Mosh Hamedani

Coding Made Simple

- Courses20
- Tutorials
- YouTube
- Discounts
- About me
- Contact me

RSS
April 3rd, 2017
Comments

4 Common Mistakes with the Repository Pattern

Connect with Me

UPDATE (Nov 5 2018): While you’re here to become a better C# developer, I strongly recommend you to watch my Python tutorial on YouTube. Python is super-hot these days. It's the number one language employers are looking for and gives you 4x more job opportunities than C#.

As part of my new ASP.NET Core course, I've been encouraging my students to post their code to GitHub. While reviewing various solutions, I've encountered a few common mistakes in the implementation of the repository pattern.

One repository per domain

You should think of a repository as a collection of domain objects in memory. If you're building an application called Vega, you shouldn't have a repository like the following:

```
public class VegaRepository
{
}
```

Instead, you should have a separate repository per domain class, like OrderRepository, ShippingRepository and ProductRepository.

Repositories that return view models/DTOs

Once again, a repository is like a collection of domain objects. So it should not return view models/DTOs or anything that is not a domain object. I've seen many students using AutoMapper inside their repository methods:

```
public IEnumerable<OrderViewModel> GetOrders()
{
    var orders = context.Orders.ToList();

    return mapper.Map<List<Order>, List<OrderViewModel>(orders);
}
```

Mapping is not the responsibility of the repository. It's the responsibility of your controllers. Your repositories should return domain objects and the client of the repository can decide if it needs to do the mapping. By mapping the domain objects to view models (or something else) inside a repository, you prevent the client of your repositories from getting access to the underlying domain object. What if you return OrderViewModel but somewhere else you need OrderDetailsViewModel or OrderSnapshotViewModel? So, the client of the repository should decide what it wants to map the Order object to.

Categories

- Angular
- ASP.NET
- Backend
- C# / .NET
- Courses
- Flutter
- General
- JavaScript
- Mobile
- Python
- React
- React Native

Popular Posts

Save/Update method in repositories

Yet another very common mistake! As I've explained in my YouTube video before, your repositories should **not** have a Save() or Update() method. I repeat: think of a repository as a collection of domain objects in memory. Do collections have a Save() or Update() method? No! Here's an example:

```
var list = new List<int>();
list.Add(1);
list.Remove(1);
list.Find(1);

list.Save();    // doesn't exist!
list.Update();  // doesn't exist!
```

Another reason your repositories should not have a Save() method is because sometimes as part of a transaction you may work with multiple repositories. And then you want to persist the changes across multiple repositories in one transaction. Here's an example:

```
orderRepository.Add(order);
orderRepository.Save();

shippingRepository.Add(shipping);
shippingRepository.Save();
```

Can you see the problem in this code? For each change, we need a separate call to the Save() method on the corresponding repository. What if one of these calls to the Save() method fails? You'll end up with a database in an inconsistent state. Yes, we can wrap that whole thing inside a transaction to make it even more ugly!

A pattern that goes hand in hand with the repository pattern is the unit of work. With the unit of work, we can re-write that ugly code like this:

```
orderRepository.Add(order);
shippingRepository.Add(shipping);
unitOfWork.Complete();
```

Now, either both objects are saved together or none are saved. The database will always be in a consistent state. No need to wrap this block inside a transaction. No need for two separate calls to the Save() method!

If you want to learn how to implement the repository and unit of work pattern together, watch my

React vs. Angular: The Complete Comparison

53 Python Exercises and Questions for Beginners

React Functional or Class Components: Everything you need to know

4 Common Mistakes with the Repository Pattern

5 C# Collections that Every C# Developer Must Know

Python 3 Cheat Sheet

React Lifecycle Methods – A Deep Dive

YouTube video here.

