How does a service layer fit into my repository implementation?

Asked 8 years, 9 months ago Active 1 year, 9 months ago Viewed 11k times

52

33

I have created a POCO model class and a repository class which handles persistence. Since the POCO cannot access the repository, there are lots of business logic tasks in the repository which doesn't seem right. From what I have read, it looks like I need a service layer that sits between the UI consumers and the repository layer. What I am not sure of is exactly how it is supposed to work...

In addition to the service layer, should there also be a separate business logic layer, or is that the role of the service layer?

Should there be one service per repository?

Is the service layer the only way that the UI can instance the model objects or does the repository provide the new model instance to the service?

Do I put my parameter, model and other validations in the service layer that do things like check to make sure a input is valid and that a item to update exists in the database before updating?

Can the model, repository and UI all make calls to the service layer, or is it just for the UI to consume?

Is the service layer supposed to be all static methods?

What would be a typical way to call the service layer from the UI?

What validations should be on the model vs the service layer?

Here is some sample code for my existing layers:

```
public class GiftCertificateModel
{
    public int GiftCerticiateId {get;set;}
    public string Code {get;set;}
    public decimal Amount {get;set;}
    public DateTime ExpirationDate {get;set;}

    public bool IsValidCode(){}
}


public class GiftCertificateRepository
```

```
{
    //only way to access database
    public GiftCertificateModel GetById(int GiftCertificateId) { }
    public List<GiftCertificateModel> GetMany() { }
    public void Save(GiftCertificateModel gc) { }
    public string GetNewUniqueCode() { //code has to be checked in db }

    public GiftCertificateModel CreateNew()
    {
        GiftCertificateModel gc = new GiftCertificateModel();
        gc.Code = GetNewUniqueCode();
        return gc;
    }
}
```

UPDATE: I am currently using web forms and classic ADO.NET. I hope to move to MVC and EF4 eventually.

UPDATE: Big thanks to @Lester for his great explanation. I now understand that I need to add a service layer for each of my repositories. This layer will be the ONLY way the UI or other services can communicate with the repository and will contain any validations that do not fit on the domain object (eg - validations that need to call the repo)

```
public class GiftCertificateService()
{

    public void Redeem(string code, decimal amount)
    {
        GiftCertificate gc = new GiftCertificate();
        if (!gc.IsValidCode(code))
        {
            throw new ArgumentException("Invalid code");
        }

        if (amount <= 0 || GetRemainingBalance(code) < amount)
        {
            throw new ArgumentException("Invalid amount");
        }

        GiftCertificateRepository gcRepo = new GiftCertificateRepository();
        gcRepo.Redeem(code, amount);
    }

    public decimal GetRemainingBalance(string code)
    {
        GiftCertificate gc = new GiftCertificate();
        if (!gc.IsValidCode(code))
        {
```

```
            throw new ArgumentException("Invalid code");
        }

        GiftCertificateRepository gcRepo = new GiftCertificateRepository();
        gcRepo.GetRemainingBalance(code);
    }

    public SaveNewGC(GiftCertificate gc)
    {
        //validates the gc and calls the repo save method
        //updates the objects new db ID
    }

}
```

Questions

1. Do I add the same (and possibly more) properties to the service as I have on my model (amount, code, etc) or do I only offer methods that accept GiftCertificate objects and direct parameters?
2. Do I create a default instance of the GiftCertificate entity when the Service constructor is called or just create new ones as needed (eg - for validation methods in the service that need to call validation methods in the entity? Also, same question about creating a default repository instance...?
3. I know i expose the functionality of the repo via the service, do I also expose the methods from the entity as well (eg - IsValidCode, etc)?
4. It is ok for the UI to simply create a new GiftCertificate object directly without going through the service (eg - to call parameter validation methods from the entity). If not, how to enforce it?
5. On the UI layer, when I want to create a new gift certificate, do I call the model/service validations (like IsValidExpirationDate, etc) directly from the UI layer OR do I hydrate the object first, then pass it in to be validated and then return some sort of validation summary back to the UI?

