.NET Okruženje

Vježba 7: Entity framework

Sadržaj

Entity framework (EF)

Pri radu s bazom podataka iz C# postoji nekoliko raznih načina kako možemo iz baze dohvatiti ili spremiti podatke:

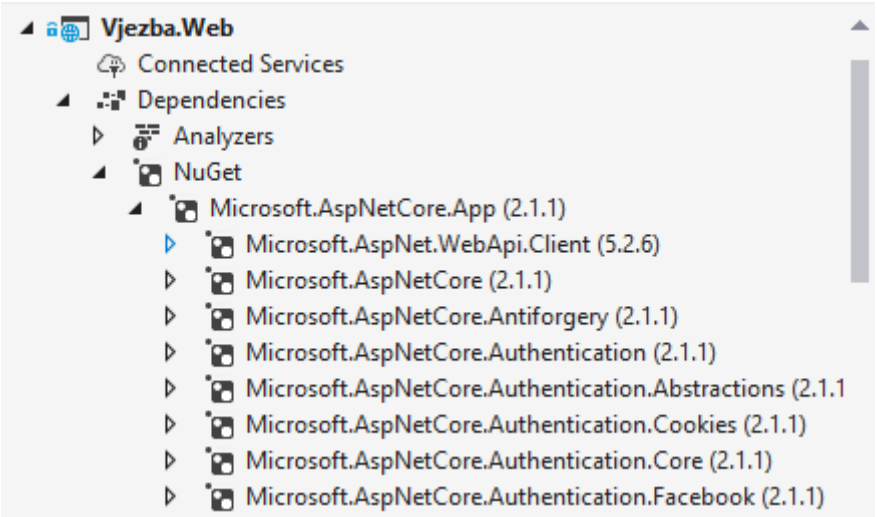
- Korištenjem ADO.NET adaptera – set komponenti kojima se može iz C# dohvaćati podatke iz baze podataka i spremati ih u odgovarajuće prilagođene objekte (DataSet, DataTable, ...)
- Korištenjem ORM alata kao što je NHibernate, Dapper ili **EF**
  - ORM = Object Relational Mapping
  - Prikazivanje podataka iz baze korištenjem klasa u C#
  - Praćenje promjena
  - ...

Većina modernih aplikacija pisanih u .NET tehnologiji koristi Entity Framework kao alat za komunikaciju s bazom podataka.

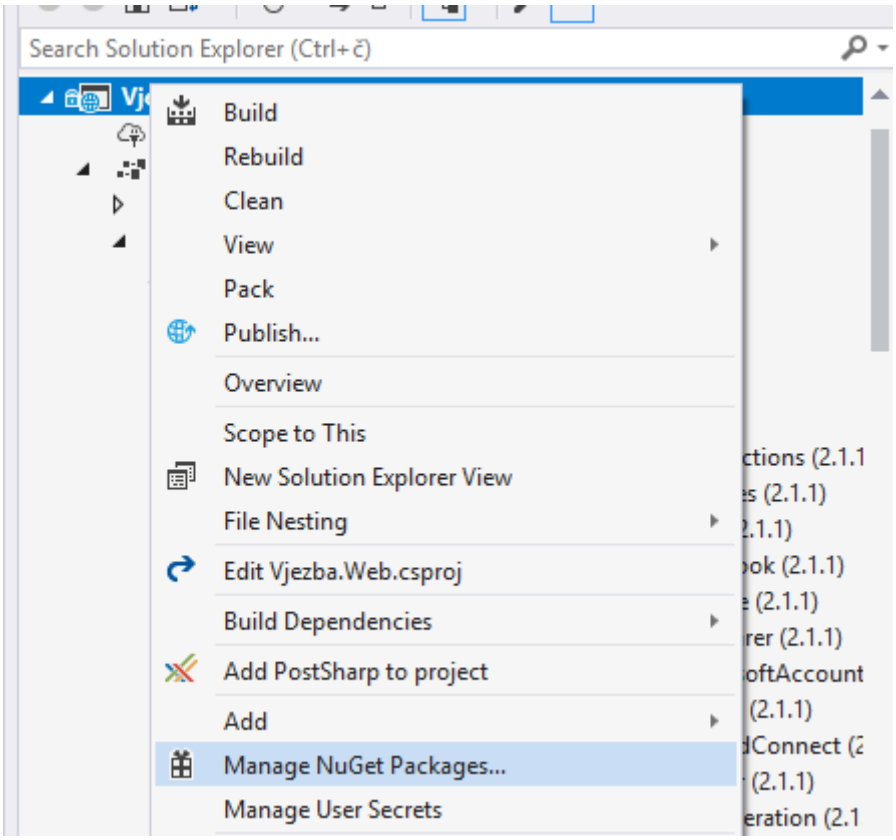
Biblioteke i zavisne komponente

Da bi koristili entity framework u aplikaciji, potrebno je upoznati se s načinom kako se referenciraju biblioteke i zavisni projekti u .NET Core MVC Web aplikaciji. Potrabn je alat za jednostavnu manipulaciju pomoćnim bibliotekama (pluginovima), te izvođenje raznih aktivnosti vezanih uz pomoćne biblioteke, kao što je generiranje koda, izvršavanje skripti nad bazom podataka i sl. Alat koji tako nešto omogućava je Nuget package manager, i može se koristiti kroz konzolu (*package manager console*) ili UI sučelje (*Manage nuget packages for solution*)