Repositories that return IQueryable

One of the reasons we use the repository pattern is to encapsulate fat queries. These queries make it hard to read, understand and test actions in ASP.NET MVC controllers. Also, as your application grows, the chances of you repeating a fat query in multiple places increases. With the repository pattern, we encapsulate these queries inside repository classes. The result is slimmer, cleaner, more maintainable and easier-to-test actions. Consider this example:

```
var orders = context.Orders
    .Include(o => o.Details)
        .ThenInclude(d => d.Product)
    .Where(o => o.CustomerId == 1234);
```

Here we are directly using a DbContext without the repository pattern. When your repository methods return IQueryable, someone else is going to get that IQueryable and compose a query on top of it. Here's the result:

```
var orders = repository.GetOrders()
    .Include(o => o.Details)
        .ThenInclude(d => d.Product)
    .Where(o => o.CustomerId == 1234);
```

Can you see the difference between these two code snippets? The only difference is in the first line. In the first example, we use **context.Orders**, in the second we use **repository.GetOrders().** So, what problem is this repository solving? Nothing!

Your repositories should return domain objects. So, the **GetOrders()** method should return an **IEnumerable.** With this, the second example can be re-written as:

```
var orders = repository.GetOrders(1234);
```

See the difference?

What are the other issues you've seen in the implementation of the repository pattern? Share your thoughts!

If you enjoyed this article, please share it.

Mosh

Hi! My name is Mosh Hamedani. Over the last three years, I’ve taught over 1M students how to code or how to become a better coder through my online courses and YouTube channel. I’ve been working as a software engineer for the past 18 years and I love to share my knowledge with you. It’s my mission to make coding accessible to everyone.

Share this:

Related

Why I don't like IEntity interfaces
March 3, 2016
In “C# / .NET”

Repositories or Command / Query Objects?
February 26, 2016
In “C# / .NET”

Layered Architecture in ASP.NET Core Applications
April 18, 2017
In “ASP.NET”

Tags: entity framework, repository-pattern

61 responses to “4 Common Mistakes with the Repository Pattern”

Dayshine says:
April 3, 2017 at 12:15 pm

“Mapping is not the responsibility of the repository. It’s the responsibility of your controllers”

What about mapping from your repository object to your domain object?

This is all good advice for the interface. But aren't you glossing over the implementation part of your repository pattern? None of this is actual *implementation*.

Reply

Saeid says:
April 3, 2017 at 1:04 pm

About "Repositories that return view models/DTOs"
Suppose your domain model has 20 fields with large amount of data and you want to use only 3 fields here, you have to fetch the whole row first and then map it, which is very inefficient.

Reply

admin says:
April 13, 2017 at 9:37 am

Very good point!

What you're referring to here is a simple data structure and not a domain object. You want this structure with 3 properties only for a specific view (or multiple views) in the application. This data structure is nothing but a data container. It's not a domain object. It doesn't have any methods, neither does it encapsulate any business rules.

To expand more on your example, those 3 properties may be mapped to a subset of columns in 1 table, or they can be the result of joining N tables and picking 3 columns. In either case, it is not a domain object, and by definition, it doesn't fit the definition of the repository pattern.

These objects are only used for the read operations and for these I use a separate group of classes, which I call providers (eg CustomerProvider). It has a bunch of methods for querying customer data mainly for reporting. In the implementation of this provider, you can also use AsNoTracking() to improve the performance. You wouldn’t do that in your repository implementations because you need to keep track of changes to your domain objects and persist them in the database. You may even use a stored procedure for performance optimization.

The more complex your application gets, the more you may want to consider separating the read and write models. And then eventually you’ll find in the CQRS realm.

Reply

Leon says:
September 21, 2017 at 12:48 pm

Would you care to elaborate on what you call Providers? I’m really curious because I often find myself using AsNoTracking() and really large “reporting” or “stats” queries joining multiple tables and I don’t know where to put them.

Also unsure if it is okay in this pattern to return a custom class of combined domain entities that is not a ViewModel but just a data structure which then can be mapped to a ViewModel.

Thanks a lot for your courses. Landed my first software dev job much because of you.

