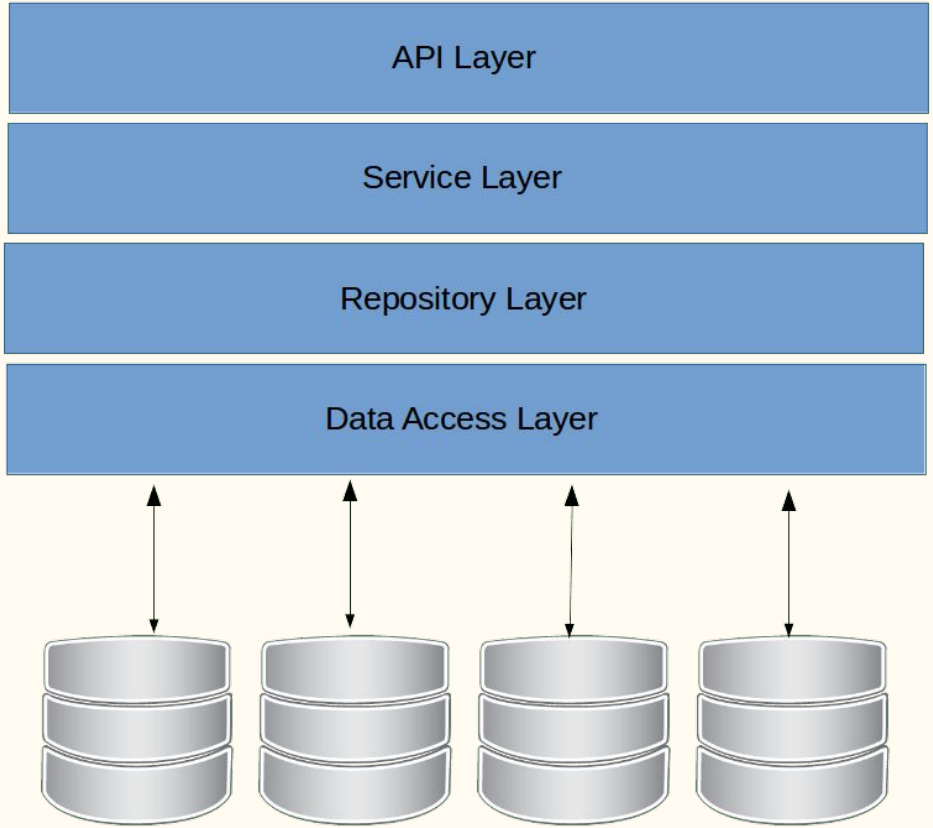
Fundamental Patterns

- Repository Pattern
- Unit of Work (UoW)
- Service


API Layer
Service Layer
Repository Layer
Data Access Layer

Repository Pattern

—

Martin Fowler

What is Repository Pattern?

“The repository Pattern mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects. It is like a object collection in memory”

Repository Pattern Goals

- Minimizes the query logic in your application avoiding queries spread by the code;
- Minimize duplicate query logic
- Decouples Business access code from data access. Then it could separate your application from persistence frameworks like EF Core
- Facilitates unit testing in your application

Methods used in the Repository Pattern

The repository pattern handles objects in memory, like this:

- Remove (entity)
- Get (id)
- IEnumerable GetAll (entity)
- Find (predicate)

