

# EUVIC:

THE GOOD *People*

**.NET**  
**EntityFramework**  
**Part 1**

# ORM

## Czym jest ORM

ORM jest to technologia, która mapuje struktury bazy danych na klasy występujące w obiektowych językach programowania.

## Przykłady technologii ORM dla dotneta

EntityFramework, Nhibernate, Dapper

# EntityFramework



## Wyróżnione możliwości EntityFramework

- Zmapowanie tabel bazodanowych na klasy
- Zmapowanie kolumn tabel na properties
- Mapowanie relacji bazodanowych
- Mapowanie widoków bazodanowych na klasy
- Generowanie migracji
- Wbudowana transakcyjność

# EntityFramework

---

## Słownik pojęć

- **Encja** – klasa reprezentująca tabelę
- **DbContext** – klasa która jest źródłem dostępu do bazy danych. DbContext implementuje wzorzec „Unit of Work”.
- **Unit of work** – wzorzec który zbiera zmiany z repozytoriów i wykonuje je w jednej tranzakcji
- **DbSet** – „Repozytorium” które jest przeznaczone dla pojedynczej encji
- **Model** – Reprezenacja bazy danych jako zestawu encji znajdujących się w DbContextcie
- **Migracja** – kod który reprezentuje zmianę w modelu której wynikiem będzie zmiana bazy danych
- **Model Snapshot** – aktualny stan naszego modelu
- **Migration Snapshot** – poprzedni stan modelu który był przed wygenerowaniem migracji



# DbContext

## Przykładowy DbContext

```
public class StaffTrainingContext : DbContext
{
    0 references
    public StaffTrainingContext(DbContextOptions<StaffTrainingContext> options)
        : base(options)
    {
    }

    0 references
    DbSet<Attendee> Attendees { get; set; }
    0 references
    DbSet<Lecturer> Lecturers { get; set; }
    0 references
    DbSet<Technology> Technologies { get; set; }
    0 references
    DbSet<Training> Trainings { get; set; }
}
```

- Encje nazywamy jako liczba pojedyncza np. „Attendee”
- DbSet nazywamy jako liczba mnoga np. „Attendees”. Nazwa DbSet będzie automatycznie nazwą tabeli po wygenerowaniu migracji

# DbContext

## Rejestracja DbContext

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<StaffTrainingContext>(
        options => options.UseSqlServer(Configuration.GetConnectionString("Sql")));
}

"ConnectionStrings": {
    "Sql": "Server=(localdb)\\mssqllocaldb;Database=StaffTraining"
}
```

- Aby móc zarejestrować DbContext potrzebujemy ConnectionString do bazy danych.
- Na potrzeby lokalne warto skorzystać z serwera „(localdb)\\mssqllocaldb”
- Connection String zawsze dostarczy nam usługodawca w przypadku wykupionego hostingu dotneta

# Konfiguracja encji

- Bardzo polecam aby skorzystać z interfacu [IEntityTypeConfiguration](#), aby skonfigurować każdą encję
- Przeniesienie konfiguracji do osobnego pliku wprowadza porządek
- Konfiguracja może być w innym projekcie niż klasy domeny (encje), przez co projekt domeny, nie musi mieć zależności na EntityFramework

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    // -- Bulk registration
    modelBuilder.ApplyConfigurationsFromAssembly(
        Assembly.GetAssembly(typeof(StaffTrainingContext))
    );

    // -- Single registration
    modelBuilder.ApplyConfiguration(new AttendeeConfiguration());
}
```

```
public class AttendeeConfiguration : IEntityTypeConfiguration<Attendee>
{
    0 references
    public void Configure(EntityTypeBuilder<Attendee> builder)
    {
        builder.HasKey(e => e.Id);
        builder.Property(x => x.Firstname)
            .HasMaxLength(50)
            .IsRequired();

        builder.Property(x => x.Lastname)
            .HasMaxLength(50)
            .IsRequired();
    }
}
```