Reply

Mosh says:
September 21, 2017 at 4:13 pm

The scenario you’re talking about is for reporting. In these situations, you’re not working with your domain objects. As you said, you may join multiple tables or select a subset of their columns and return the data in a structure that fits your reports. So, you wouldn't be using a repository in this class. You could have a provider class that is purely used for returning read-only data with AsNoTracking. You may have several reports around employees. You can have EmployeeProvider with methods such as GetEmployeesReturningNextYear(), GetEmployeesEligibleForIncentives(), etc.

Reply

Leon says:
September 21, 2017 at 4:34 pm

Alright. What are the objects being returned then? Viewmodels? Just projected domain objects which later can be mapped to viewmodels?

I assume these Provider classes fit into Persistance layer and not core? Should I then do like repositories put interfaces in Core? Then use DI with the interface? Do providers have access tp Context? Because like Saeid said with these large reporting queries you would want to optimize Sql generation which i guess is easier with Context.

Thank you so much for your time and effort.

Leon says:
[September 21, 2017 at 5:06 pm]

In your ASP.NET Core Udemy tutorial you had a VehicleRepository.cs return a QueryResult – which is not an entity? Would that method fit more into a Provider class?

Leon says:
[September 25, 2017 at 3:36 pm]

I hope you get a chance to answer my latest questions here.. Perhaps even write a blog post or make a video? I just got really confused and I think this is largely misunderstood by many others!

Leon says:
[September 27, 2017 at 10:08 pm]

Continued experimenting with this. Another question came up. I guess automapper will be heavily used in provider classes if they return viewmodels/resources. Utilizing IQueryable extension ProjectTo().

Now how do I unit test provider methods if they are dependent on automapper?

Leon says:
[September 28, 2017 at 8:37 pm]

You commented somewhere else that repository is for getting data and mapping is the responsibility of controller but now I'm thinking EVERY Get request should just go through a "Provider" returning ViewModels for better generated SQL and performance..?

Sorry if I'm asking too much.

Leon says:
September 21, 2017 at 12:49 pm

Also just in general when returning objects from repository that will be used for readonly situations such as details pages. Got confused when you said AsNoTracking() doesn't fit in here.

Reply

leo says:
October 14, 2017 at 4:25 pm

i'm waiting for the answer from mosh to the very good questions of Leon 😀

Reply

Diego says:
October 19, 2017 at 10:21 pm

That make two of us.

leo says:
October 28, 2017 at 7:30 pm

actually we are 3, there’s Leon, leo and diego,haha! ^_^

Evgeny says:
November 30, 2017 at 7:24 am

I’m also confusing about CustomerProvider class. What type of query it should return and how it’s type should mapping to ViewModel for example.

Clayton says:
February 6, 2019 at 9:27 pm

I also would love to see these questions answered!

Bartosz Jarmuż says:
May 18, 2018 at 2:39 pm

Hey man,
Good article – however, in this comment you have left quite a 'controversial' notion of 'doesn't fit the repository pattern. (…) I use a separate group of classes, which I call providers (eg CustomerProvider) (…) mainly for reporting. It would be good if you could either elaborate on this or at least point to other sources which describe this approach. I haven't seen any notion like that in the internet.
Cheers

Reply

Rahul says:
April 5, 2017 at 12:45 pm

Hi Mosh,

@'Save/Update method in repositories'

Lets assume that we need convert existing application to its latest tech with pattern, where the request or update are done with existing SQL or Stored procedures.

Please let us know how do i implement the Unit of work complete method (needs to be utilized existing stored procedures for update function) and if not please prove as stated 'Repositories should not have a Save() or Update() method'

Reply

admin says:
April 13, 2017 at 9:30 am

Hi Rahul,

I recommend you to watch my YouTube video first and then read my comment below:

https://youtu.be/rtXpYpZdOzM

Regarding your question about implementing the unit of work for an application with existing stored procedures: you don’t really need to implement a unit of work from scratch. One complex part of this implementation is about keeping track of object changes in memory. DbContext already does that for you and you don’t need to re-implement it. You can map the CRUD operations to stored procedures in the database:

https://msdn.microsoft.com/en-us/library/dn468673(v=vs.113).aspx

When EF tries to generate queries for inserting, updating, deleting and getting data, it’ll call your stored procedures.

