# Repository minta

* [ASP.NET Core Web API - **Repository Pattern** - Code Maze (code-maze.com)](https://code-maze.com/net-core-web-development-part4/)

Mi az a Repository minta és miért használjuk?

A Repository mintával egy absztrakciós réteget hozunk létre az alkalmazás adathozzáférési (context) és üzleti logikai rétege között.

* Használatával egy lazábban kapcsolt megközelítést támogatunk az adatbázisból származó adataink eléréséhez.
* Emellett a kód tisztább, könnyebben karbantartható és újrafelhasználható.
* Az adatelérési logika egy külön osztályban, vagy osztályok halmazában van, amelyet tárolónak (repository) nevezünk, és amelynek feladata az alkalmazás üzleti modelljének tartós fennmaradása.

A repository minta megvalósításához a generikus osztályokat fogjuk használni:

* [C# Intermediate - Generics - Code Maze Blog (code-maze.com)](https://code-maze.com/csharp-generics/)

# Repository minta előkészítése

Vizsgáljuk meg az IStudentRepo interface-t (és ennek mintájára készült ITeacher és IParent interfaceket):

Task<List<Student>> GetAllAsync();

Task<Student?> GetByIdAsync(Guid id);

Task<ControllerResponse> UpdateStudentAsync(Student student);

Task<ControllerResponse> DeleteStudentAsync(Guid id);

Task<ControllerResponse> InsertStudentAsync(Student student);

Láthatjuk, hogy a metódusok neve és a paraméterek, visszatérési értékek megegyeznek, csak a típus, amit használunk (Student, Teacher, Parent) az ami változik. Ezért ez a kódrészlet kitűnően alkalmas generikus osztályok használatára.

Vizsgáljuk meg az osztály kódját is:

public class StudentRepo : IStudentRepo

{

private readonly KretaInMemoryContext \_dbContext;

public StudentRepo(KretaInMemoryContext dbContext)

{

\_dbContext = dbContext;

}

A StudentRepo-ba egy KretaInMemoryContext adatbázishozzáférési réteget injektálunk. Ez a context most InMemoryContext, vagyis memóriában tárolt tesztadatokkal működik. Hasznos lehetne úgy megírni a repository mintát, hogy a contextustól is független legyen. Vagyis a generikus osztálynak két típusa lesz:

* DbContext az adatbázis elérési réteghez
* Egy osztály (pl. Teacher, Student, Parent,…) amelyek az adatbázis táblájának felelnek meg.

### Kódújrafelhasználás

Ha ezt a generikus mintát kifejlesztjük, akkor egy kódot használva (kódújrafelhasználás) az összes osztályra (Student, Teacher, Parent,…) elkészíthetjük a repository mintát.

## Generikus minta

### A repository mintának megfelelő interface

public interface IRepositoryBase<T>

{

IQueryable<T> FindAll();

IQueryable<T> FindByCondition(Expression<Func<T, bool>> expression);

T GetById(Guid id);

Task<ControllerResponse> UpdateAsync(T entity);

Task<ControllerResponse> DeleteAsync(Guid id);

Task<ControllerResponse> InsertAsync(T entity);

}

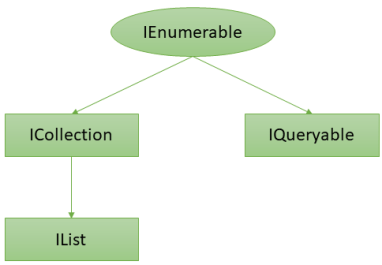
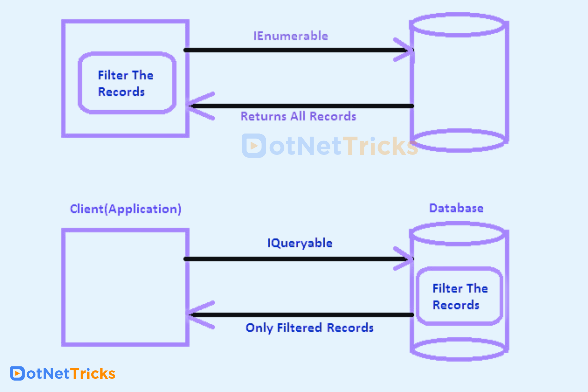
* A Student helyett a T-t írjuk. Z helyett egy tetszőleges típus, egy osztályt kell elképzelni (eddig a Student volt).
* A List<T> helyett IQueryable<T> generikus típus kerül.

[IEnumerable, ICollection, IList, and IQueryable in C# (dicodeplace.com)](https://dicodeplace.com/enumerable-icollection-ilist-and-iqueryable-in-csharp/)

IEnumerable, ICollection és IList a memóriában lévő adatokkal dolgozó gyűjtemények.

Az IQueryable a másik oldalon a külső adatforrásokban, például adatbázisokban található gyűjteményekkel való munkára szolgál.

Amikor egy IQuerable LINQ-lekérdezéséhez átad egy kódot, az futásidőben kifejezésfává alakul át, amelyet az SQL megért. Ez biztosítja számunkra az adatok szűrésének hatékony lehetőségét, közvetlenül az adatbázisban.



