

Entity framework Core



Entity framework core

- Entity framework core je objekt-relacioni mapper za .NET Core.
- Mapira redove u tabelama baze podataka na objekte klasa modela i obrnuto.
- Omogućava nam interakciju sa bazom podataka bez pisanja SQL upita i parsiranja odgovora.
- Podacima se pristupa preko LINQ izraza koje EF prebacuje interno u SQL upite, zbog toga je bitno znati da je u sistemima gde je kritična brzina bolje pisati SQL procedure ručno.
- Klasa DbContext se mapira na celu bazu dok se individualni DbSet-ovi mapiraju na tabele u bazi.
- DbContext je implementacija repository i Unit of work softverskih šablona.



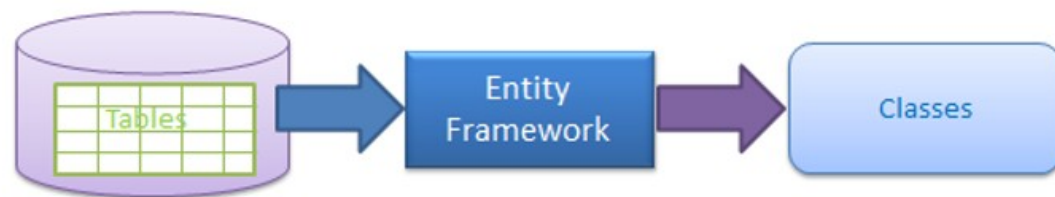
Potrebni NuGet paketi

- Microsoft.EntityFrameworkCore, verzije idu u skladu sa verzijom .NET core, u našem slučaju sve sa 5.xx dolazi u obzir
- Microsoft.EntityFrameworkCore.SqlServer, ako koristimo SQL server bazu
- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.Tools
- AutoMapper

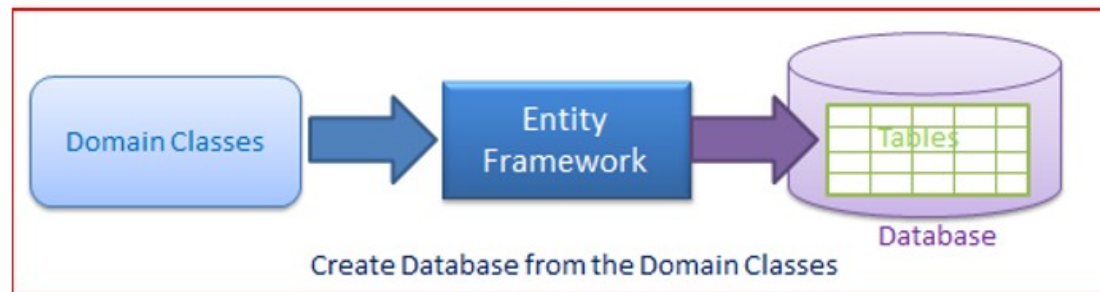


Procesi rada u EF

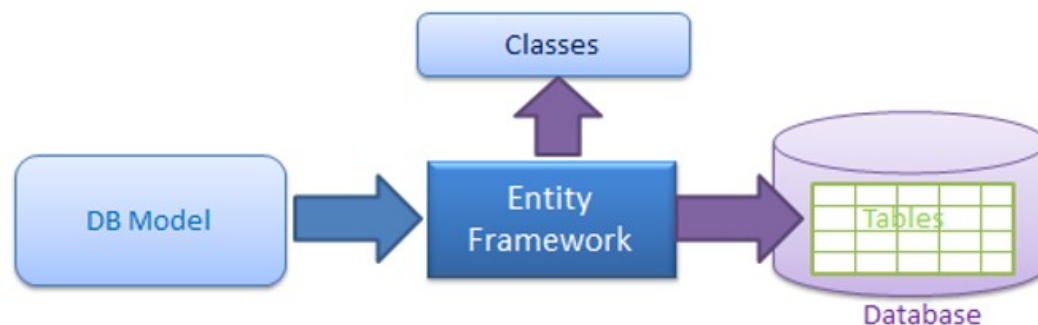
- Code-First: Prvo kreiramo klase, pa na osnovu njih kreiramo bazu podataka
- Database-First: Iz postojeće baze podataka, kreiramo klase u programu
- Model-First: Nekim alatom za modelovanje kreiramo model klasa i na osnovu njega se generišu klase i baza podataka



Generate Data Access Classes for Existing Database



Create Database from the Domain Classes



Create Database and Classes from the DB Model design

Code-First

- Počinjemo pisanjem naših domenskih klasa
- Prilikom pokretanja aplikacije EF kreira bazu podataka (ukoliko ne postoji) i mapira domenske klase na tabele u bazi podataka

