

# Szerver oldali .Net programozás

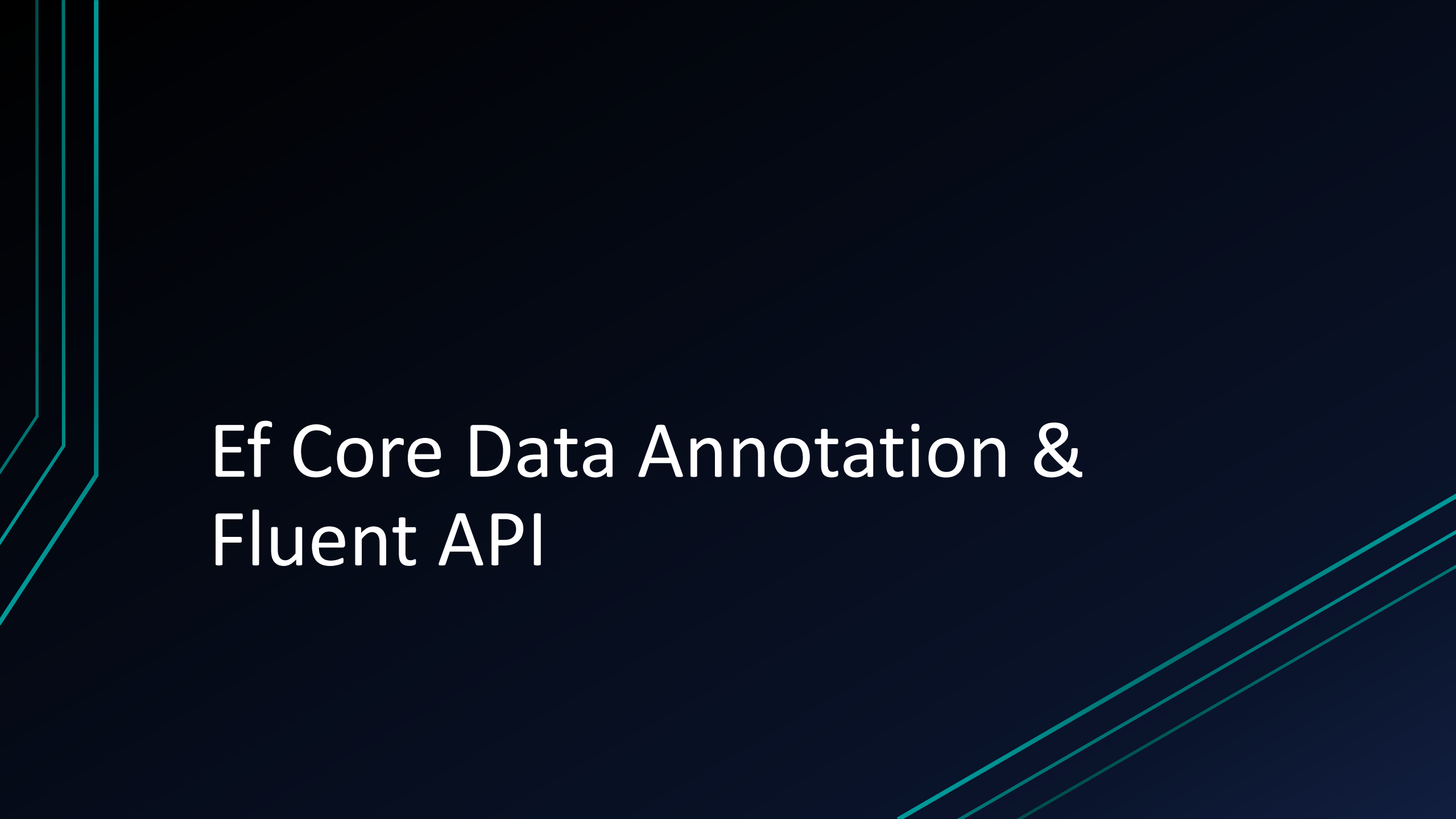
PEKÁRDY MILÁN – [PEKARDY@DCS.UNI-PANNON.HU](mailto:PEKARDY@DCS.UNI-PANNON.HU)  
HORVÁTH ÁDÁM – [HORVATH@DCS.UNI-PANNON.HU](mailto:HORVATH@DCS.UNI-PANNON.HU)