# Relacje

## Konfiguracja „One To Many”

```
public class TrainingConfiguration : IEntityTypeConfiguration<Training>
{
    0 references
    public void Configure(EntityTypeBuilder<Training> builder)
    {
        builder.HasOne(x => x.Lecturer)
                .WithMany(x => x.Trainings)
                .HasForeignKey(x => x.LecturerId);
    }
}
```

```
public class Lecturer
{
    1 reference
    public long Id { get; set; }
    1 reference
    public string Firstname { get; set; }
    1 reference
    public string Lastname { get; set; }
    1 reference
    public ICollection<Training> Trainings { get; set; }
}
```

```
public class Training
{
    2 references
    public Guid Id { get; set; }
    1 reference
    public string Title { get; set; }
    1 reference
    public long LecturerId { get; set; }
    1 reference
    public Lecturer Lecturer { get; set; }
}
```



# Relacje

## Konfiguracja „Many To Many”

```
public class AttendeeConfiguration : IEntityTypeConfiguration<Attendee>
{
    0 references
    public void Configure(EntityTypeBuilder<Attendee> builder)
    {
        builder.HasMany(x=>x.Trainings)
                .WithMany(x=>x.Attendees);
    }
}
```

```
public class Attendee
{
    1 reference
    public long Id { get; set; }
    1 reference
    public string Firstname { get; set; }
    1 reference
    public string Lastname { get; set; }
    0 references
    public DateTime CreateDate { get; set; }
    0 references
    public DateTime UpdateDate { get; set; }
    2 references
    public ICollection<Training> Trainings { get; set; }
}
```

```
public class Training
{
    0 references
    public long Id { get; set; }
    0 references
    public string Title { get; set; }
    0 references
    public string Description { get; set; }
    1 reference
    public long LecturerId { get; set; }
    1 reference
    public Lecturer Lecturer { get; set; }
    2 references
    public ICollection<Attendee> Attendees { get; set; }
}
```

# Relacje

## Konfiguracja „Many To Many”

```
public class TrainingAttendeeConfiguration : IEntityTypeConfiguration<TrainingAttendee>
{
    0 references
    public void Configure(EntityTypeBuilder<TrainingAttendee> builder)
    {
        builder.HasKey(x => new { x.TrainingId, x.AttendeeId });

        builder.HasOne(x => x.Training)
            .WithMany(x => x.Attendees)
            .HasForeignKey(x => x.TrainingId);

        builder.HasOne(x => x.Attendee)
            .WithMany(x => x.Trainings)
            .HasForeignKey(x => x.AttendeeId);
    }
}

public class Training
{
    0 references
    public long Id { get; set; }
    0 references
    public string Title { get; set; }
    1 reference
    public ICollection<TrainingAttendee> Attendees { get; set; }
}
```

```
public class TrainingAttendee
{
    2 references
    public long TrainingId { get; set; }
    2 references
    public long AttendeeId { get; set; }
    1 reference
    public TrainingAttendeeStatus Status { get; set; }

    1 reference
    public Training Training { get; set; }
    2 references
    public Attendee Attendee { get; set; }
}

public class Attendee
{
    1 reference
    public long Id { get; set; }
    1 reference
    public string Firstname { get; set; }
    1 reference
    public string Lastname { get; set; }

    1 reference
    public ICollection<TrainingAttendee> Trainings { get; set; }
}
```

# Relacje

## Konfiguracja „One To One”

```
internal class UserConfiguration : IEntityTypeConfiguration<User>
{
    0 references
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.HasKey(x => x.Id);
        builder.HasOne(x => x.Settings)
            .WithOne()
            .HasForeignKey<UserSettings>(x => x.UserId);
    }
}
```

```
public class User
{
    1 reference
    public long Id { get; set; }
    0 references
    public string Email { get; set; }

    1 reference
    public UserSettings Settings { get; set; }
}
```

```
public class UserSettings
{
    1 reference
    public long Id { get; set; }
    0 references
    public string Language { get; set; }
    1 reference
    public long UserId { get; set; }
}
```

# Relacje

## Konfiguracja „Owns One”

```
public class User
{
    1 reference
    public long Id { get; set; }
    0 references
    public string Email { get; set; }

    1 reference
    public UserProfile Profile { get; set; }
}

1 reference
public class UserProfile
{
    0 references
    public bool EnableNotifications { get; set; }
}
```

```
internal class UserConfiguration : IEntityTypeConfiguration<User>
{
    0 references
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.HasKey(x => x.Id);

        builder.OwnsOne(x => x.Profile);
    }
}
```

identity.Users

Columns

- Id (PK, bigint, not null)
- Email (nvarchar(max), null)
- Profile\_EnableNotifications (bit, null)