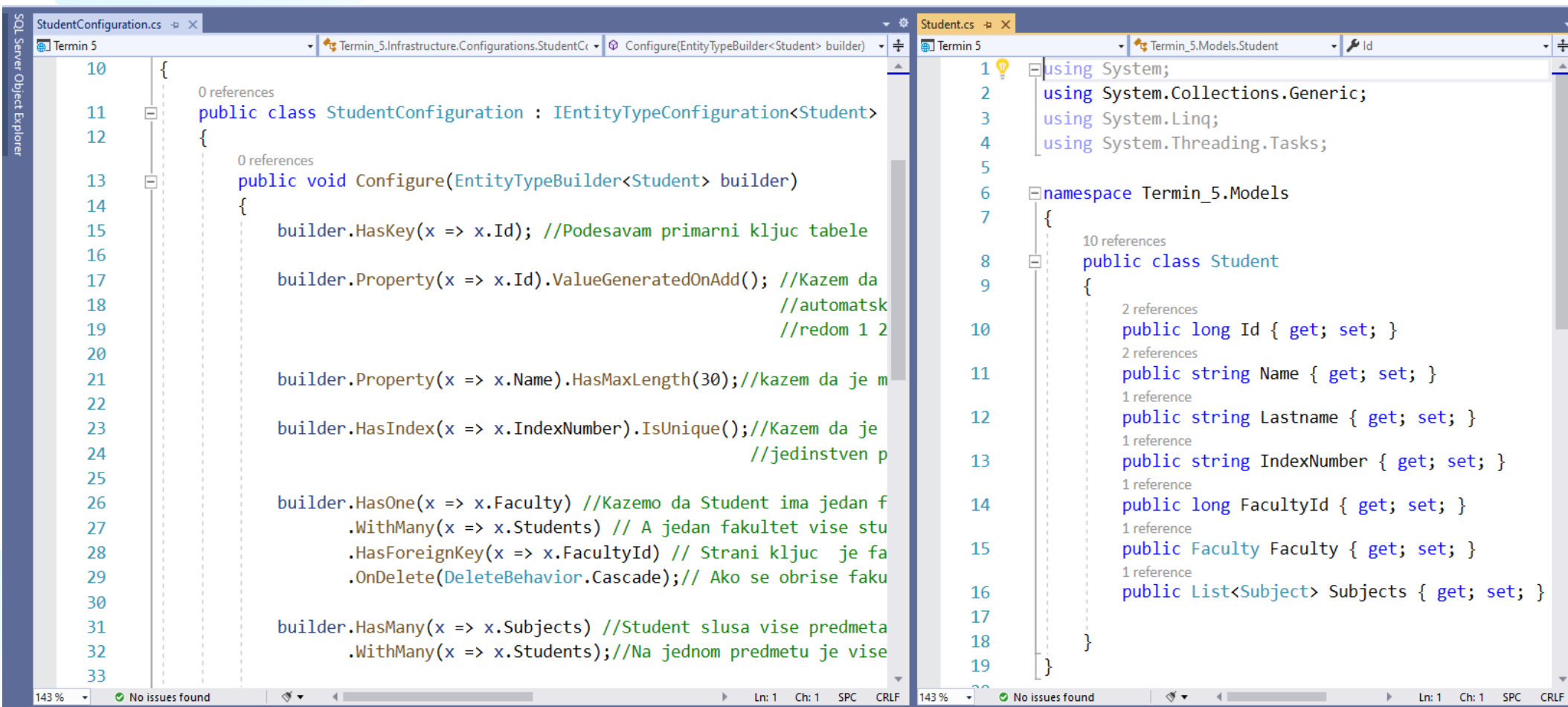


Konfiguracije modela



Konfiguracije modela

- Svaka tabela u bazi podataka mora biti predstavljena klasom modelom. Klasa modela treba da ima polja koja predstavljaju kolone u tabeli u bazi.
- Kako bi Entity Framework znao kakvu tabelu treba da kreira i kakve su veze između tabela moramo mu to nekako reći – kreirajući klase konfiguracije za svaki model.



```
StudentConfiguration.cs
10 {
11     0 references
12     public class StudentConfiguration : IEntityTypeConfiguration<Student>
13     {
14         0 references
15         public void Configure(EntityTypeBuilder<Student> builder)
16         {
17             builder.HasKey(x => x.Id); //Podesavam primarni kljuc tabele
18             builder.Property(x => x.Id).ValueGeneratedOnAdd(); //Kazem da
19                                     //automatski
20                                     //redom 1 2
21             builder.Property(x => x.Name).HasMaxLength(30); //kazem da je m
22             builder.HasIndex(x => x.IndexNumber).IsUnique(); //Kazem da je
23                                     //jedinstven p
24             builder.HasOne(x => x.Faculty) //Kazemo da Student ima jedan f
25                 .WithMany(x => x.Students) // A jedan fakultet vise stu
26                 .HasForeignKey(x => x.FacultyId) // Strani kljuc je fa
27                 .OnDelete(DeleteBehavior.Cascade); // Ako se obrise faku
28             builder.HasMany(x => x.Subjects) //Student slusa vise predmeta
29                 .WithMany(x => x.Students); //Na jednom predmetu je vise
30
31
32
33
34 }
```

```
Student.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Models
7 {
8     10 references
9     public class Student
10     {
11         2 references
12         public long Id { get; set; }
13         2 references
14         public string Name { get; set; }
15         1 reference
16         public string Lastname { get; set; }
17         1 reference
18         public string IndexNumber { get; set; }
19         1 reference
20         public long FacultyId { get; set; }
21         1 reference
22         public Faculty Faculty { get; set; }
23         1 reference
24         public List<Subject> Subjects { get; set; }
25     }
26 }
```


Konfiguracije modela

- Svaka konfiguracija implementira IEntityConfiguration interfejs.
- Kako bi se u potpunosti konfigurisala veza između 2 modela potrebno je da oba modela imaju adekvatne reference jedan na drugi.
- Primer: Student ide na jedan fakultet, jedan fakultet ima više studenata. Student sluša više predmeta, na predmetu je više studenata.

```
Faculty.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Models
7 {
8     5 references
9     public class Faculty
10     {
11         2 references
12         public long Id { get; set; }
13         0 references
14         public string Name { get; set; }
15         0 references
16         public string City { get; set; }
17         1 reference
18         public List<Student> Students { get; set; }