# Óra ütemezés

- 2019.09.13. 13:50-17:00 I.fsz.I2 – Bevezetés, ASP.NET Core/EF core alapok
- 2019.09.20. 13:50-17:00 I.fsz.I2 – EF Core/Repository
- 2019.10.04. 13:50-17:00 I.fsz.I2 – UnitOfWork, szolgáltatási réteg
- 2019.10.18. 13:50-17:00 I.fsz.I2 – Autentikáció/autorizáció
- 2019.11.08. 13:50-17:00 I.fsz.I2 – ASP.NET Core kiegészítő témák
- 2019.11.15. 13:50-17:00 I.fsz.I2 – Konzultáció/gyakorlati ZH

# Tartalom

- Ef Core Data Annotation & Fluent API
- Repository



# Ef Core Data Annotation & Fluent API

# Ef Core Data Annotation

- <https://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>
- `dotnet add package System.ComponentModel.DataAnnotations`
- Data Annotations attributes are .NET attributes which can be applied on an entity class or properties to override default conventions.
- **Note:** Data annotations only give you a subset of configuration options. [Fluent API](#) provides a full set of configuration options available in Code-First.

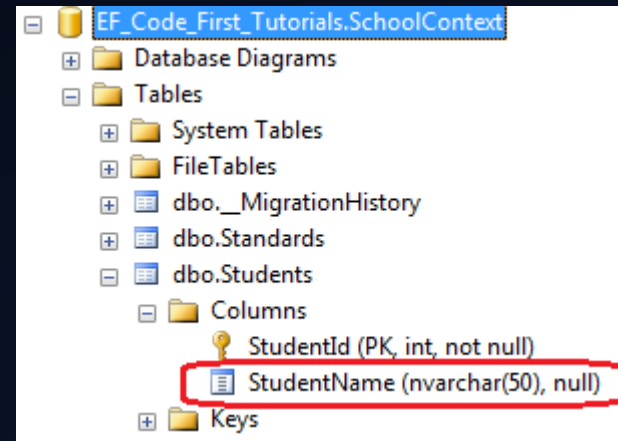
# Ef Core Data Annotation

## System.ComponentModel.DataAnnotations Attributes

Attribute	Description
<a href="#">Key</a>	Can be applied to a property to specify a key property in an entity and make the corresponding column a PrimaryKey column in the database.
<a href="#">Timestamp</a>	Can be applied to a property to specify the data type of a corresponding column in the database as <i>rowversion</i> .
<a href="#">ConcurrencyCheck</a>	Can be applied to a property to specify that the corresponding column should be included in the optimistic concurrency check.
<a href="#">Required</a>	Can be applied to a property to specify that the corresponding column is a NotNull column in the database.
<a href="#">MinLength</a>	Can be applied to a property to specify the minimum string length allowed in the corresponding column in the database.
<a href="#">MaxLength</a>	Can be applied to a property to specify the maximum string length allowed in the corresponding column in the database.
<a href="#">StringLength</a>	Can be applied to a property to specify the maximum string length allowed in the corresponding column in the database.

```
using System.ComponentModel.DataAnnotations;

public class Student
{
    public int StudentID { get; set; }
    [MaxLength(50)]
    public string StudentName { get; set; }
}
```



# Ef Core Data Annotation

## System.ComponentModel.DataAnnotations.Schema Attributes

Attribute	Description
<a href="#">Table</a>	Can be applied to an entity class to configure the corresponding table name and schema in the database.
<a href="#">Column</a>	Can be applied to a property to configure the corresponding column name, order and data type in the database.
<a href="#">Index</a>	Can be applied to a property to configure that the corresponding column should have an Index in the database. (EF 6.1 onwards only)
<a href="#">ForeignKey</a>	Can be applied to a property to mark it as a foreign key property.
<a href="#">NotMapped</a>	Can be applied to a property or entity class which should be excluded from the model and should not generate a corresponding column or table in the database.
<a href="#">DatabaseGenerated</a>	Can be applied to a property to configure how the underlying database should generate the value for the corresponding column e.g. identity, computed or none.
<a href="#">InverseProperty</a>	Can be applied to a property to specify the inverse of a navigation property that represents the other end of the same relationship.
<a href="#">ComplexType</a>	Marks the class as complex type in EF 6. EF Core 2.0 does not support this attribute.