Also, if I want to Redeem from the UI layer, do I first call the model/service validation methods from the UI to give user feedback and then call the Redeem method which will run the same checks again internally?

Example for calling service to do a Redeem operation from UI:

```
string redeemCode = RedeemCodeTextBox.Text;

GiftCertificateService gcService = new GiftCertificateService();
GiftCertificate gc = new GiftCertificate(); //do this to call validation methods (should
be through service somehow?)
```

```
if (!gc.IsValid(redeemCode))
{
    //give error back to user
}

if (gcService.GetRemainingBalance(redeemCode) < amount)
{
    //give error back to user
}

//if no errors
gcService.Redeem(code,amount);
```

Example for creating a new Gift certificate from UI:

```
GiftCertificateService gcService = new GiftCertificateService();
GiftCertificate gc = new GiftCertificate();

if (!gc.IsValidExpDate(inputExpDate))
{
    //give error to user..
}

//if no errors...
gc.Code = gcService.GetNewCode();
gc.Amount = 10M;
gc.ExpirationDate = inputExpDate;
gcService.SaveNewGC(gc);
//method updates the gc with the new id...
```

Something feels wrong about way GCs are being created and how the validations are separated between entity/service. The user/consumer should not have to be concerned with what validations are in which place... advice?

c# asp.net domain-driven-design repository repository-pattern

edited May 26 '11 at 15:53

asked May 26 '11 at 12:43

jpshook

4,186 6 32 43

Related post - ASP.NET MVC (Domain Model, Repository, Fluent, Services - Structure for my Project) & Best Structure for ASP.NET MVC Solution – RBT Aug 28 '18 at 11:20

3 Answers

42

Take a look at S#arp Architeture . It's like a best practices architectural framework for building ASP.NET MVC applications. The general architecture pattern is to have 1 repository per entity which is responsible only for data access and 1 service per repository which is responsible only for business logic and communicating between controllers and services.

To answer your questions based on S#arp Architeture:

In addition to the service layer, should there also be a separate business logic layer, or is that the role of the service layer?

Models should be responsible for field-level validation (ex. using required field attributes) while controllers can validate data before saving (ex. checking state before saving).

Should there be one service layer per repository?

Yes - there should be one *service* per repository (not 1 service layer per repository but I'm guessing you meant that).

Is the service layer the only way that the UI can instance the model objects or does the repository provide the new model instance to the service?

Repositories and services can return a single entity, a collection of entities, or data transfer objects (DTOs) as required. Controllers will pass these values to a static constructor method in the model which will return an instance of the model.

ex Using DTOs:

```
GiftCertificateModel.CreateGiftCertificate(int GiftCerticiateId, string Code, decimal
Amount, DateTime ExpirationDate)
```

Do I put my parameter, model and other validations in the service layer that do things like check to make sure a input is valid and that a item to update exists in the database before updating?

Models validate field-level values ex. making sure input is valid by checking for required fields, age or date ranges, etc. Services should do any validation needed that requires checking outside of the model value ex. Checking that the gift certificate hasn't been redeemed yet, checking properties of the store the gift certificate is for).

Can the model, repository and UI all make calls to the service layer, or is it just for the UI to consume?

Controllers and other services should be the only ones making calls to the service layer. Services should be the only one makes making calls to repositories.

Is the service layer supposed to be all static methods?

They can be but it's easier to maintain and extend if they aren't. Changes to entities and adding/removing subclasses are easier to change if there's 1 service per entity / subclass.

What would be a typical way to call the service layer from the UI?

Some examples of controllers calling the service layer:

```
giftCertificateService.GetEntity(giftCertificateId); (which in turn is just a call to
the giftCertificateRepository.GetEntity(giftCertificateId)

giftCertificateService.Redeem(giftCertificate);
```

What validations should be on the model vs the service layer?

Already answered above.

UPDATE

Since you're using WebForms it may be a little harder to grasp some of the concepts but everything I've mentioned is applicable since what I'm describing is a general MVC paradigm. ADO.NET for data access doesn't matter since data access is decoupled via repositories.