19     }
20 }

Student.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Models
7 {
8     10 references
9     public class Student
10     {
11         2 references
12         public long Id { get; set; }
13         2 references
14         public string Name { get; set; }
15         1 reference
16         public string Lastname { get; set; }
17         1 reference
18         public string IndexNumber { get; set; }
19         1 reference
20         public long FacultyId { get; set; }
21         1 reference
22         public Faculty Faculty { get; set; }
23         1 reference
24         public List<Subject> Subjects { get; set; }
25     }
26 }

Subject.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Models
7 {
8     5 references
9     public class Subject
10     {
11         2 references
12         public long Id { get; set; }
13         1 reference
14         public string Name { get; set; }
15         0 references
16         public int Semester { get; set; }
17         1 reference
18         public List<Student> Students { get; set; }
19     }
20 }
```


Konfiguracije modela

- Ako je u pitanju veza 1 na više zavisni entitet je onaj koji ima strani ključ. U primeru student-fakultet zavisni entitet je student pa prema tome mora imati FacultyId i Faculty objekat u sebi. Dok fakultet sa druge strane mora imati listu studenata.
- Objekti koje smo dodali nam služe kako bismo lakše dobavljali povezane podatke. Ako recimo želimo da učitamo odjednom studenta i njegov fakultet. Ovo se u SQL-u rešava spajanjem tabela (JOIN) ✨
- Kod veze 1 na 1 NEOPHODNO je eksplicitno reći koji entitet je zavisan jer Entity Framework to ne može sam zaključiti. U ovakvom slučaju oba entiteta mogu imati referentni objekat ali samo jedan entitet treba da ima strani ključ.
- Kod veze više na više nijedan od 2 entiteta nije zavisni tako da oba imaju samo listu referentnih objekata. Interno, Entity Framework će znati da treba da kreira treću poveznju tabelu.



DbContext



DbContext

DbContext je polazna klasa u EF koja se vezuje za jednu bazu podataka

Omogućava:

- Opis Entiteta - Sadrži skup entiteta u vidu kolekcije `DBSet<>` klasa
- Marijalizaciju upita: konvertuje LINQ-to-Entities upite u SQL upite i šalje ih u bazu podataka
- Vodi računa o izmenama nad entitetima koje su nastale nakon njihovog učitavanja iz baze podataka
- Kreiranje CRUD operacije u zavisnosti od stanja entiteta
- Rukovanje vezama među tabelama
- Konvertovanje raw tabelarnih podataka u entitete



DbSet<TEntity>

- Mapira se na tabele u bazi podataka
- Koristi se za CRUD operacije (Create, Read, Update, Delete)

```
namespace CRUD_Example.Data
{
    7 references
    public class CRUD_ExampleContext : DbContext
    {
        0 references
        public CRUD_ExampleContext (DbContextOptions<CRUD_ExampleContext> options)
            : base(options)
        {
        }

        6 references
        public DbSet<CRUD_Example.models.Books> Books { get; set; }
    }
}
```