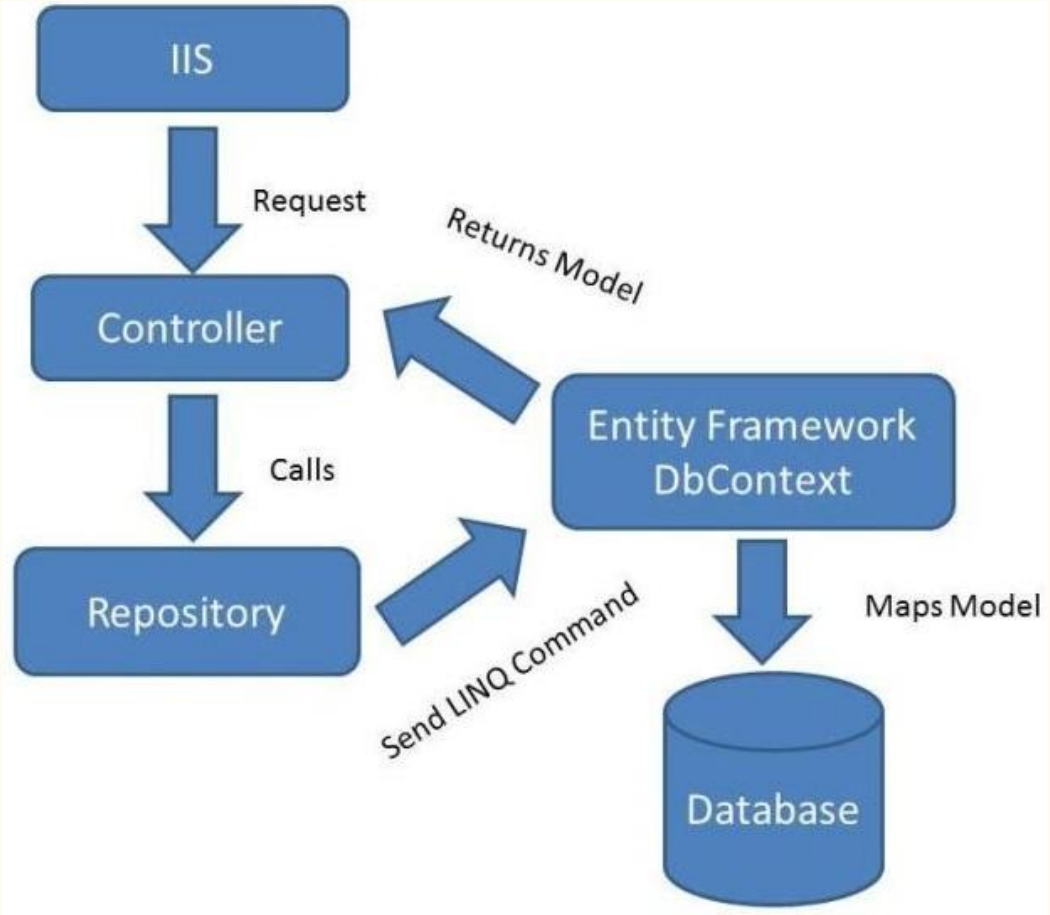
The repository pattern is NOT related to database persistence! So it is not part of the Repository Patter:

- Add (entity)
- Save (entity)
- Update (entity)

Return IEnumerable instead of IQueryable

In the repository standard, IQueryable should NOT be returned, but IEnumerable! Example:

IEnumerable queries on the client side, while IQueryable on the bank side. This prevents queries from spreading to other layers, so it does not allow you to extend or perform redundancies, giving the repository exclusivity to perform this function.