Do I add the same (and possibly more) properties to the service as I have on my model (amount, code, etc) or do I only offer methods that accept GiftCertificate objects and direct parameters?

You need to look at the services as exactly what their name implies - actions that controllers can invoke. You won't need properties that are defined in your model since they are already available in the model.

Do I create a default instance of the GiftCertificate entity when the Service constructor is called or just create new ones as needed (eg - for validation methods in the service that need to call validation methods in the entity? Also, same question about creating a default repository instance...?

Controllers and services should have private fields for services and repositories respectively. You shouldn't be instantiating for every action / method.

I know i expose the functionality of the repo via the service, do I also expose the methods from the entity as well (eg - IsValidCode, etc)?

Not too sure what you mean here. If services return entities then those methods on the entities are already exposed. If they return DTOs then that implies you're interested only in certain information.

For validation I can see why you're a bit concerned since there's validation done directly on the model and other types of validation done in services. The rule of thumb I've used is that if validation requires calls to the db then it should be done in the service layer.

It is ok for the UI to simply create a new GiftCertificate object directly without going through the service (eg - to call parameter validation methods from the entity). If not, how to enforce it?

On the UI layer, when I want to create a new gift certificate, do I call the model/service validations (like IsValidExpirationDate, etc) directly from the UI layer OR do I hydrate the object first, then pass it in to be validated and then return some sort of validation summary back to the UI?

For these 2 questions lets go through a scenario:

User enters information to create a new certificate and submits. There is field level validation so if a textbox is null or if the dollar amount is negative it throws a validation error. Assuming all fields are valid the controller will call the service `gcService.Save(gc)` .

The service will check other business logic, such as if the store already has issued too many gift certificates. It either returns an enum for the status if there are multiple error codes or throws an exception with the error information.

Finally, the service calls `gcRepository.Save(gc)` .

edited May 9 '18 at 20:14

secretwep
634 10 25

answered May 26 '11 at 14:20

Lester
3,733 2 22 28

@Lester - You rock! finally starting to pull it all together. I should have noted in the post that I am using web forms and classic ADO for data access. Please see my additional questions and code above and let me know your thoughts. I am still a bit fuzzy on how to instantiate new GiftCertificate objects and validate parameters from the UI layer. – jpshook May 26 '11 at 15:54

Updated with answers to your new questions. I'm glad to be of help but I think it'll be a lot easier and make sense more quickly if you were using ASP.NET MVC. Although like I mentioned the MVC paradigm is still valid in WebForms. – Lester May 26 '11 at 16:59

@Lester - Will you please review my UI calling code samples and let me know if that is how it should work? For field level validations in the UI, should it not call the validation methods on the model and/or service as needed? Also, from what you are saying, it is ok for the UI to create gc objects outside of the service and then submit them to the service... – jpshook May 26 '11 at 17:15

I do plan to move to MVC as well, but trying to get my head around some architecture fundamentals as I have some basic web forms experience and can only take on so much at once. – jpshook May 26 '11 at 17:16

Both code samples look fine except for GiftCertificate gc = new GiftCertificate(); I mentioned in my answer you shouldn't be instantiating just to do validation. Take a look at S#arp they have full code solutions that you can look through. – Lester May 26 '11 at 17:27

3

1. You don't have to create repository per entity,see here for more,

> Usually one defines a repository per aggregate in the domain. That is: we don't have a repository per entity! If we have a look at a simple order entry system the entity Order might be the root of a Order aggregate. Thus we will have an Order Repository.

2. Should there be one service per repository? -> Not always, as you may use multiple repositories in one service.
3. Service creates Model instance, repository will never interact to Model, in fact it returns the Entity which model will use subsequently.
4. Handle Input/range etc kind of validation at UI level(u can use javascript or any other library), and let Services handle only business aspects. You can get the benefits of Attributes which will does the same.
5. UI->Service->Repository, if repository is calling service than thr must be something wrong IMO.