# Ignorowanie properties

- Domyślnie EntityFramework generuje migracje dla każdego publicznego property
- Czasem się zdarza, że chcemy stworzyć property, które oblicza coś z innego property
- W takim przypadku musimy zignorować takie property w konfiguracji

```
public class Lecturer
{
    1 reference
    public long Id { get; set; }
    1 reference
    public string Firstname { get; set; }
    1 reference
    public string Lastname { get; set; }

    2 references
    public ICollection<Training> Trainings { get; set; }
    1 reference
    public int TrainingsCount => Trainings?.Count ?? 0;
}
```

```
public class LecturerConfiguration : IEntityTypeConfiguration<Lecturer>
{
    0 references
    public void Configure(EntityTypeBuilder<Lecturer> builder)
    {
        builder.HasKey(e => e.Id);

        builder.Ignore(x => x.TrainingsCount);
    }
}
```

# Mapowanie widoków

```
public class AttendeesSummaryConfigurations : IEntityTypeConfiguration<AttendeeSummary>
{
```

0 references

```
public void Configure(EntityTypeBuilder<AttendeeSummary> builder)
```

```
{
```

```
    builder.HasNoKey();
```

```
    builder.ToView("dbo.AttendeeSummary");
```

```
}
```

```
}
```

```
public class AttendeeSummary
```

```
{
```

0 references

```
public long AttendeeId { get; set; }
```

0 references

```
public int TotalHours { get; set; }
```

```
}
```

```
public partial class AddViewAttendeesSummary : Migration
```

```
{
```

0 references

```
protected override void Up(MigrationBuilder migrationBuilder)
```

```
{
```

```
    migrationBuilder.Sql(
```

```
        @"
```

```
        CREATE VIEW [dbo].[AttendeeSummary]
```

```
        AS
```

```
        SELECT    dbo.Attendees.Id AS AttendeeId, SUM(dbo.Trainings.Duration) AS TotalHours
```

```
        FROM      dbo.Attendees
```

```
        INNER JOIN dbo.TrainingAttendee ON dbo.Attendees.Id = dbo.TrainingAttendee.AttendeeId
```

```
        INNER JOIN dbo.Trainings ON dbo.TrainingAttendee.TrainingId = dbo.Trainings.Id
```

```
        GROUP BY dbo.Attendees.Id
```

```
        ");
```

```
}
```

0 references

```
protected override void Down(MigrationBuilder migrationBuilder)
```

```
{
```

```
    migrationBuilder.Sql("DROP VIEW [dbo].[AttendeeSummary]");
```

```
}
```

```
}
```

# Słowniki

---

## Słowniki (Lookups, Dictionaries)

- Zawsze w aplikacjach mamy dane słownikowe, które często nie są dodawane przez użytkowników tylko są ewentualnie dodawane przez programistów
- Typowym przykładem takiego słownika są statusy jakiegoś stanu w aplikacji

W takim przypadku mamy dwa podejścia:

- Możemy dodać stworzyć encje, którą możemy zaseedować danymi w migracjach
- Możemy stworzyć enum oraz zrobić converter



# Relacje – podejście z Encją

```
public class TrainingAttendeeStatus
{
    5 references
    public int Id { get; set; }
    4 references
    public string Name { get; set; }
}

public class TrainingAttendee
{
    2 references
    public Guid TrainingId { get; set; }
    2 references
    public long AttendeeId { get; set; }
    2 references
    public int StatusId { get; set; }

    1 reference
    public TrainingAttendeeStatus Status { get; set; }
    1 reference
    public Training Training { get; set; }
    2 references
    public Attendee Attendee { get; set; }
}

public void Configure(EntityTypeBuilder<TrainingAttendeeStatus> builder)
{
    builder.HasKey(x => x.Id);

    builder.HasData(TrainingAttendeeStatusesSeed.GetSeed());
}
```

```
public static class TrainingAttendeeStatusesSeed
{
    1 reference
    public static IEnumerable<TrainingAttendeeStatus> GetSeed() =>
        new List<TrainingAttendeeStatus>()
        {
            new TrainingAttendeeStatus()
            {
                Id = (int)TrainingAttendeeStatuses.Interested,
                Name = TrainingAttendeeStatuses.Interested.ToString()
            },
            new TrainingAttendeeStatus()
            {
                Id = (int)TrainingAttendeeStatuses.Confirmed,
                Name = TrainingAttendeeStatuses.Confirmed.ToString()
            },
            new TrainingAttendeeStatus()
            {
                Id = (int)TrainingAttendeeStatuses.Declined,
                Name = TrainingAttendeeStatuses.Declined.ToString()
            }
        };
}
```