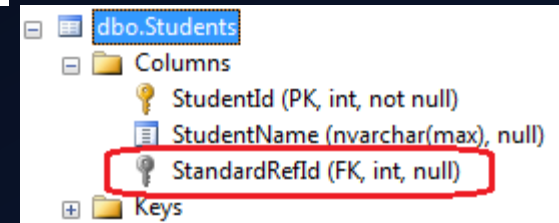
```
using System.ComponentModel.DataAnnotations.Schema;

public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }

    [ForeignKey("Standard")]
    public int StandardRefId { get; set; }
    public Standard Standard { get; set; }
}

public class Standard
{
    public int StandardId { get; set; }
    public string StandardName { get; set; }

    public ICollection<Student> Students { get; set; }
}
```



# Ef Core Fluent API

- <https://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx>
- Entity Framework Fluent API is used to configure domain classes to override conventions. EF Fluent API is based on a Fluent API design pattern (a.k.a [Fluent Interface](#)) where the result is formulated by [method chaining](#).
- To write Fluent API configurations, override the OnModelCreating() method of DbContext in a context class.



# Ef Core Fluent API

- You can use Data Annotation attributes and Fluent API at the same time.
- Fluent API configures the following aspect of a model in Entity Framework:
  - Model-wide Configuration: Configures the default Schema, entities to be excluded in mapping, etc.
  - Entity Configuration: Configures entity to table and relationship mappings e.g. PrimaryKey, Index, table name, one-to-one, one-to-many, many-to-many etc.
  - Property Configuration: Configures property to column mappings e.g. column name, nullability, Foreignkey, data type, concurrency column, etc.

```
public class SchoolContext: DbContext
{
    public DbSet<Student> Students { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //Write Fluent API configurations here
    }
}
```

# Ef Core Fluent API

Configurations	Fluent API Methods	Usage
Model-wide Configurations	HasDefaultSchema()	Specifies the default database schema.
	ComplexType()	Configures the class as complex type.
Entity Configurations	HasIndex()	Configures the index property for the entity type.
	HasKey()	Configures the primary key property for the entity type.
	HasMany()	Configures the Many relationship for one-to-many or many-to-many relationships.
	HasOptional()	Configures an optional relationship which will create a nullable foreign key in the database.
	HasRequired()	Configures the required relationship which will create a non-nullable foreign key column in the database.
	Ignore()	Configures that the class or property should not be mapped to a table or column.
	Map()	Allows advanced configuration related to how the entity is mapped to the database schema.
	MapToStoredProcedures()	Configures the entity type to use INSERT, UPDATE and DELETE stored procedures.
	ToTable()	Configures the table name for the entity.

Property Configurations	HasColumnAnnotation()	Sets an annotation in the model for the database column used to store the property.
	IsRequired()	Configures the property to be required on SaveChanges().
	IsConcurrencyToken()	Configures the property to be used as an optimistic concurrency token.
	IsOptional()	Configures the property to be optional which will create a nullable column in the database.
	HasParameterName()	Configures the name of the parameter used in the stored procedure for the property.
	HasDatabaseGeneratedOption()	Configures how the value will be generated for the corresponding column in the database e.g. computed, identity or none.
	HasColumnOrder()	Configures the order of the database column used to store the property.
	HasColumnType()	Configures the data type of the corresponding column of a property in the database.
	HasColumnName()	Configures the corresponding column name of a property in the database.
	IsConcurrencyToken()	Configures the property to be used as an optimistic concurrency token.

## Ef Core Fluent API feladatok

- Készítsen új indexet a Person táblához, az index a Name és DateOfBirth property-ket használja.
- Nevezze át az Event osztályban található PlaceId property-t, PlaceIdentity-re
- Írja felül a konvención alapuló egy-több kapcsolatot az Event és a Place osztályok között.

## Ef Core Fluent API feladatok

- Készítsen egy People listát a Person osztályba Friends néven.
- Adja meg a Fluent API-ban az adatbázis kapcsolatot ehhez a listához. Mi a probléma?