You code changes,

1. Make Model and Repositories separate.

```
public class GiftCertificateModel
{
}
public class GiftCertificateRepository
{
   //Remove Model related code from here, and just put ONLY database specific code here,
(no business logic also). Common methods would be Get, GetById, Insert, Update etc.

    Since essence of Repository is to have common CRUD logic at one place soyou don't
have to write entity specific code.
    You will create entity specific repository in rare cases, also by deriving base
repository.

}

public class GiftCertificateService()
{
    //Create Model instance here
    // Use repository to fill the model (Mapper)

}
```

edited May 26 '11 at 18:08

answered May 26 '11 at 17:27

paragy
6,585 3 23 38

@alliswell - when you say "3.Service creates Model instance, repository will never interact to Model, in fact it returns the Entity which model will use subsequently." What is the difference between the model and entity? Right now I have Service/Repository/POCO. The repository grabs the data and calls a mapper which maps to my POCOs (Domain objects/entities/whatever). Are you saying that the service should be creating new empty entities when I need to make a new one? Any advice on my last 2 code samples? – jpshook May 26 '11 at 17:41

Entity which is related database(repository) and model in here which is attached with UI. So, your approach is correct except I mean Service will call repository to fill/insert/update data, and UI will interact with Service. So repositories will never interact with UI directly. (Separation of Concern) – paragy May 26 '11 at 17:49

@alliswell - gotcha! New gift certificates require a special code that must be generated and checked for uniqueness in the database. Should my UI create the new gift certificate entity and get the code from service layer, then call service to save OR should it simply call the service layer which will return a new gift certificate object with code already filled OR should the UI just call a method on the service for create new that takes parameters for amount, expDate and it does everything create objects, get code, save to database and pass back complete object all in one step? – jpshook May 26 '11 at 18:09

Ok,in such case first query for code uniqueness, if it is unique than do corresponding steps, like save or get. Also If ur UI dont need that code than pass everything, else generate it first than use it and pass values for modification. – paragy May 26 '11 at 18:28

Re: #4, aren't which fields are required to purchase a gift card defined by the business, and thus business access? What if you want to build a WP7 app that enables the same use cases, doesn't this suggestion mean you have to recode that validation again in the WP7 UI? What about DRY? – Andy May 26 '11 at 18:35

0

You can create a service called GiftCertificateService.

That way you will coordinate any task that does not belong to the responsibility of the GiftCertificateModel into it's service. (Not to be confused with a WCF service).

The service will control all the tasks so your UI (or whatever caller that might be) will use the methods defined in the service.

The service will then call methods on the model, use the repository, create transactions, etc.

For ex. (based on the sample code you provided):

```
public class GiftCertificateService
{
   public void CreateCertificate()
   {
```

```
        //Do whatever needs to create a certificate.
        GiftCertificateRepository gcRepo = new GiftCertificateRepository();
        GiftCertificateModel gc = gcRepo.CreateNew();
        gc.Amount = 10.00M;
        gc.ExpirationDate = DateTime.Today.AddMonths(12);
        gc.Notes = "Test GC";
        gcRepo.Save(gc);
    }
}
```

The UI will call the CreateCertificate method (passing arguments, etc) and the method may return something also.

NOTE: If you want the class to act on the UI though then create a controller class (if you are doing MVC) or a presenter class (if you are doing MVVM, and don't want to put everything inside the ViewModel) and use the GiftCertificateService from that class.

answered May 26 '11 at 13:49

Nikos Baxevanis
9,276 2 38 73

I am using web forms with classic ADO for database interaction. So, do I need to create all the same properties on the service as I have on my model for amount, expiration date, etc? And just to clarify, I will never call new repo, or new model from the UI, but only from the service layer.. eg GiftCertificate gc = GiftCertificateService.NewGiftCertificateObject(), then complete the object and call the service save methods? – jpshook May 26 '11 at 14:35

If you are using web forms most chances are your views are either autonomous with event handlers or follow a model-view-x pattern (either presenter or controller). Event handlers or controllers/presenters are the right place where you can call methods on your services (even perform data access although most guidelines discourage this). You don't need to create all the same properties. You can add a reusable layer on top (that's a service here) and from there call methods that coordinate actions (data access, validation, calls to model objects, etc). – Nikos Baxevanis Jun 6 '11 at 9:04