Best architecture design using service layer and interacting services?

Asked 5 years, 1 month ago Active 3 years, 3 months ago Viewed 1k times

1

I have several services that are currently highly decoupled. Now I have to extend them and they need to depend to access each other.

Let's say I have 4 services: **EmailService**, **HouseService**, **UserService**, **PriceService**. Each user has an email address and each user belongs to a house.

I want to send an email to each user about the price of the house that they are connected to. So in the EmailService I have *SendEmailToAddress(string email, string text)*, in PriceService I have *GetHousePrice(int id)*, in HouseService I have *GetUsersInHouse(int id)* and in UserService I have *GetEmailOfUser(int id)*.

What would be the best approach to send an email to all the users from the HouseController? Should I just init all the services in the controller action and call each one in order or should I use the Mediator pattern? If I should use it, it would probably contain only one method so it seems a bit of an overkill. Also if I use it everywhere should I create different mediators for each service connection or should it be only one class that has all my services as private properties and then in the methods use only the once I need for a specific action? If I go with the Mediator pattern should I use it in every controller or should I stick with the bare services where they don't need to interact together (e.g. if I only need a list of houses I think it's probably best to just get them directly from the service object instead of the Mediator)?

c# asp.net-mvc asp.net-mvc-5 service-layer mediator

asked Jan 28 '15 at 9:54

Marti Markov
564 3 21

are you also using a repository layer above service layer for database interaction ? – Ankush Jain Jan 28 '15 at 10:08

Yeah, I am. Would that make some sort of difference? Except transaction wise? – Marti Markov Jan 28 '15 at 10:10

4 Answers

0

As I was looking for best practices since past couple of days in ASP.Net MVC and I concluded that our services should contain all business logic (using repositories of different domain models) and expose public methods that are accessible by controller.

In your case you should create a new service and put the whole logic of calculation and sending email in a method of that service. So that your service will work like a black box. Other developers (who work on your project) don't need to know that how thing are managed in that method. All they need to know is to call that method with required parameter and handle response.

answered Jan 28 '15 at 10:19

Ankush Jain
3,354 2 17 35

Yes but the question then is what is the best place for that? It could go in either House or User. And I gave a very simple example. The actual problem involves 7 or 8 different services. The point is that I don't want to duplicate code at all. So if I put it in one service I will have to duplicate at least one method from the other service, which is why I was looking into the Mediator pattern. – Marti Markov Jan 28 '15 at 10:24

In that case you can move your method in repository and call that method from repository in both services. Concern is that a service method should not call it's own service method or other service methods, It should only call repository methods. This is what i concluded when i was building an architecture and though of such issue. This is totally what i think, May be there are some better solution. – Ankush Jain Jan 28 '15 at 10:51

Yeah but adding the code to send emails... I don't know, it doesn't seems like the best practise to put it in repository which is used to save them to the database. The way I'm using the repository is to save or get stuff from the db. Do you think this is correct? – Marti Markov Jan 28 '15 at 18:15

Yes you are right. Repositories should be only used for database operations. But while we make a web-application there may be lots of functionality which is common to every application. So I would suggest to make a class library for such tasks like sending mails. – Ankush Jain Jan 29 '15 at 7:09

0

Just create HouseServiceFacade that contains the services you need. In this facade you can put all methods for the controller.

answered Feb 7 '15 at 10:06

Michele Scotucci
1

0

You could create a workflow service that contains the actual logic to look up the information and send the mail using the existing services.

This service is then called from your HouseController. You could use the service directly as a class library or expose it as a WCF service; but it depends on your requirements.

This way your entity services remain loosely coupled, and all of your cross-service logic is in a dedicated component.

answered Jan 28 '15 at 10:13

L-Four

10.9k 6 47 92

By service I meant service layer, the design pattern: techbrij.com/… Here is example of it. – Marti Markov Jan 28 '15 at 10:22

0

Given that your services aren't actually needing to communicate with each other, you just need to call various methods on each and use the return values to complete a higher level task, I don't think the Mediator pattern is appropriate here.

For example, its not like you need the HouseService to manipulate the state of objects managed by the PriceService...you just need data from the PriceService that the HouseService provides input for:

```
var houseId = houseService.GetIdOfHouse(someCriteria);
var price = priceService.GetPriceOfHouse(houseId);
```

Instead, I think what you need to implement is the Facade pattern, which will:

> Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Good example of Facade pattern can be found on the dofactory.com site:

http://www.dofactory.com/net/facade-design-pattern

Here's what I would consider doing:

```
public class NotificationFacade
{
    private IPriceService _priceService;
```

```
    private IHouseService _houseService;
    private IUserService _userService;
    private IEmailService _emailService;

    public NotificationFacade(IPriceService priceService, IHouseService houseService,
IUserService userService, IEmailService emailService)
    {
        _priceService = priceService;
        _houseService = houseService;
        _userService = userService;
        _emailSerice = emailSerice;
    }

    public void NotifyUsersAboutPriceForHouse(int houseId)
    {
        var price = _priceService.GetHousePrice(houseId);
        var users = _houseService.GetUsersInHouse(houseId);
        foreach(var user in users)
        {
            var emailAddress = _userService.GetEmailOfUser(user);
            _emailService.SendEmailToAddress(emailAddress, "Your House Price is:" +
price);
        }
     }
 }
```

In your controller:

```
public HouseController
{
    private NotificationFacade _notificationFacade;
    public HouseController(NotificationFacade notificationFacade)
    {
        _notificationFacade = notificationFacade;
    }

    public void SomeActionMethod(int houseId)
    {
        _notificationFacade.NotifyUsersAboutPriceForHouse(houseId);
    }
}
```

The dependencies should be resolved using Dependency Injection with a container such as Unity, Ninject, StructureMap or something similar...

answered Nov 10 '16 at 8:01

JTech
2,730 4 31 42