# Ef Core Fluent API feladatok

- Készítsen egy People listát a Person osztályba Friends néven.
- Adja meg a Fluent API-ban az adatbázis kapcsolatot ehhez a listához. **Mi a probléma?**
- Megoldás:
  - Új kapcsoló osztály bevezetése.
  - Törlés esetén a Cascade törlés kikapcsolása.

```
public class Friend
{
    1 reference
    public int Id { get; set; }
    1 reference
    public int PersonId { get; set; }
    1 reference
    public Person Person { get; set; }
    0 references
    public int FriendPersonId { get; set; }
    0 references
    public Person FriendPerson { get; set; }
}
```

```
modelBuilder.Entity<Friend>().HasKey(f => new {f.Id});
modelBuilder.Entity<Friend>().HasOne(f => f.Person).WithMany(p => p.Friends).HasForeignKey(f => f.PersonId);
```

## Ef Core Fluent API feladatok

- Alakítsuk úgy át az entitásokat, hogy egy esemény tartalmazza a szervezőinek adatait (Név, telefonszám).
  - Egy eseménynek több szervezője is lehet,
  - Egy szervező több eseménynek lehet a szervezője.

# Ef Core Fluent API feladatok

- Alakítsuk úgy át az entitásokat, hogy egy esemény tartalmazza a szervezőinek adatait (Név, telefonszám).
  - Egy eseménynek több szervezője is lehet,
  - Egy szervező több eseménynek lehet a szervezője.
- Megoldás:
  - EventStaff osztály létrehozása a több-több kapcsolat felírására.

```
modelBuilder.Entity<EventStaff>().HasKey(es => new { es.EventId, es.OrganizerId });  
modelBuilder.Entity<EventStaff>().HasOne(es => es.Event).WithMany(e => e.Staff).HasForeignKey(es => es.EventId);  
modelBuilder.Entity<EventStaff>().HasOne(es => es.Organizer).WithMany(o => o.Events).HasForeignKey(es => es.OrganizerId);
```

# Repository

[HTTPS://CODINGBLAST.COM/ENTITY-FRAMEWORK-CORE-GENERIC-REPOSITORY/](https://codingblast.com/entity-framework-core-generic-repository/)