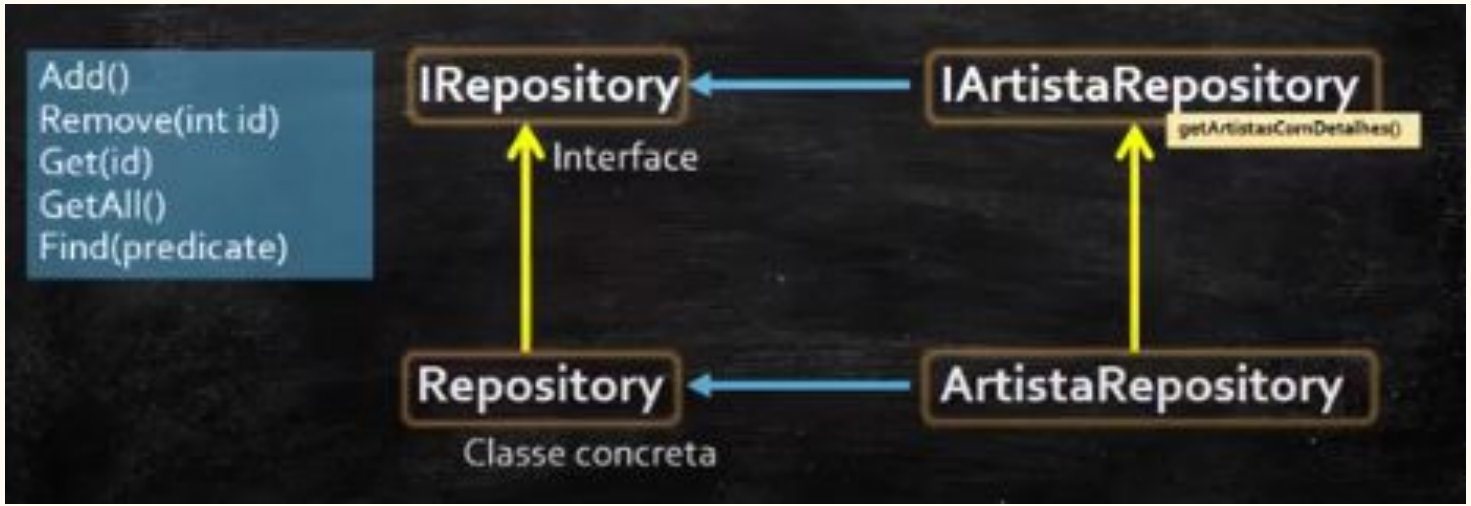

IIS
Request
Returns Model
Controller
Entity Framework
DbContext
Calls
Repository
Send LINQ Command
Maps Model
Database

Clean code for Repository Pattern

- Use generics
- Don't use one repository per domain
- Focus your repository to manage a specific entity collection
- Share the context
- Don’t return view models from ur repository
- Avoid Save/Update method in repositories (idk)