# Słowniki – podejście z enumem

```
public enum TechnologyScope
{
    Backend = 1,
    Frontend = 2,
    DevOps = 3,
}
```

```
public class Technology
{
    1 reference
    public long Id { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public TechnologyScope Scope { get; set; }
}
```

```
public class TechnologyConfiguration : IEntityTypeConfiguration<Technology>
{
    0 references
    public void Configure(EntityTypeBuilder<Technology> builder)
    {
        builder.HasKey(e => e.Id);

        builder.Property(x => x.Scope).HasConversion(
            v => v.ToString(),
            v => (TechnologyScope)Enum.Parse(typeof(TechnologyScope), v));
    }
}
```

	Id	Name	Scope	CreateDate	UpdateDate
1	2	EntityFramework	Backend	2022-09-05 00:00:00.0000000	2022-09-05 00:00:00.0000000

# Migracje

## Co to są migracje

Migracje jest to mechanizm entity frameworka, który pozwala w łatwy sposób trzymać spójność modelu z bazą danych.

Każda zmiana w modelu, która będzie powodowała zmianę bazy danych musi być zakończona wygenerowaniem migracji

Migracje pozwalają utrzymać kontrolę w jakiej „wersji” jest aktualnie baza na środowisku lub lokalnie

# Migracje

## Kilka porad o migracjach

- Po wygenerowaniu pierwszej migracji, stworzy się nam folder „Migrations” w „root” projektu, folder ten możemy przenieść w dowolne miejsce w projekcie i następne migracje będą już generowały się właśnie tam.
- Każda z migracji zawiera nazwę migracji, polecam nazywać migracje aby odzwierciedlały to co rzeczywiście zawierają
- W folderze migrations znajduje się plik „ModelSnapshot.cs” który reprezentuje aktualny stan modelu
- Dodatkowo każda z migracji zawiera swój plik snapshot który jest wersją „ModelSnapshot” zanim został on zmieniony w związku z nową migracją
- Migracje możemy cofać do dowolnego punktu w przeszłości

# Migracje

## Generowanie migracji THE GOOD People

Aby wygenerować migracje należy wykonać komendę:

```
Package Manager Console
Package source: All [v] [g] Default project: Euvic.StaffTraining.WebAPI [v]
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 6.1.0.106

Type 'get-help NuGet' to see all available NuGet commands.

PM> Add-Migration MigrationName
```

W przypadku gdy jest więcej niż 1 context musi mieć dodatkowy parameter -Context

```
PM> Add-Migration MigrationName -Context StaffTrainingReadOnlyContext|
```

Gdy context znajduje się w innym projekcie niż projekt uruchomiony (WebAPI) wtedy musimy wybrać „Default project”

```
Package Manager Console
Package source: All [v] [g] Default project: Euvic.StaffTraining.Identity [v]
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any li
Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 6.1.0.106

Type 'get-help NuGet' to see all available NuGet commands.

PM> Add-Migration MigrationName -Context IdentityContext|
```



# Migracje

## Kasowanie migracji

Aby skasować migracje należy wykonać komendę:

```
Package Manager Console
Package source: All [gear icon] Default project: Euvic.StaffTraining.WebAPI
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 6.1.0.106

Type 'get-help NuGet' to see all available NuGet commands.

PM> Remove-Migration|
```

W przypadku gdy jest wiele contextów trzeba dodać parameter -Context

```
PM> Remove-Migration -Context StaffTrainingContext|
```