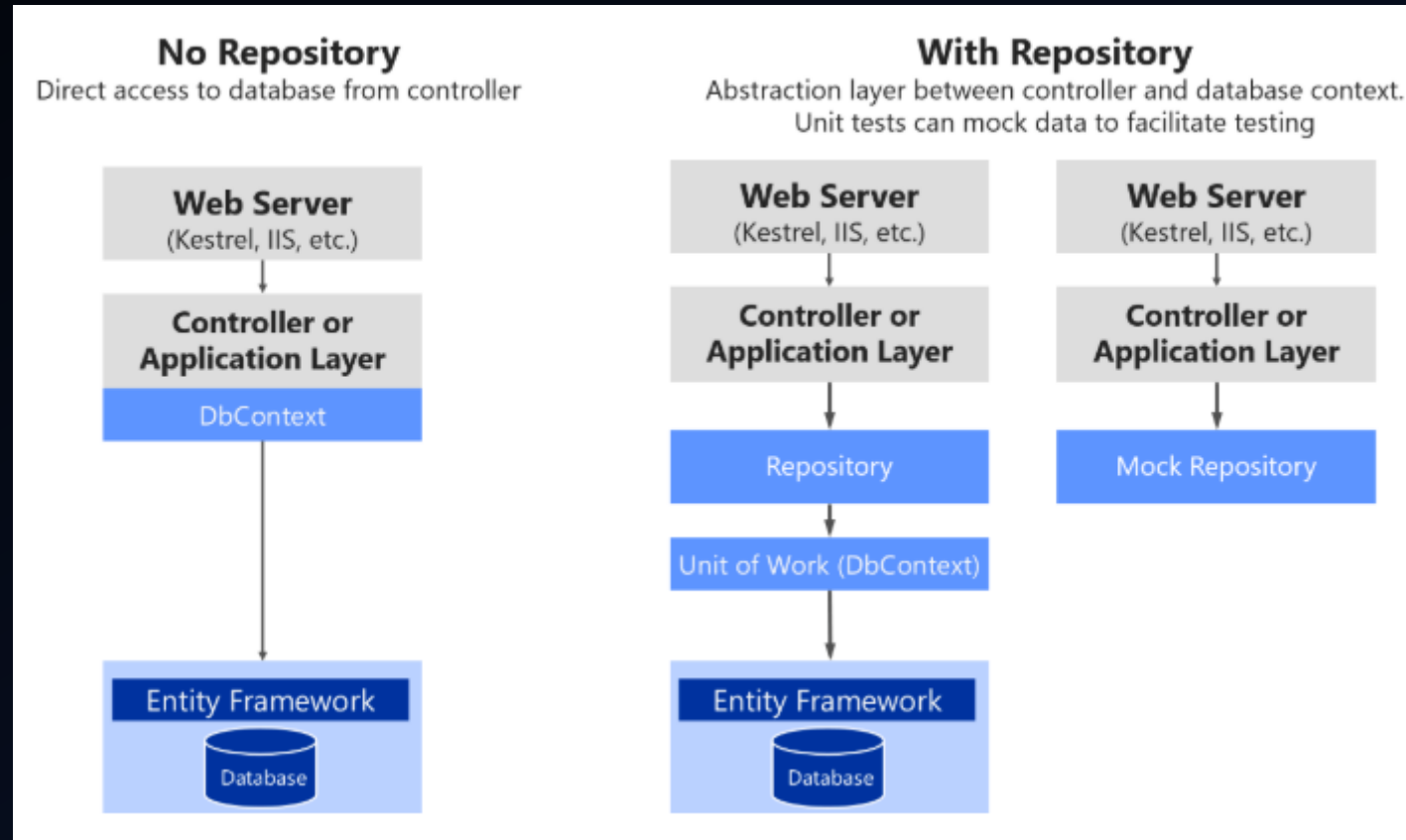
# Repository

- As with everything, the generic repository pattern has its pros and cons. You are the one to decide if it's a good fit for your project.
- The advantage of having generic CRUD repository is that you can inherit from it, pass it entity type and you have CRUD repository for any entity type with a minimal amount of code.
- We will not build something bullet-proof that will cover all of your needs in future. Rather, we will try to build a base for a generic repository that you will be able to use to create CRUD operations easily and later extend per your needs.
- For anything additional, you could inherit from GenericRepository class and extend it.
- In larger real-world applications you don't want your Web layer to be aware of your Database layer. Hence, you will not be injecting repositories in your controllers.
- You could only use it for parts of your application. You don't have to use it as a de-facto solution for all of your database needs.

# Repository pros and cons

- Pros:
  - Separation of concerns; the application need not know about or track any or all data sources.
  - The Repository pattern helps to isolate both the service and the list access code. Isolation makes it easier to treat them as independent services and to replace them with mock objects in unit tests. Typically, it is difficult to unit test the repositories themselves, so it is often better to write integration tests for them.
  - DRY (Dont Repeat Yourself) design, the code to query and fetch data from data source(s) is not repeated.
- Cons:
  - Adds another layer of abstraction which adds a certain level of complexity making it an overkill for small applications.
  - If you are caching data in heavily loaded systems, performance can be an issue. Consider synchronizing access to the data source. This ensures that only a single request for the data is issued to the list or back-end service. All other clients rely on the retrieved data.
- <https://miafish.wordpress.com/2014/12/31/repository-pros-and-cons/>

# Repository



## Ef Core Repository

- We use DbContext to query a database and group together changes that will be written back to the store as a unit.
- The great thing about DbContext class is that it's generic and therefore supports generics on methods that we will use to interact with the database.

## . Ef Core Repository

- The base interface that we will use looks like this:

```
1 public interface IGenericRepository<TEntity>
2     where TEntity : class, <span class="pl-en">IEntity</span>
3 {
4     IQueryable<TEntity> GetAll();
5
6     Task<TEntity> GetById(int id);
7
8     Task Create(TEntity entity);
9
10    Task Update(int id, TEntity entity);
11
12    Task Delete(int id);
13 }
```

- Let's start implementing generic repository!

# Tracking vs. No-Tracking Queries

- Tracking behavior controls whether or not Entity Framework Core will keep information about an entity instance in its change tracker. If an entity is tracked, any changes detected in the entity will be persisted to the database during `SaveChanges()`. Entity Framework Core will also fix-up navigation properties between entities that are obtained from a tracking query and entities that were previously loaded into the `DbContext` instance.
- Tracking queries

```
public Task<TEntity> GetById(int id)
{
    return DbSet.FirstOrDefaultAsync(e => e.Id == id);
}
```

- No-tracking queries

```
public Task<TEntity> GetById(int id)
{
    return DbSet.AsNoTracking().FirstOrDefaultAsync(e => e.Id == id);
}
```

# Ef Core Repository

- Feladat:
  - Az előző Repository-t felhasználva készítsünk olyan repository-t, amelynek egy funkciója visszaadja az 1990 után született embereket az adatbázisból.