- Don’t return IQueryable

Implementation of the Repository Pattern

Generic implementation - Specific implementation

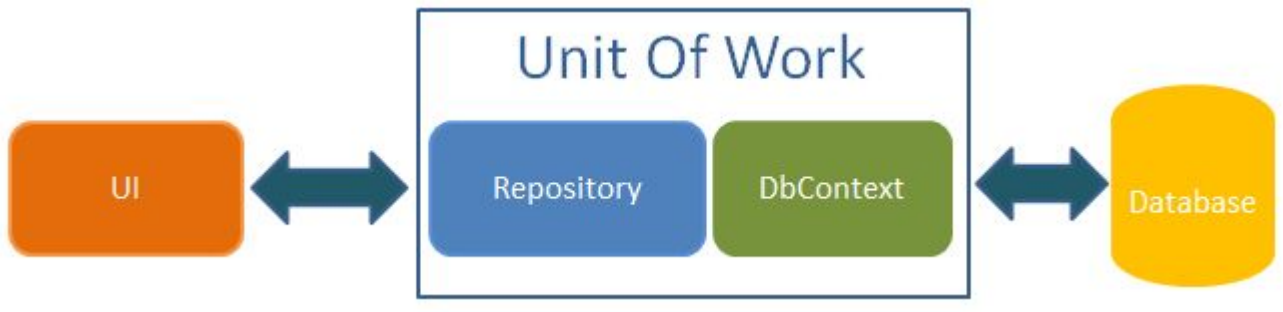


Unit Of Work Pattern

How about the entities persistence?

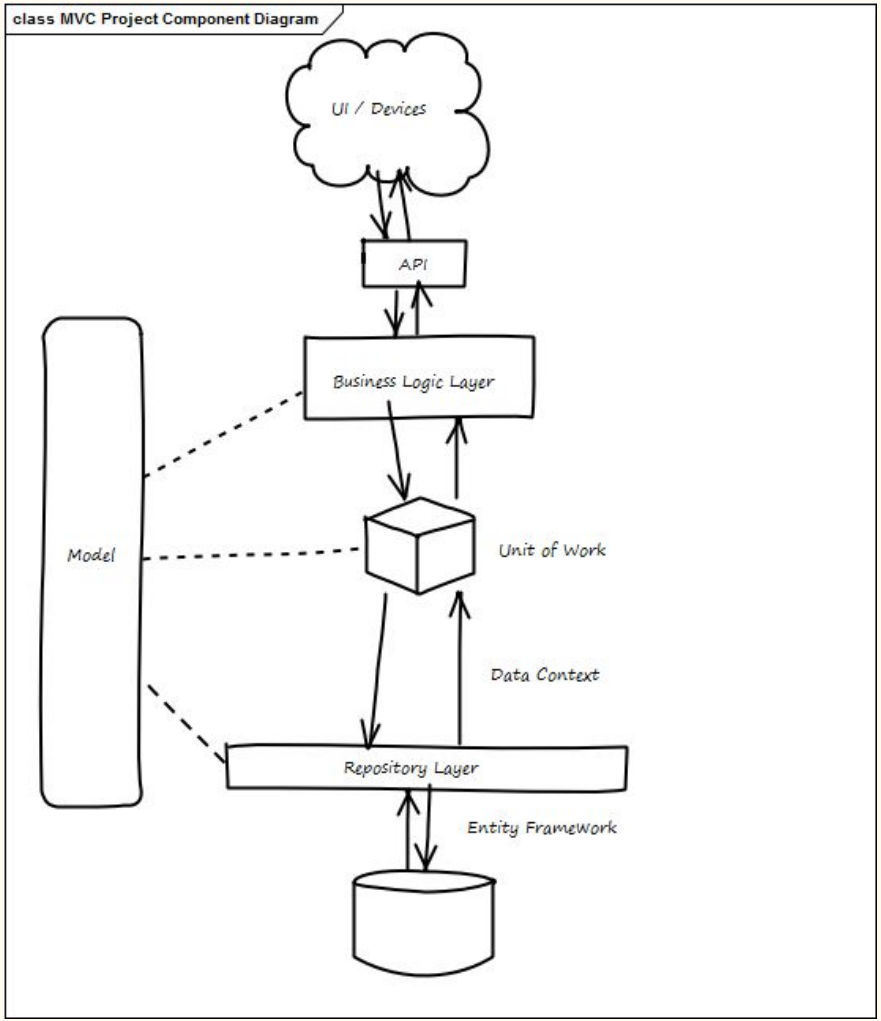
The Repository pattern deals with objects in memory. While the management of the states of the objects and the realization of their persistence is performed using the Unit Of Work Pattern.