Dodavanje connection String-a

Podatke o bazi dajemo u vidu connection stringa u appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CRUD_ExampleContext": "Server=(localdb)\\mssqllocaldb;Database=CRUD_ExampleDB;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Proširivanje Startup-a sa DbContext-om

```
2 references
public IConfiguration Configuration { get; }

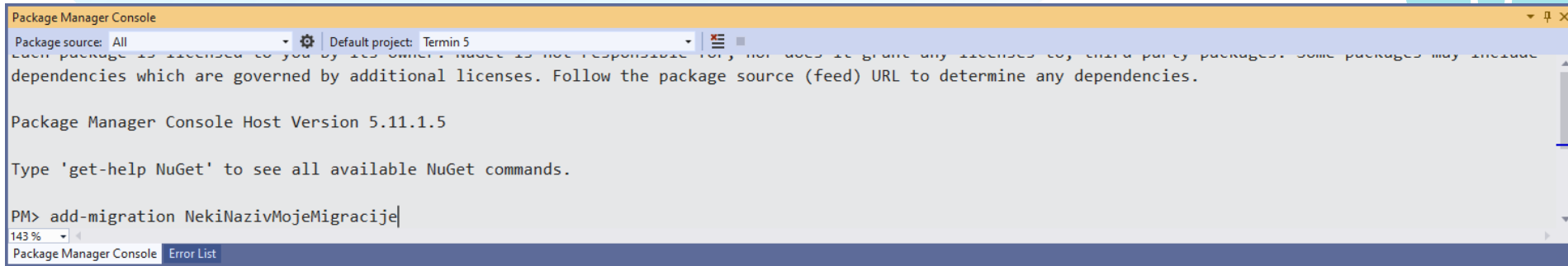
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    services.AddDbContext<CRUD_ExampleContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("CRUD_ExampleContext")));
}
```



Migracije

- Migracije predstavljaju C# kod koji će Entity Framework pretvoriti u SQL skriptu i izvršiti nad bazom podataka kako bi je kreirao/izmenio.
- Migracije moramo kreirati kada prvi put pravimo bazu i kad god nešto izmenimo u modelima ili konfiguraciji.
- Migracije generišemo kucajući: `add-migration NazivMigracije` u package manager konzolu



```
Package Manager Console
Package source: All Default project: Termin 5
Even package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

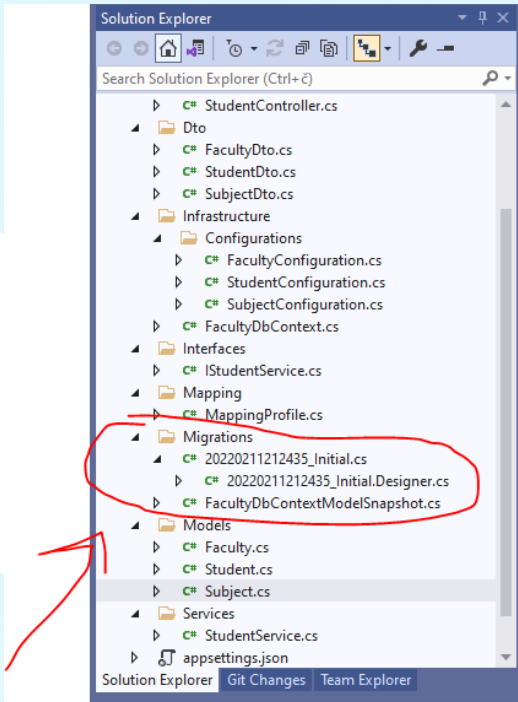
Package Manager Console Host Version 5.11.1.5

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

PM> add-migration NekiNazivMojeMigracije
143 %
Package Manager Console Error List
```


Migracije

- Nakon kreiranja prve migracije u projektu će nam se pojaviti folder Migrations sa migracijom i još nekim propratnim fajlovima.
- Migraciju možemo izvršiti nad bazom podataka kucajući : update-
database u package manager konzolu.