Finally, you wrap your DbContext with an IUnitOfWork interface do decouple it from your application and simplify testing. Again, I’ve explained that in my YouTube video.

Reply

Hooman Bahreini says:
September 27, 2018 at 11:29 pm

DbContext keeps track of changes, so long as the application keeps the DbContext. In a Website/Disconnected environment, where each request has it’s own life cycle, you cannot rely on DbContext change tracking to save changes… To make the matter worse, DbContext is probably living in multiple servers behind a load balancer.

Reply

Clayton says:
April 6, 2017 at 6:15 pm

Hi Mosh,

I agree with you that the repository must return only domain entities. If you want to return ViewModel objects, it’s possible to create a Service layer that would be responsible for get domain entities using a repository and convert to ViewModels. Finally, the controller(mvc or webApi) could use these services and know nothing about the entities.

Leave your comments.

Have a nice day!

Reply

admin says:
April 13, 2017 at 9:25 am

Hi Clayton,

Services should not return ViewModels either! A ViewModel is part of the presentation layer which sits above the service layer. Mapping of domain objects to ViewModels is the responsibility of whoever wants to display those objects. In an ASP.NET MVC application, it’s the controller, in a WPF application it’s the Form (or Window).

Services are often misunderstood. Technically, your services should be used only for the write operations and not for reading data. It took me a little while to really grasp this. The reason is that if a controller needs some data from the database, it can simply use the repository interfaces to get the data. There is no need to go through a service layer for getting data. With this implementation, you'll end up with services methods that have 1 line of code, which is simply delegating to the repository. What's the point of these methods? Nothing but increasing the development and maintenance cost. I had a lot of services like these several years ago but eventually, I came to this realization that these methods are redundant and add absolutely no value.

Just to clarify, controller uses the repository interface (not the implementation). So it doesn't go straight to the data access layer. Repository interfaces are part of the business layer or the core domain.

Hope that helps!

Reply

Shehab says:
April 10, 2017 at 6:59 pm

Mosh,

If we return IEnumerable from the repository and the entity has a large dataset, would that be slower than returning IQueryable with a 'fatter' query inside the repository?
I think not because the service call to the repository is filtering the records before it get called but I wanted to see what you think.

Thanks in advance.

Reply

admin says:

April 13, 2017 at 9:19 am

I’m not quite sure what you mean. A fat query can get slow when dealing with large datasets. Whether you return IQueryable from your repositories (which you shouldn’t) and then do a ToList() somewhere else, or do the ToList() inside the repository (which you should), the performance is the same.

Reply

Ali Dehqan says:
November 12, 2017 at 7:03 am

Hey mosh,
There are many questions about implementing repository pattern.
Assume a table with large data and many columns with nvarcharmax columns:

1- As you said in your videos and this post:

Repository Layer : GetUsersByFilter() -> returns IEnumerable

Because we are fetching all of the columns it doesn’t seem to be efficient.
We are returning domain IEnumerable with all of the fields and constraints implemented on domain object and its fields.

2- Another way

Repository Layer : GetUsersByFilter1() -> returns IEnumerable
Repository Layer : GetUsersByFilter2() -> returns IEnumerable
Repository Layer : GetUsersByFilter3() -> returns IEnumerable
Repository Layer : GetUsersByFilter4() -> returns IEnumerable

Which doesn't return all of the UserDomain columns in each method. We can select fields which we need in select query

GetUsersByFilter1(FilterModel filterModel){

return _context.Users
.Where(//filter goes here)
.Select(x => new UserDomain{
//just set 4 of 20 columns
})
.AsEnumerable()
}

3- Another another way

Repository Layer : GetUsersByFilter() -> returns IQueryable

Service Layer: Implement multiple methods according presentation layer needs with different select queries and filters.

Conclusion:

I think the first way is not efficient in large databases.