"It keeps a list of objects affected by a transaction and coordinates the writing of changes and addresses possible competition issues".



UoW examples

- TTransaction from NHibernate
- DataContext from LINQ to SQL
- ObjectContent from Entity Framework


class MVC Project Component Diagram
UI / Devices
API
Business Logic Layer
Model
Unit of Work
Data Context
Repository Layer
Entity Framework

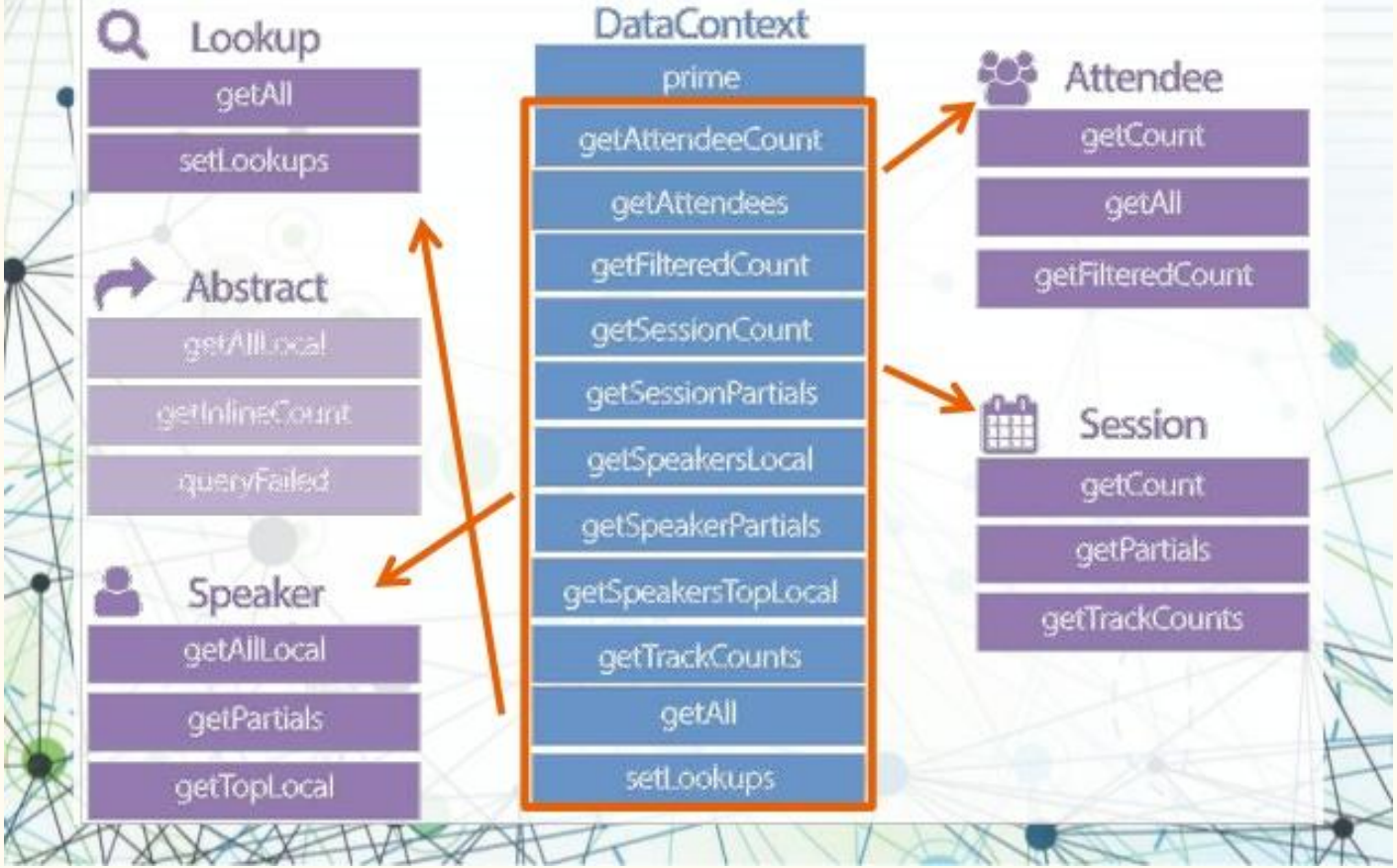
Example

Situation 1: When working with multiple repositories, but with dependent transactions, for example, if it is necessary to include the customer and then the order if the first is successful, then these are actions that have to be in the same transaction, for that it is necessary to generate a dependency by passing the same context in the repository builder.

Unit of Work Pattern Goals

- Keeps track of manipulated objects in order to synchronize data -> data store;
- Provides a single transaction for multiples queries
- The UoW commits the transaction
- Single commit call on database (single transaction)
- All object tracking information is centralized

Unit of Work & Repository Patterns



Implementation of the Unit Of Work Pattern

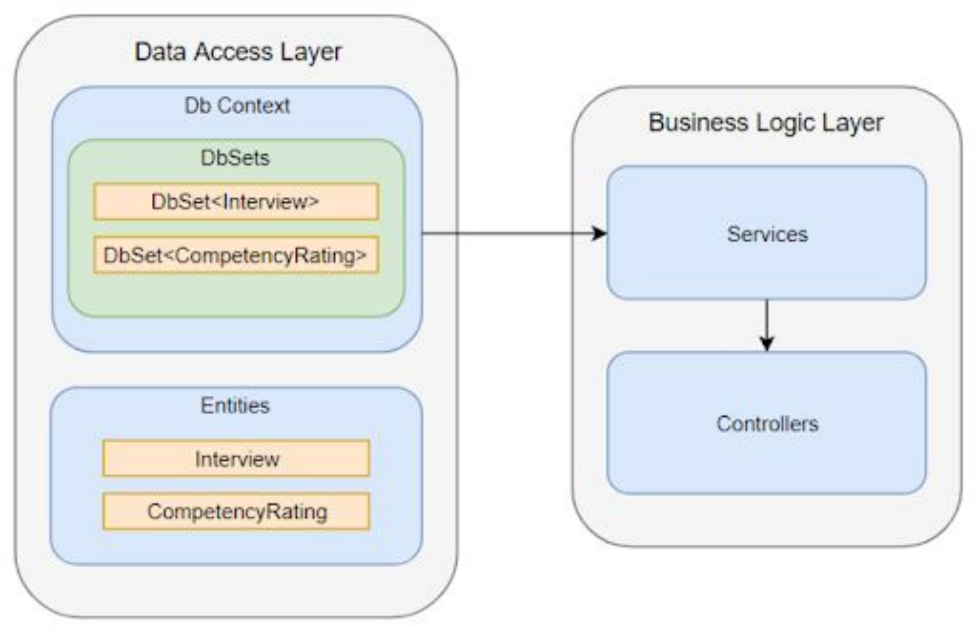
Generic implementation - Specific implementation