CRUD operacije

Pomoću objekta DbContext klase pristupamo tabelama u bazi

Učitavanje svih elemenata nekog tipa u listu:

```
await _context.Books.ToListAsync();
```

Elemente pretražujemo po identifikatoru pomoću Find(id) metode, gdje je id identifikator objekta koji se traži

```
var books = await _context.Books.FindAsync(id);
```

Nove elemente dodajemo putem Add(object) metode, parametar Add metode je objekat koji se dodaje

```
_context.Books.Add(books);
```



CRUD operacije

Elemente brišemo putem Remove(object) metode, parametar Remove metode je objekat koji se briše

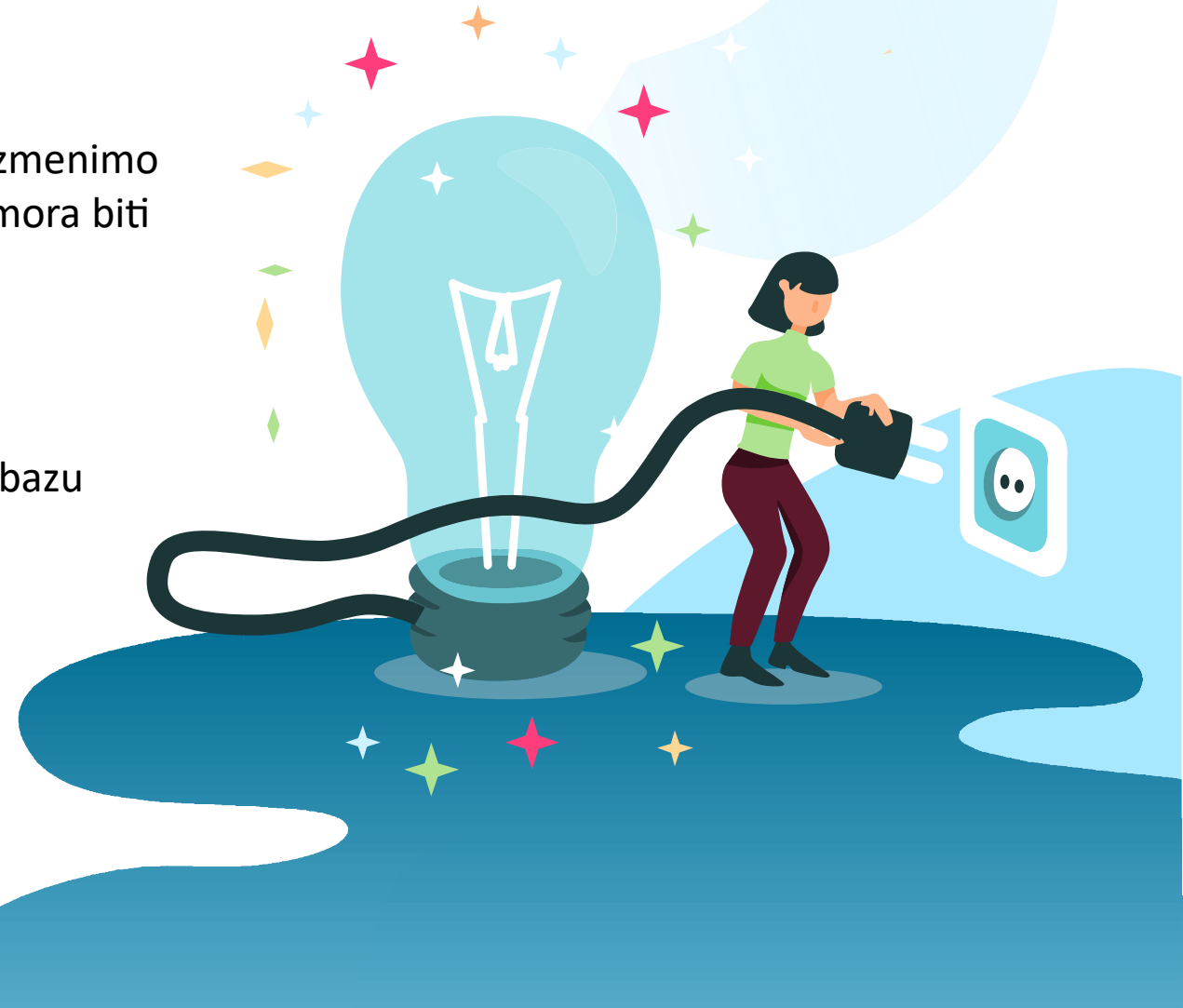
```
_context.Books.Remove(books);
```

Za izmenu objekta, moramo da obavestimo EF da želimo da izmenimo neki objekat. Uslov za izmenu je da identifikator tog objekta mora biti postojeći identifikator u bazi

```
_context.Entry(books).State = EntityState.Modified;
```

Kako bismo rezultat izvršavanja CRUD operacija primenili na bazu pozivamo SaveChangesAsync() metodu

```
await _context.SaveChangesAsync();
```



Code-First konvencije

Mapiranje klasa na tabele u bazi:

1. Za svaku domensku klasu koja se nalazi u DbSet<> kreiraće se nova tabela u bazi
2. Za klase koje su sadržane u klasama iz 1, biće kreirane tabele u bazi
3. Za podklase klasa iz 1, biće kreirane tabele u bazi



Data transfer objekat (Dto)



Dto

- Data transfer objekti ili Dto su objekti koje nose podatke između procesa ili između slojeva aplikacije.
- Glavna ideja je da se smanji količina podataka koji se šalju i odvoji model podataka koji se razmenjuje od onog koji se čuva npr. u bazi podataka.
- Klase Dto ne bi smele da imaju biznis logiku niti da vrše obradu podataka, sem eventualno serijalizacije ako je potrebno.
- Dto je naročito bitan u slučajevima kada različite vrste prikaza istog modela podataka na frontu aplikacije
- Primer: Kada se korisnik registruje na sistem upisuje ime prezime, password, email.. A kada se prijavljuje unosi samo email i password, jasno je da nam u ovom slučaju trebaju 2 Dto-a



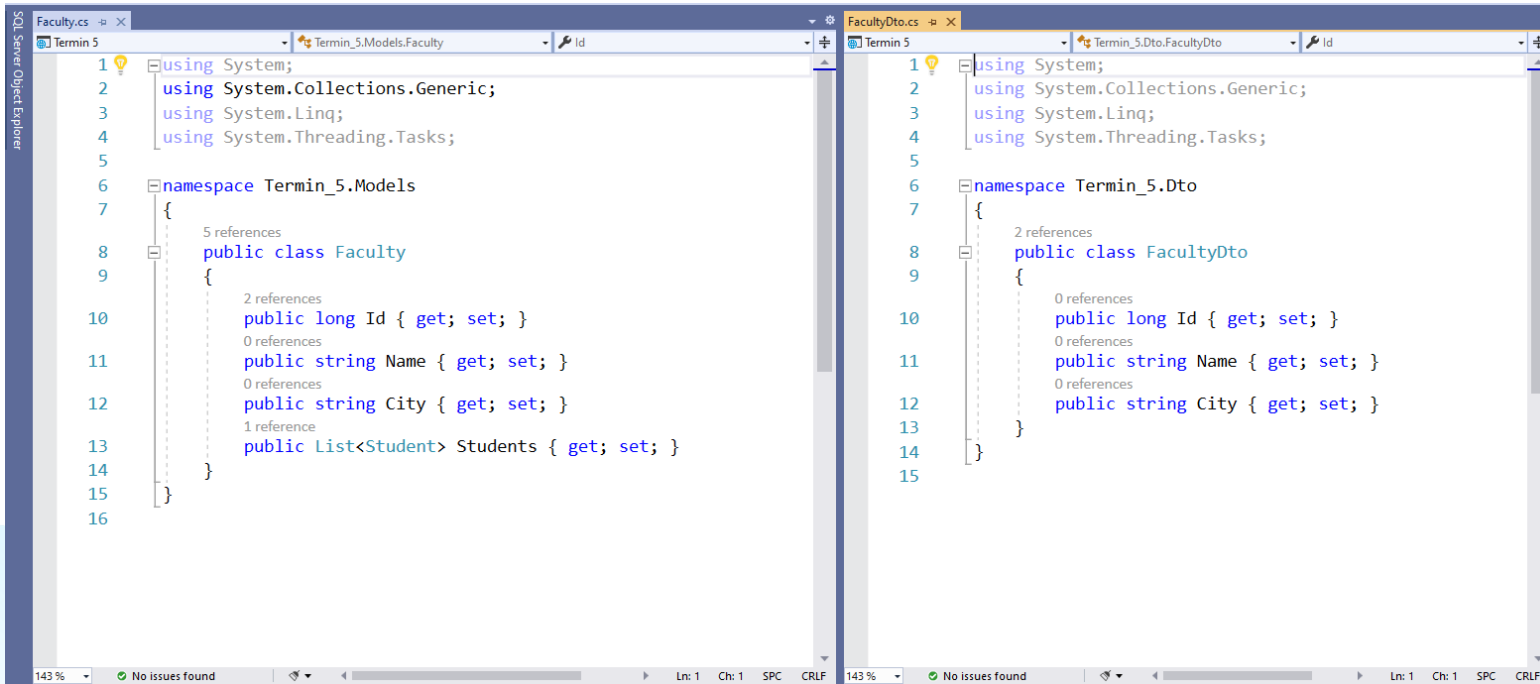
Dto u troslojnoj arhitekturi

- U troslojnoj arhitekturi Dto i model podataka koji se čuvaju u bazi su različite klase i jasno je definisano koji sloj sme da vidi koji model
- Kontrolerski sloj vidi samo Dto, prima sa fronta Dto i prosleđuje ga servisnom sloju, a po potrebi kao odgovor od servisnog sloja dobija isto samo Dto.
- Servisni sloj od kontrolera prima Dto, a od sloja podataka modele iz baze.
- Sloj podataka rukuje samo modelima iz baze.