[Reply]()

chong says:
April 11, 2017 at 1:34 am

"Instead, you should have a separate repository per domain class, like OrderRepository, ShippingRepository and ProductRepository."

Hi Mosh,

Do you mind to share your thought about Generic Repository pattern? What do you think about it?
Thanks.

Reply

admin says:
April 13, 2017 at 9:17 am

Generic repository is good as long as your methods don't method IQueryable. Otherwise, it's like you're wrapping DbSet with another class with the same interface. No point in doing that!

It's useful when you wanna get simple objects (like reference data) without having to create a separate repository for each entity.

Reply

Jarbas Carvalho says:
September 19, 2017 at 5:07 pm

Hi Mosh,

What about OData integration that needs IQueryable?

Reply

Neo says:

September 27, 2017 at 11:56 am

Great article. From the last section, I formulated the following coding convention for my team which I think is highly useful:

“For repository class methods, never return a IQueryable object. Always enumerate or convert it first (e.g., ToArray, ToList, AsEnumerable).”

The reasoning being that IQueryable would allow the caller to build on this and ultimately modify the SQL query that is executed on the database. This can potentially be dangerous in terms of DB performance, but it is more about SoC. The caller doesn’t care about the data source; it just wants the data.

Reply

Alex says:
September 28, 2017 at 7:29 pm

How would you return an entity with only specific columns (e.g. having the SQL built only query what you need) if neither the repository nor the service are ‘allowed’ to return IQueryable or DbSet ?

Reply

Rafael says:
October 26, 2017 at 6:06 pm

Hi, Mosh

” a repository is like a collection of domain objects. ”

yeah thats right and it should only return domain objects.I agree with that but I have a question

“Mapping is not the responsibility of the repository.”

but what if my domain entity is different that my data model entity?? who and where I have to make the mapping to persist the data.

Thanks!!

Reply

bugged87 says:
June 20, 2018 at 12:53 am

Mosh’s point was to avoid returning presentation layer objects such as view models in your repository interface (domain / core layer). Mapping from EF data entities to domain entities, however, would be appropriate in the repository implementation which is part of the data layer.

Reply

Even Schjølberg says:
February 15, 2018 at 9:27 am

So you say not to save/update in repositories. That sounds good, but could you answer me this:

Where should I call .SaveChanges on a DbContext when using an IoC-container?
If I do manual injection I can easily use the DbContext as a unit of work.
But I fail to see how to solve this when using something like Unity or an ASP.net DI-container.
I’m also using services which have the repositories injected.

Thanks for your time!

Reply

Prathap Chandran says:
August 6, 2018 at 7:38 am

I think Save and Update are allowed in repositories – in which case, there won't be Unit of Work implementation. This is the best approach for DI. Unit of Work works well if we don't use DI and we are ok with the coupling of UoW library being aware of changes across repositories and committing them all at one go – this won't be suitable in every scenario.

A good explanation here:
https://stackoverflow.com/questions/14263331/repository-and-unit-of-work-patterns-how-to-save-changes

@Mosh – I disagree that Save and Update don't belong in the repository. The repositories main intent is to work with collections/aggregates and perform complex queries on domain objects. That doesn't mean they should NOT have update API. As long as they don't call the storage directly, and make calls on an adapter (or an entity manager), they are perfectly good to have, where Unit of Work is not always appropriate.

Reply

Georgi Karavasilev says:
February 23, 2018 at 12:53 pm

Hello Mosh!

What about mapping in repositories when we have domain model and persistent model? Where we should do the transformation between the different type of objects? I usually have a separate folder with static classes and call it in the repository before the write methods and after the read ones.

Will be glad to hear your opinion! Greets!

Reply

Masoud Shabanloo says:
February 24, 2018 at 12:29 am

Hi mosh

I watched your Repository video and read this blog post, both more than 3 times. It's really good explained and I should say you are one of my favorite bloggers. There is something in this post made me confused; The last part of the post. You emphasized, Repository should never return IQueryable, because it results in performance issue. But I read something that sounds contradictory.
This is the confusing part:

IEnumerable: While querying data from database, IEnumerable executes select query on server side, load data in-memory on client side and then filter data. Hence does more work and becomes slow.

