other scenario where i think ViewModel needs to have access to the service/repository is when you are validating model on server. For Example

```
public class MyModel : IValidatableObject
{
        // removed code from brevity purpose

        public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
        {
            // here i need to retrieve data from the database and do validation logic.
            // so i need access to service/repository
        }
}
```

CONTROLLER VIEWMODEL MVC DESIGN

Re: In MVC can ViewModels access service layer? What is the best practice?

Jul 06, 2015 04:19 PM | Mikesdotnetting

> **lax4u**
>
> I certainly do not want to pass Service using DI to ViewModel Constructor.

Then don't. It's not a common pattern. You asked if it could be done. I showed how it could be done. I didn't say it's a good idea or recommended practice.

> **lax4u**
> That's because typically you do not implement IDisposable on ViewModel, and because of that you can’t control when to dispose the service.

Why would you need to dispose of the service? Garbage collection will take care of it. If some methods in your service make calls to unmanaged resources such as databases, file systems, SMTP servers etc. make those calls in using statements so that they are properly disposed within the service.

> **lax4u**
> However, I could not find a design pattern to populate ViewModel in Controller.

It's a common practice. I don't think anyone has given the practice a design pattern name. I don't think it deserves one.

> **lax4u**
> If ViewModel has bunch of unrelated properties, meaning there is no DB relationship between these entities, then you cannot load these properties by making one service call(at least not using EF entities). or lets say some properties are populated by making external call.

Correct. Some of my more complex ViewModels will make use of several services to get their data. It's not a problem. I use AutoMapper to keep the code in the controller lean.

CONTROLLER VIEWMODEL MVC DESIGN