# Ef Core Repository

- Feladat:
  - Az előző Repository-t felhasználva készítsünk olyan repository-t, amelynek egy funkciója visszaadja az 1990 után született embereket az adatbázisból.
- Megoldás:
  - Készítsünk egy interfészt az új repository-hoz, majd valósítsuk meg.

```
public interface IPersonRepository : IRepository<Person>
{
    0 references
    IQueryable<Person> GetAfter1990People();
}
```

```
public class PersonRepository : Repository<Person>, IPersonRepository
{
    0 references
    public PersonRepository(EventAppDbContext ctx) : base(ctx)
    {
    }
    0 references
    public IQueryable<Person> GetAfter1990People()
    {
        DateTime year1990 = new DateTime(1990,1,1);
        return DbSet.AsNoTracking().Where(p => p.DateOfBirth >= year1990);
    }
}
```



# Ef Core repository

- For us to be able to use PersonRepository inside of our code we have to add it to DI container

```
services.AddScoped<IPersonRepository, PersonRepository>();
```

- Now we can inject it in the service and call from controller

```
public class PersonService : IPersonService
{
    10 references
    private readonly EventAppDbContext _context;
    2 references
    private readonly IPersonRepository _personRepository;

    0 references
    public PersonService(EventAppDbContext context, IPersonRepository personRepository){
        _context = context;
        _personRepository = personRepository;
    }
}
```

```
public IQueryable<Person> GetAfter1990()
{
    return _personRepository.GetAfter1990People();
}
```

## Ef Core repository feladatok

- A Repository osztályt egészítsük ki egy olyan függvénnnyel, ahol meg tudjuk mondani, hogy az objektumoknak milyen kritériumnak kell megfelelniük(Expression), mely függőségeit kérjük le (IncludableQueryable), meg tudjuk adni, hogy mi alapján rendezze a találatokat (OrderedQueryable), és hogy track-elje az objektumokat.

# Ef Core repository feladatok

- A Repository osztályt egészítsük ki egy olyan függvénnnyel, ahol meg tudjuk mondani, hogy az objektumoknak milyen kritériumnak kell megfelelniük(Expression), mely függőségeit kérjük le (IncludableQueryable), meg tudjuk adni, hogy mi alapján rendezze a találatokat (OrderedQueryable), és hogy track-elje az objektumokat.

```
IQueryable<TEntity> GetAsQueryable(  
    Expression<Func<TEntity, bool>> predicate = null,  
    Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,  
    Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,  
    bool disableTracking = true);
```

# Ef Core repository feladatok

```
public IQueryable<TEntity> GetAsQueryable(
    Expression<Func<TEntity, bool>> predicate = null,
    Func<IQueryable<TEntity>, IQueryable<TEntity>> include = null,
    Func<IQueryable<TEntity>, IQueryable<TEntity>> orderBy = null,
    bool disableTracking = true)
{
    IQueryable<TEntity> query = DbSet;

    if (disableTracking)
    {
        query = query.AsNoTracking();
    }

    if (include != null)
    {
        query = include(query);
    }

    if (predicate != null)
    {
        query = query.Where(predicate);
    }

    if (orderBy != null)
    {
        query = orderBy(query);
    }

    return query;
}
```

## Ef Core repository feladatok

- A Repository osztályt egészítsük ki egy olyan függvénnel, az objektumot id alapján kérjük le, meg tudjuk adni, hogy mely függőségeit kérjük le (IncludableQueryable), és hogy track-elje az objektumokat.
- A Repository osztályt egészítsük ki egy olyan függvénnel, ahol meg tudjuk mondani, hogy az objektumoknak milyen kritériumnak kell megfelelniük(Expression).
- A Repository osztályt egészítsük ki olyan függvénnel, amellyel egyszerre több objektumot tudunk beszúrni az adatbázisba.
- A Repository osztályt egészítsük ki olyan függvénnel, amellyel egyszerre több objektumot tudunk frissíteni az adatbázisban.

## Ef Core repository feladatok

- A Repository osztályt egészítsük ki olyan függvénnnyel, amellyel egyszerre több objektumot tudunk törölni az adatbázisból.
- A Repository osztályt egészítsük ki olyan függvénnnyel, amely megvizsgálja, hogy a megadott kritériumnak megfelelő rekord létezik-e az adatbázisban.
- A Repository osztályt egészítsük ki olyan függvénnnyel, amely meghívásakor törli az adott tábla teljes tartalmát (Truncate és DeleteAll). Mi a különbség a kettő között?