IQueryable: While querying data from database, IQueryable executes select query on server side with all filters. Hence does less work and becomes fast.

It's opposite of your advice. If it's true, why we should not use IQueryable instead of IEnumerable.

And something else, What about situations that we want to use OData; As you know it's better to use IQueryable instead of IEnumerable in OData queries.

please let me know your opinion.

Thank you

Reply

Muhammad Bilal says:
March 2, 2018 at 7:59 am

Hello Mosh!

What about the navigation properties that comes with an order when I execute the following query and is there any need of Include?

{code}
public Order GetOrder(Expression<Func> whereExp)
{
return _OrderRepositories.Get(whereExp);//It will get its related record as well
}
{code}

Reply

Imran Ahmad Mughal says:
March 12, 2018 at 5:30 am

Hi Mosh,
As each repository returns data for a single domain object e.g. Customer. In often cases we do need to return data from multiple tables for example Customer and its order. I know we can include Order data as navigational property but lets say If I assume that there is no relationship between these two table for the time being. Then in which repository we will write a method to return data from multiple tables?

Regards,
Imran

Reply

Henry Wyckoff says:
June 8, 2018 at 7:47 pm

Mosh — I've looked at your videos and Pluralsight courses, and I'm still confused. Given that a repository shouldn't have methods for updating an entity (say, from an Edit post action), how should one then implement something like this? Your videos say not to have an update method in a repository class, but it doesn't actually demonstrate what I should do instead.

“\`
```
_unitOfWork.Repository.Update(city);
```
“\`

which calls …

“\`
```
public void Update(City city)
{
_db.Entry(city).State = EntityState.Modified;
}
```
“\`

Reply

Laurens says:
June 10, 2018 at 9:37 am

Well, If you've made a entity and added it to the list, you can just change 'city' for example: 'city.Name = "Apeldoorn" ', that way the changes are reflected and on commit it will automatically change it (_db.Entry(city).State = EntityState.Modified; will be done automatically)

Reply

Graham Cottle says:
August 26, 2018 at 8:02 am

I am trying to work this out as well. I have an MVC app, where I want to edit records. At the moment, the only way I can see of doing in the "Post" method is to fetch the record from the db using the updated objects Id and then transfer the changes into the db record and then hit complete on there.
This doesn't feel right.
What's the best way of performing an update in an MVC situation?

Reply

Laurens says:
June 10, 2018 at 9:55 am

'Instead, you should have a separate repository per domain class, like OrderRepository, ShippingRepository and ProductRepository.'
Do you mean that the repository could be the same as the controller name as prefix?
When you say 'Order' or 'Shopping' this could very well be entitiy name, controller names or of course area (probably not). We are using the name of the entity to create a set of collections, is that the right way? or…?

Personally having doubt because when you create list or enumerable you lose the flexibility of ef linq so it returns all data (hit to the database), instead of the set you'll need. So came around searching around for some idea's want to change my thinking pattern ^.^

Reply

DalSoft says:
June 12, 2018 at 2:27 pm

The only mistake is using the repository pattern. Please don't use it. Command and Queries are the right way to go.

https://rob.conery.io/2014/03/03/repositories-and-unitofwork-are-not-a-good-idea/
https://lostechies.com/jimmybogard/2012/10/08/favor-query-objects-over-repositories/

I've had to unpick well intended repositories multiple times due to misuse, and side effects.

Reply

Filipe Guedes says:
July 11, 2018 at 6:37 pm

Hi Mosh
How did you resolve a Grouping problem ?

This is what I did : (But I'm finding this strange…)

I have a repository of documents(DocumentRepository)
It have a method called GetDocumentReport that return a new entity called DocumentReport
This new entity is the result of Groupings and Counting of the Documents

I'm in doubt of

-A new repository just for this report
-Put the grouping stuff in one layer above
-Just do what I already did

Reply

laminos says:

August 23, 2018 at 5:21 pm

Hi Mosh,

I would like to know how to use Repository Pattern in Wpf (in your example, the source code provided in Wpf Example section I didn't see any usage of it).