[IEnumerable VS IQueryable (dotnettricks.com)](https://www.dotnettricks.com/learn/linq/ienumerable-vs-iqueryable)

## Repository mintában használatos típusnak van-e id-je?

A repository mintában több olyan metódus is lesz, amely feltételezi, hogy az adatokat Guid id alapján lehet megtalálni. Ezért az osztályokat egy olyan interface-től örököltetjük, amelyben van Guid id:

public interface IDbEntity<TEntity> where TEntity : class, new()

{

public string GetDbSetName() => new TEntity().GetType().Name;

public Guid Id { get; set; }

public bool HasId => Id == Guid.Empty ? false : true;

}

## Repository minta osztály fejléce

Az repository minta osztály fejlécében látható, hogy a generikus osztály létrehozásakor egy DbContext-te és egy Guid id-vel rendelkező osztályt kell megadni:

public class RepositoryBase<TDbContext, TEntity> : IRepositoryBase<TEntity>

where TDbContext : DbContext

where TEntity : class, new()

## Konstruktor

A konstuktor létrehoz egy DbContextet és a \_dbSet változón keresztül biztosítja az adatbázis tábla elérését:

private readonly IDbContextFactory<TDbContext> \_dbContextFactory;

private DbSet<TEntity>? \_dbSet;

public RepositoryBase(IDbContextFactory<TDbContext> dbContextFactory)

{

\_dbContextFactory = dbContextFactory;

TDbContext dbContext = \_dbContextFactory.CreateDbContext();

## \_dbSet = dbContext.Set<TEntity>();

## }

## A repository minta osztály kódrésze

A repository minta osztály kódrészét az IRepositoryBase interface implementálásával kapjuk, és ezt az implementálást kell kiegészíteni a megfelelő saját kódokkal:

### FindAll

FindAll visszadja az adatbázis tábla összes adatát

public IQueryable<TEntity> FindAll()

{

if (\_dbSet is null)

{

return Enumerable.Empty<TEntity>().AsQueryable().AsNoTracking();

}

return \_dbSet.AsNoTracking();

}

### GetById

Id alapján megkeresi az entitást ha az létezik

public TEntity GetById(Guid id)

{

if (\_dbSet is null)

{

return new TEntity();

}

return \_dbSet.FirstOrDefault(entity => entity.Id == id) ?? new TEntity();

}

### FindByCondition

FindByCondition a feltételeknek megfelelő entitásokat adja:

public IQueryable<TEntity> FindByCondition(Expression<Func<TEntity, bool>> expression)

{

if (\_dbSet is null)

{

return Enumerable.Empty<TEntity>().AsQueryable().AsNoTracking();

}

return \_dbSet.Where(expression).AsNoTracking();

}

### CRUD metódusok

Az Update, Delete és Insert metódusok a StudentRepo osztályban használt metódusokra emlékeztetnek.

## Az új StudentRepo

A Student osztályt a Student.cs állományban az IDbEntity interfactől örököltetjük az Id miatt:

public class Student : IDbEntity<Student>

Az IStudentRepo inteface az IRepositoryBase interfactől öröklődik:

public interface IStudentRepo : IRepositoryBase<Student>

{

}

Az új StudentRepo kódja sokkal egyszerűbb lesz az öröklődés miatt:

public class StudentRepo<TDbConstext> : RepositoryBase<TDbConstext, Student>, IStudentRepo

where TDbConstext : DbContext

{

public StudentRepo(IDbContextFactory<TDbConstext> dbContextFactory) : base(dbContextFactory)

{

}

}

Létrehoztunk egy mintát, amelyet használva gyorsan megírhatjuk a legalapvetőbb adatbázis műveleteket a táblákhoz.

Nézzük hogyan alakul át akkor a StudentRepo osztály és IStudentRepo interface.

## KreataContex-tek

Módosítsuk a már megírt StudentRepo-t és új IStudent interfacet úgy, hogy most már a repository mintától öröklődjenek:

public **interface** IStudentRepo : IRepositoryBase<Student>

{

}

public **class** StudentRepo<TDbConstext> : RepositoryBase<TDbConstext, Student>, IStudentRepo

where TDbConstext : DbContext

{

public StudentRepo(IDbContextFactory<TDbConstext> dbContextFactory) : base(dbContextFactory)

{

}

}

Evvel azt értük el, hogy létrehozhatunk egy StudentInMemoryRepo-t úgy, hogy az a KretaInMemoryDbContext-et használja.

Később létrehozhatunk pl. a MySql vagy más adatbázisoknak egy Student repo-t ami teljesen azonosan fognak működni a mostani kódunkhoz, de más adatbázist hasnálnak.

Így függetlenítettük a kódunkat az adatbázistól

Készítsük el az új StudentInMemory repót:

public class StudentInMemoryRepo : StudentRepo<KretaInMemoryContext>, IStudentRepo

{

public StudentInMemoryRepo(IDbContextFactory<KretaInMemoryContext> dbContextFactory) : base(dbContextFactory)

{

}

}

Az alkalmazásunkba használjuk ezt a repót a Controllerben dependency injection-el:

KretaBackendExtensions.cs

public static void ConfigureRepos(this IServiceCollection services)

{

services.AddScoped<IStudentRepo, **StudentInMemoryRepo**>();

}

## Controller réteg

* GetById legyen asyncron metódus és módosítsuk a repository mintát használja
* Apró javítás kell a SelectAllRecordToListAsync metódusban is
* A CRUD műveleteket is javítsuk
* A repository rétegben javítást kell ezközölni a GetByIdAsync miatt
* Konfiguráljuk a memória adatbázist ContextFactory használatára

A kód új funkciót nem eredményez, de a repository minta segítségével gyorsan készíthetünk új repókat ugyan azt a kódot használva!