AutoMapper

- Kao što vidimo servsini sloj poznaje oba modela podataka što znači da nekako mora „prepakovati“ modele iz baze u Dto kako bi ga vratio kontroleru i obrnuto.
- To možemo raditi „ručno“, a možemo i pomoću automatske biblioteke AutoMapper
- AutoMapper je konvencijski baziran mapper koji na osnovu naziva polja u klasama može da namapira jedan objekat na drugi.



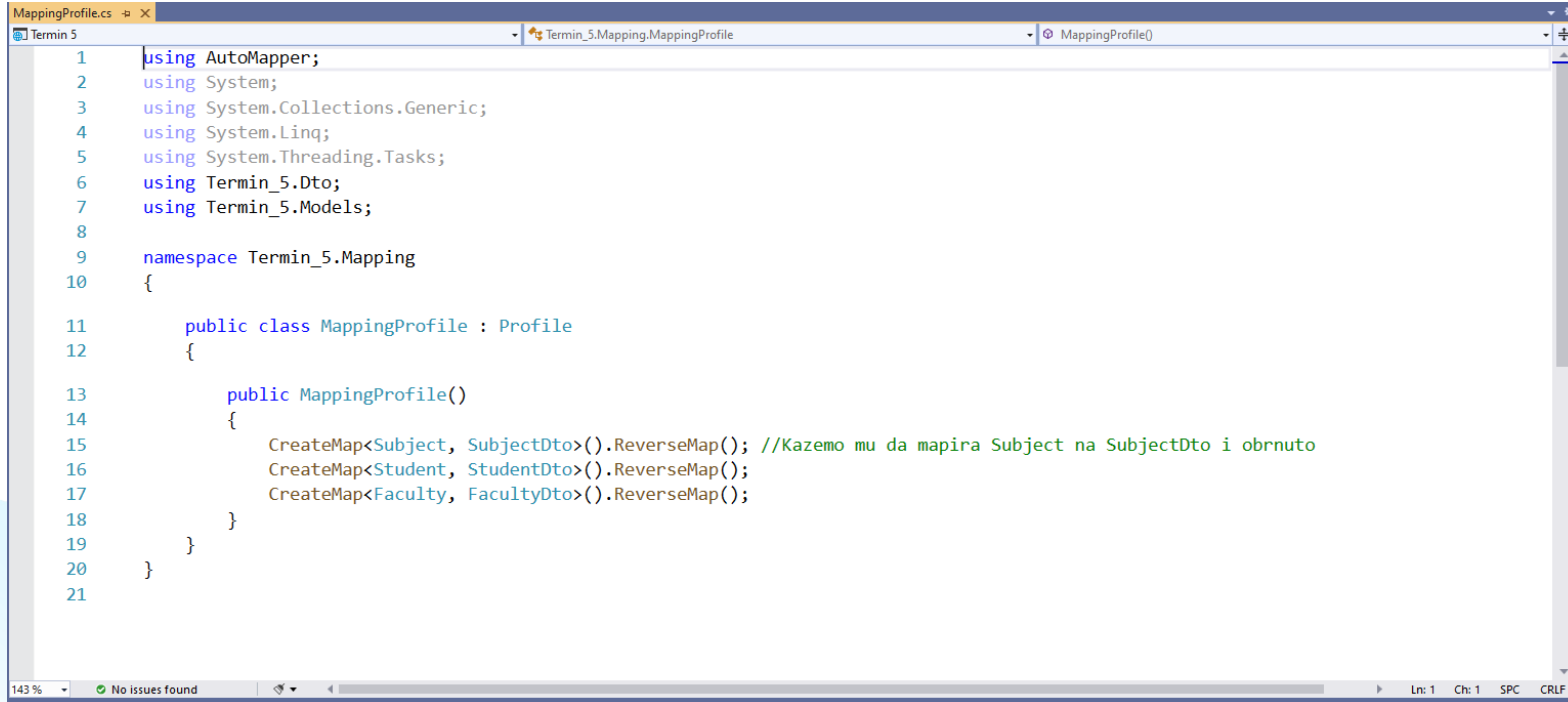
```
Faculty.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Models
7 {
8     public class Faculty
9     {
10         public long Id { get; set; }
11         public string Name { get; set; }
12         public string City { get; set; }
13         public List<Student> Students { get; set; }
14     }
15 }
16

FacultyDto.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Termin_5.Dto
7 {
8     public class FacultyDto
9     {
10         public long Id { get; set; }
11         public string Name { get; set; }
12         public string City { get; set; }
13     }
14 }
15
```