I mean where/when do you Instantiate UnitOfWork and where/when you release it.

Thanks a lot.

Reply

Jo says:
September 15, 2018 at 8:12 pm

Hi Mosh
I attended your ASP.NET and EF tutorials in Udemy and both were very helpful to me. Regarding the usage of Repository pattern I am bit confused. I read this article and all the comments below. In my application I mostly need a viewmodel(for joins) or a DTO(when few columns are required) instead of actual domain classes. Does it mean I have to use a provider class instead of using Repository pattern?

Reply

Callen Barton (@cal5barton) says:
September 24, 2018 at 11:12 pm

Hi Mosh,

What is your method for implementing paging on a repository? I don't want to make an expensive query to the database to retrieve the full dataset but it seems to me that paging should be handled by something other than the repository. Any though

Reply

Hooman Bahreini says:
September 27, 2018 at 11:08 am

How can you Update() an existing record, without an update method? Even DbContext which is a Repository has AddOrUpdate() method. Please see here: https://stackoverflow.com/questions/5557829/update-row-if-it-exists-else-insert-logic-with-entity-framework/5558777#5558777

Reply

ian says:
October 18, 2018 at 12:01 am

Mosh, Great stuff, One question.

How do you deal with updates vs inserts?

For example, If i am creating a new record where some primary key can be set by the caller (it happens), and i need to make sure i do not clobber an existing record, what are the options with the interface you describe? It appears that the Add method will both update and insert new records. I could add a “Contains” method but now i have to get UoW involved. I feel that there is a need for a Set method and limit the function of the Add method to inserting new records only.

Your opinion would be much appreciated, Thank you.

Reply

Andriy says:
October 18, 2018 at 10:27 am

Hi Mosh,

Ideologically I agree with your statements in "Save/Update method in repositories". However, in practice that's not always possible. All is good when your repository is backed by some SQL DB and EF, but once you try to implement it using some NoSQL DB, you're out of luck. Such databases usually have no change tracking functionality in their client libraries, so each of the Get/Add/Delete operations actually perform database operations and trying to abstract that fact away is not a good idea. Not to say that these DBs often don't support transactions over multiple document collections.
Also, one of the DDD principles is "Update only one aggregate per transaction", while unit of work tries to make possible the opposite.

So, again, it's not possible to fully abstract persistence details away, and this is why the book "Implementing domain driven design" has dedicated chapter about implementing of two types of repositories.

P.S. Cosmos DB provider for EF is currently under development, which will make life much easier when targeting this DB, but again, this is only on EF.

Reply

Mahdi Ghorbanpour says:
October 29, 2018 at 1:26 pm

Hi Mosh,
I enjoyed the article. Thanks
My question is where I should implement error handling and concurrency issues? How to set transaction IsolationLevel when needed?
Also please clarify using a service layer for reading just 3 columns of a fat entity for reporting purpose with a sample project.
Thank you

Reply

Reply

Rob Vermeulen says:
December 3, 2018 at 9:10 am

Thanks for the article. I understand your reasons but I kind of think it to be bad advice, at least on the IQueryable side of things. If I need to do complex querying I want to “linq” together the outputs of multiple repositories. If my “repos” return IEnumerable, I am returning the complete set of data queried from the database. Which can be quite large when processing tens of thousands of records. Linqing multiple of those sets together uses a lot of CPU and memory resources. If the repos returned IQueryable instead, the dataset queried from the database would be substantially smaller (because it was filtered by the whole query).

Using data providers instead of repos would be a solution but that moves application logic to the data layer. And it also makes it harder to use generics as every provider has specific implementations and function calls.

Reply

reza says:
December 10, 2018 at 9:08 am

Hi,
I resolved some of my problem.
but my question is how to use Odata when I don’t use queryable in repositories?
Thanks a lot.