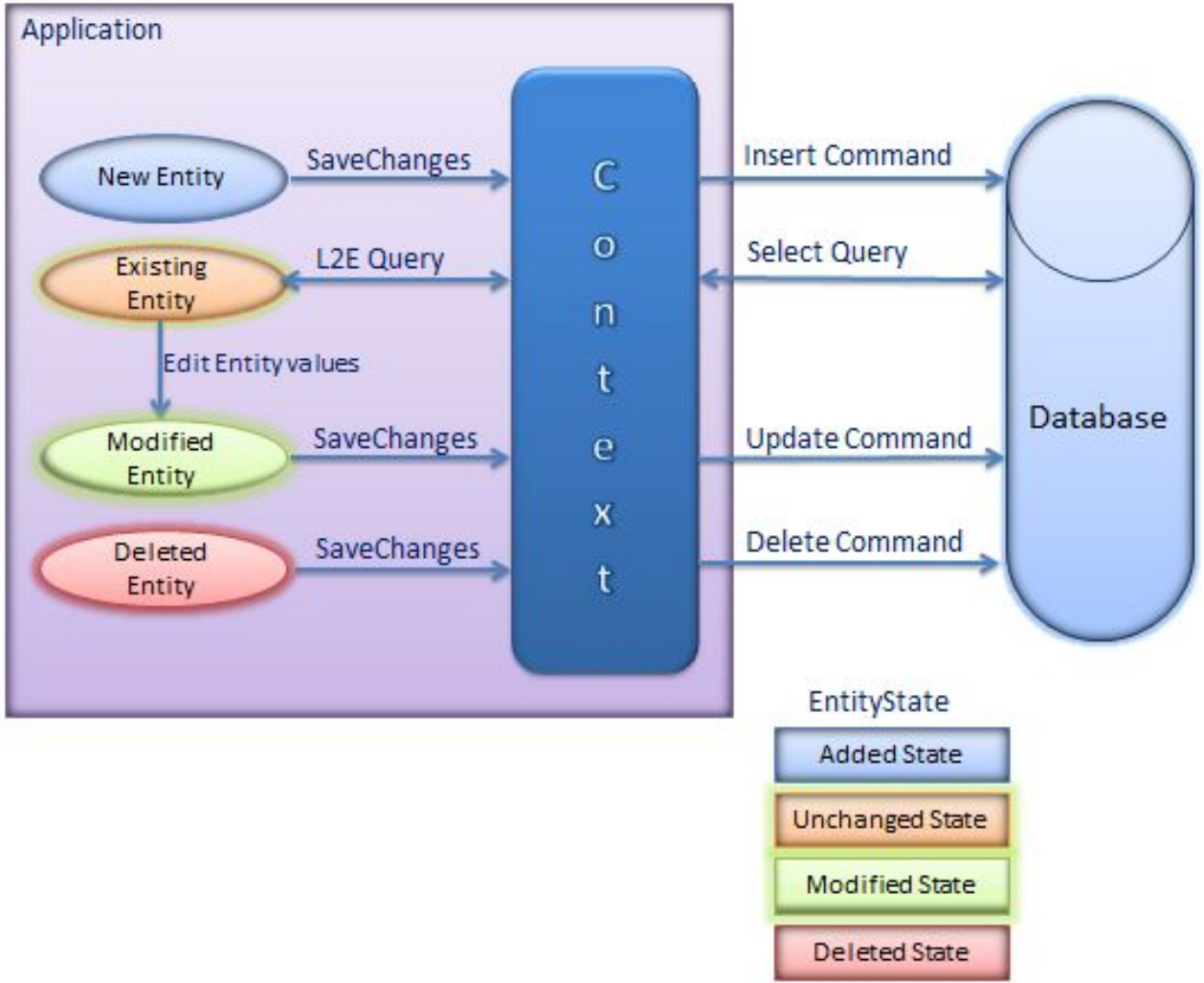
Entity Framework

Entity Framework

Usually frameworks, like the entity framework in dotnet, already implement both the Unit Of Work Pattern and the Repository Pattern.

- DbSet - Represents a collection of objects in memory (add, remove, etc.)
- DbContext - Acts as a UnitOfWork, it has DbSets and SaveChanges ()




Application
New Entity
SaveChanges
Existing Entity
L2E Query
Edit Entity values
Modified Entity
SaveChanges
Deleted Entity
SaveChanges
Context
Insert Command
Select Query
Update Command
Delete Command
Database
EntityState
Added State
Unchanged State
Modified State
Deleted State

Since EF core already provide to us the Repository Pattern and UoW, why should we make our own?

The problem with DbSet is that it returns IQueryable which makes it easier to duplicate queries by your code.

So, we understand using the standard to decouple the framework.