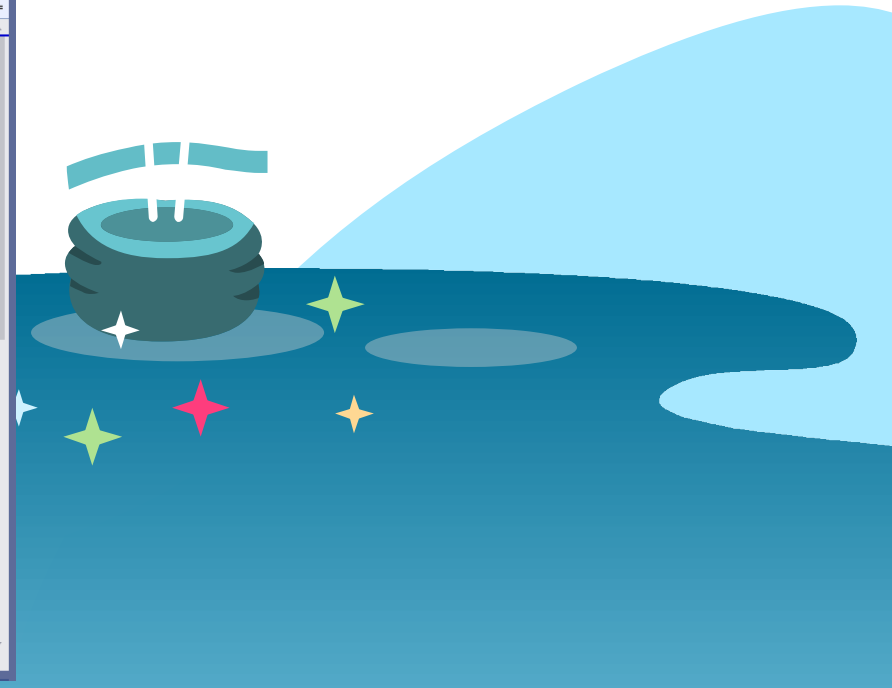


AutoMapper

- Kako bi AutoMapper znao da pretvori jedan tip objekta u drugi nazivi polja moraju biti isti kao i tipovi polja.
- Konvencije o nazivanju se mogu pronaći u dokumentaciji <https://docs.automapper.org/en/stable/index.html>
- Pored adekvatnog nazivanja polja potrebno je i eksplicitno reći AutoMapperu koju klasu da mapira na koju. Za to koristimo Mapping profil



```
1  using AutoMapper;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Termin_5.Dto;
7  using Termin_5.Models;
8
9  namespace Termin_5.Mapping
10 {
11     public class MappingProfile : Profile
12     {
13         public MappingProfile()
14         {
15             CreateMap<Subject, SubjectDto>().ReverseMap(); //Kazemo mu da mapira Subject na SubjectDto i obrnuto
16             CreateMap<Student, StudentDto>().ReverseMap();
17             CreateMap<Faculty, FacultyDto>().ReverseMap();
18         }
19     }
20 }
21
```



AutoMapper

- Da bismo koristili AutoMapper u kodu potrebno je da ga dodamo u kontejner sa zavisnostima. Dodajemo ga kao singleton jer nema stanje i ne želimo da troši previše memorije.

```
var mapperConfig = new MapperConfiguration(mc =>
{
    mc.AddProfile(new MappingProfile());
});
```

```
IMapper mapper = mapperConfig.CreateMapper();
services.AddSingleton(mapper);
```

- Mapiranje vršimo pozivanjem Map metode i navođenjem tipa objekta u koji mapiramo. U ovom slučaju prebacili smo StudentDto u Student.

```
public StudentDto AddStudent(StudentDto newStud)
{
    Student stud = _mapper.Map<Student>(newStud);
    _dbContext.Students.Add(stud);
    _dbContext.SaveChanges();

    return _mapper.Map<StudentDto>(newStud); //Dobra je praksa vratiti kreirani objekat nazad,
                                              //narocito ako se auto generise ID
}
```



Termin 5 zadatak

- Dodati model profesora sa poljima: ime, prezime, zvanje
- Profesor drži više predmeta, a predmet drži jedan profesor
- Konfigurisati njihove veze, generisati migracije i promeniti bazu podataka.
- Napraviti Dto i podesiti mapiranje.
- Napraviti servis profesora sa metodama GetAll i Create
- Napraviti kontroler profesora i testirati



THANK YOU