Reply

reza says:
December 11, 2018 at 7:06 am

Hi Mosh,
Thanks for your article.
What is your method for implementing odata when we can't use iquaryable?

Reply

Juan says:
December 18, 2018 at 2:20 am

Hello, my issue applying repository is that I dont want to expose domain entities outside the domain, but the args/return of repository methods are entities. Since what we have to persist is the state of the entity, I created an interface for it, and the repository would deal with states instead of entities. The entity would implement the state. How do you solve this problem of publishing domain entities?

Reply

hoomanbahreini says:
December 25, 2018 at 9:37 pm

In my Opinion, this implementation of UoW pattern is violating the following SOLID principles:

1. It is violating Open/Closed Principle, as it is not really extendable… also if we add a new repository, we need to modify the UoW class to include the new Repository.
2. It is violating Interface-Segregation Principle. ISP states that clients should not be forced to implement interfaces they don't use. This UoW contains an implementation of all the repositories… but then a consuming client does not need all these implementation, the client would probably just need one of them.

3. It is also violating Single Responsibility Principle, as the UoW is responsible for so many repositories.

According to: https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?redirectedfrom=MSDN&view=entity-framework-6.2.0

“DbContext represents a combination of the Unit-Of-Work and Repository patterns and enables you to query a database and group together changes that will then be written back to the store as a unit. DbContext is conceptually similar to ObjectContext.
”

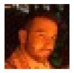
So MS is saying that DbContext is already implementing the UoW pattern… so here, we are wrapping the UoW inside another UoW…

——————————————————-

Another problem with this implementation of UoW/Repository is Update… here we don’t have any Update() method in the Repository. So how are we going update an existing record? As far as I know EF change tracking would only work when the same instance of DbContext is used to read and then update an Entity. But this is not the case in a disconnect environment (Website).

Please refer to this answer about Update using EF DbContect:
https://stackoverflow.com/questions/5557829/update-row-if-it-exists-else-insert-logic-with-entity-framework/5558777#5558777

Reply

 *Shlomo Konwisser* says:
January 8, 2019 at 6:34 pm

Mosh, how to manage n:m relationships with the Repository pattern?

1:n relationships are easy – if a User is related to many Address entries, the User class can contain a list of Address entities and the UserRepo would return such User entities including Address objects. But what if there is an n:m relation between Student and Course? Would you recommend to handle it as two 1:n relations and keep those in sync in code?

Let’s say I call user.addCourse(course) which will add a Course object to the list inside the

Let's say I call user.addCourse(course) which will add a Course object to the list inside the User object (and automatically add a User object to the list inside the Course object). Now, how do I persist this change using the Repository pattern? Do I call userRepo.update(user) and the implementation of UserRepo::update knows to call CourseRepo::update? If this cascades through my repos, it can easily result in a lot of unnecessary persistence calls and even infinite loops. Any recommendations?

Thanks!
Shlomo

Reply

Samuel Egger says:
February 22, 2019 at 10:50 am

This case should be handled by the surrounding unit of work. There should be no need to manually call any update method.

Reply

Samuel Egger says:
February 22, 2019 at 10:45 am

I am quiet curious if there is actually any reason to still use the repository pattern in a C# project? I can already hear unit testing. Well, almost all of todays ORMs including Entitiy Framework already support mocking of the underlying data source. Another reason pointed out in the comment section / article is sharing of complex queries. Simply use query objects which are possible with the power of expression trees.

Reply

Marcos says:
March 11, 2019 at 12:48 pm

I think you have to answer the questions Mosh, furthermore how will we choose the better option?

Reply

moisespongebob says:
April 6, 2019 at 7:34 pm

hi mosh do you think where is the right place for automapper in web api lets say i have service layer UserManagementService inherit IUserManagementService, and many repository use in usermanagementservice. and that service will call in your controller. Do you think is safe to implement automapper in service layer? or better in Controller.

Reply

Leave a Reply

Enter your comment here...