Nie możemy usunąć migracji, która została już zaaplikowana na bazę danych, w takim przypadku musimy przywrócić bazę do migracji poprzedzającej tą którą chcemy usunąć

# Migracje

## Aplikowanie migracji

Aby zaaplikować migracje należy wykonać komendę:

```
Package Manager Console
Package source: All [v] [g] Default project: Euvic.StaffTraining.WebAPI [v]
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 6.1.0.106

Type 'get-help NuGet' to see all available NuGet commands.

PM> Update-Database
```

W przypadku gdy jest wiele contextów trzeba dodać parameter -Context

```
PM> Update-Database -Context StaffTrainingContext
```

Gdy potrzebujemy cofnąć bazę danych do jednej z poprzednich migracji należy wykonać

```
PM> Update-Database PreviousMigrationName -Context StaffTrainingContext
```

# Migracje

## Automigracje

Poniżej przedstawiam extension method, które pozwala na automigracje na potrzeby lokalnego środowiska

```
public static async void Migrate<TDbContext>(this IApplicationBuilder applicationBuilder)
    where TDbContext : DbContext
{
    // AUTOMIGRATION ARE ALLOWED ONLY FOR LOCAL DEVELOPMENT

    var scopeFactory = applicationBuilder
        .ApplicationServices
        .GetRequiredService<IServiceScopeFactory>();

    using var scope = scopeFactory.CreateScope();
    using var dbContext = scope.ServiceProvider.GetRequiredService<TDbContext>();

    dbContext.Database.Migrate();
}
```

Uzycie wygląda następująco

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.Migrate<StaffTrainingContext>();
        app.Migrate<StaffTrainingReadonlyContext>();
        app.Migrate<IdentityContext>();
    }
}
```



# Migracje

## Migracje na środowisku

Automigracje nie nadają się do wykonania na środowisku z co najmniej 2 powodów:

- Gdy aplikacja będzie miała więcej niż 1 instancję migrację odpalą się w tym samym momencie i może się to skończyć błędem, ponieważ jedna z migracji będzie aplikowała ten sam skrypt SQL jako druga a poprzednia instancja zmieniła już bazę danych czego efektem będzie exception.
- Warto aby osoba aplikująca migracje świadomie przejrzała zmiany skryptu migracji, aby nie została zaaplikowana jakaś zmiana która ma nieoczekiwane konsekwencje na produkcji

Komenda wygenerowania skryptu migracji:

```
PM> Script-Migration --idempotent -Context StaffTrainingContext
```

Flaga `--idempotent` powoduje, że migracja która już istnieje na bazie danych zostanie pominięta





# Migracje

## Wersjonowanie Migracji

- Po zaaplikowaniu migracji na bazę danych stworzy się nam tabelka „\_\_EFMigrationsHistory” która będzie zawierała historię zaaplikowanych migracji.
- Dzięki tej tabelce mamy kontrolę w jakiej wersji jest obecnie nasza baza danych i jakich migracji jeszcze brakuje

## Jak to wygląda w bazie danych

+  dbo.\_EFMigrationsHistory  
+  identity.\_EFMigrationsHistory

	MigrationId	ProductVersion
1	20220508192540_AddViewAttendeesSummary	6.0.4
2	20220508192759_InitialMigration	6.0.4

# Migracje

## Osobny schemat dla MigrationHistory

- Bardzo polecam, aby stworzyć sobie osobne „MigrationHistory” dla każdego schematu bazy danych
- W naszym przypadku oddzieliłem „identity” od schematu „dbo”
- Aby stworzyć migration history dla innego schematu bazy danych należy dodać dodatkowy parameter do „UseSqlServer”

```
public static void AddIdentity(this IServiceCollection services, string connectionString)
{
    services.AddDbContext<IdentityContext>(
        options => options.UseSqlServer(connectionString,
            config => config.MigrationsHistoryTable(
                HistoryRepository.DefaultTableName, IdentityContext.Schema)));
}
```

# EUVIC:

THE GOOD *People*

[www.euvic.com](http://www.euvic.com)

Przewozowa 32 | 44-100 Gliwice  
+48 32 279 49 42 | [info@euvic.pl](mailto:info@euvic.pl)