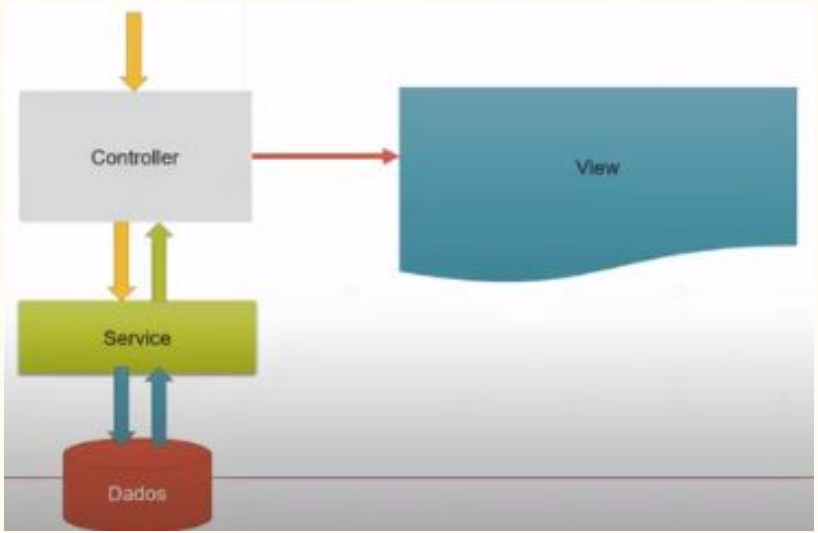
“Architecture must be independent of frameworks” - Clean Architecture

Entity Framework does not bring all the benefits of this architecture. We have to use them as tools to help include your system in limited restrictions.

Service Pattern

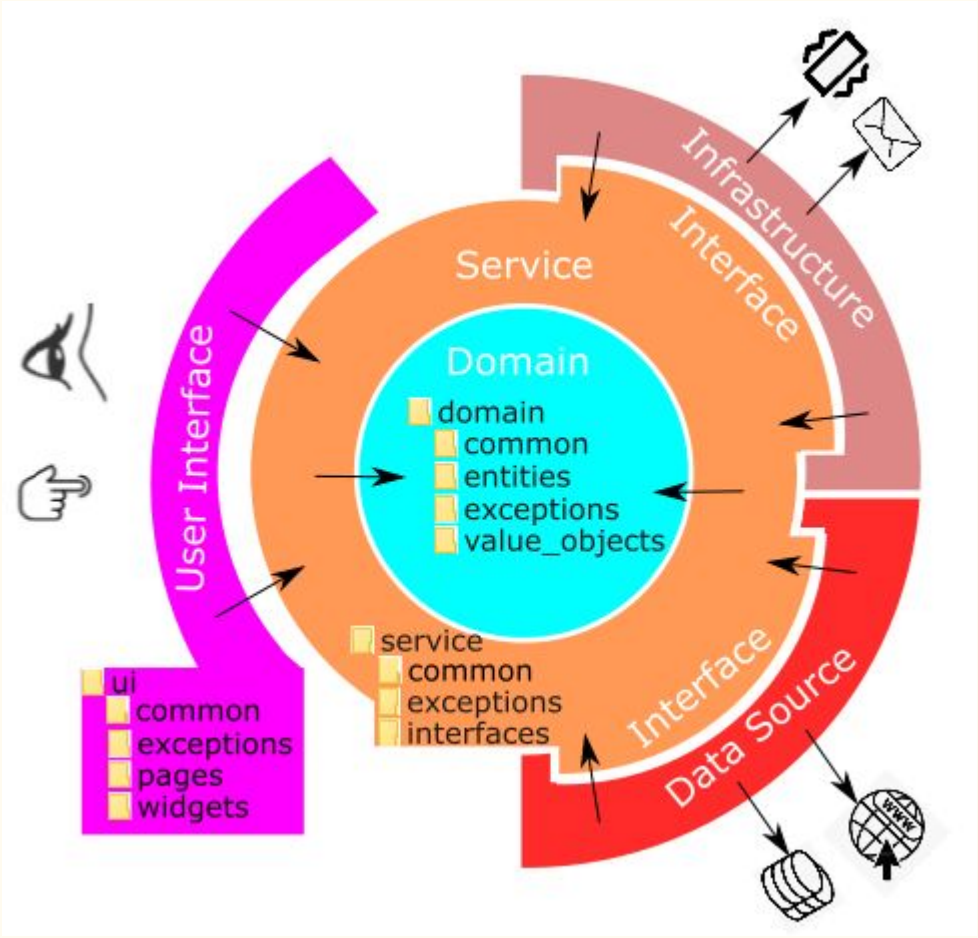
Service Layer Pattern

Introduces the separation of the service code from the application presentation layer. In this way the controller is gonna be simpler and easier to test, and if you need to make adjustments to the presentation or service, it is easier.



Service Layer Goals

- Centralizing duplicated business logic
- Allows you to organize the code by separating responsibilities
- Code that needs to be re-used by multiple clients
- Third party libraries that we have limited licenses for
- Third parties that need an integration point into our system
- Allows you to reuse the code avoiding errors and thus increase productivity
- Allows you to test the application more easily


Infrastructure
Interface
Service
Domain
domain
common
entities
exceptions
value_objects
User Interface
service
common
exceptions
interfaces
ui
common
exceptions
pages
widgets
Interface
Data Source

Implementation of the Service Layer pattern

Interface-oriented programming:

Clean code for Service Layer

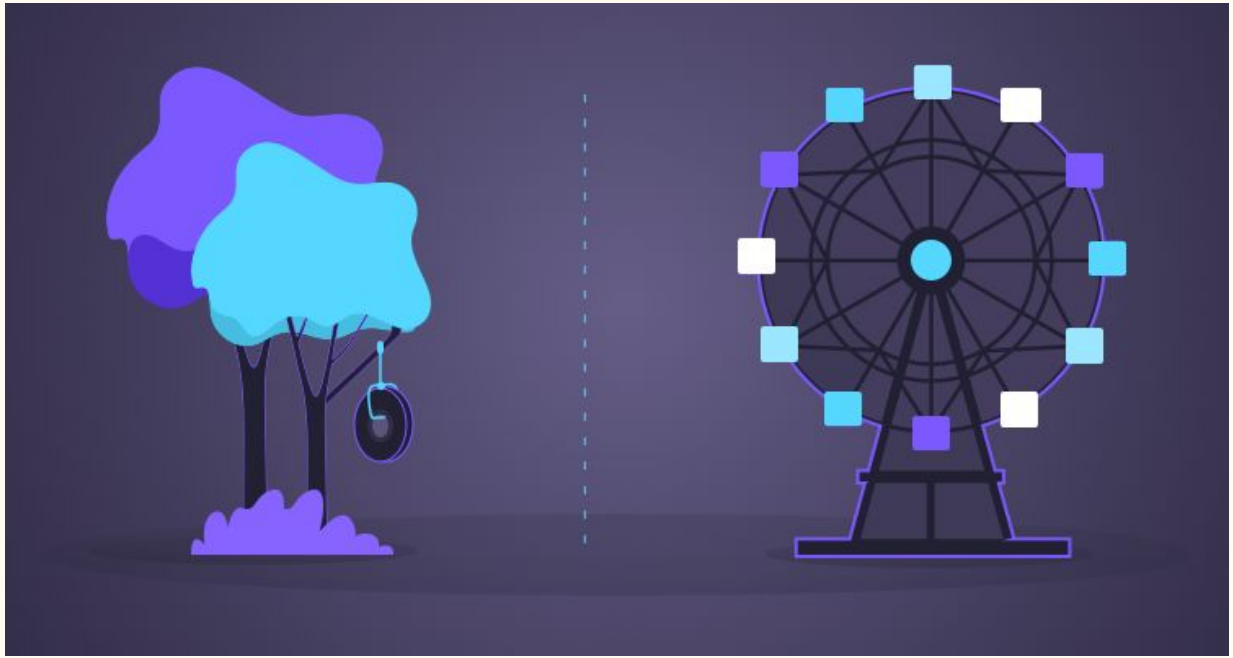
- The operation relates to a domain concept that is not a natural part of an ENTITY or VALUE OBJECT
- The interface is defined in terms of other elements of the domain model
- The operation is stateless (any client can use any instance of a particular SERVICE without regard to the instance's individual history)

Overview: Before and After

In summary

- Business objects should interface with a business oriented storage service
- Data objects should interface with a data oriented service
- The repository should be the bridging mediator
- BUT the real use case for the repository pattern is in dealing with multiple persistence services

Be aware: Use the patterns only when you need to!



TEST TIME!

- We have a real project with Entity Framework, our test is to switch EF core to PetaPoco and then Dapper.