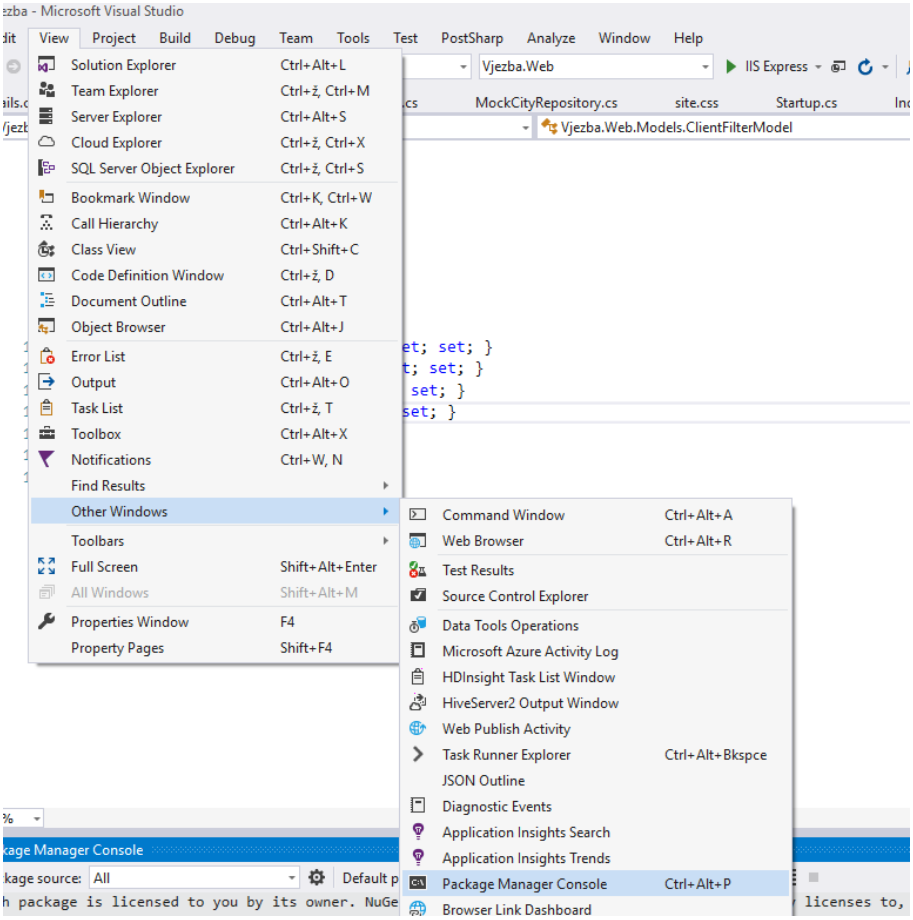
Popis trenutno instaliranih biblioteka može se vidjeti u *solution explorer* prozoru, pod *Dependencies*:



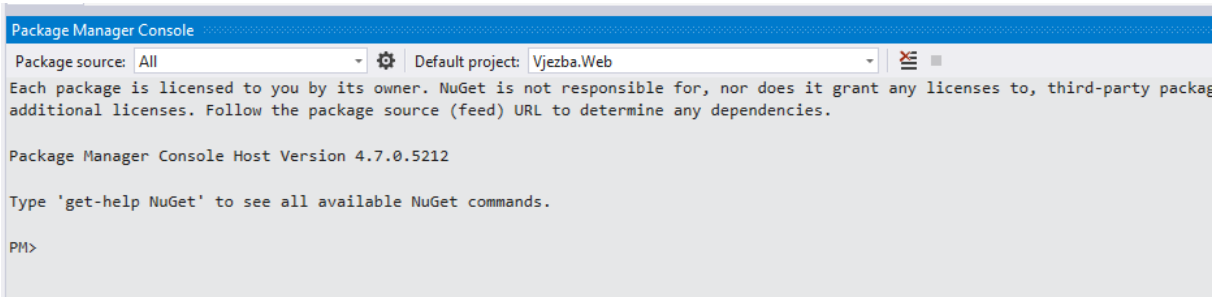
Također, moguće je pogledati instalirane biblioteke i pakete kroz Nuget UI sučelje, desnim klikom na solution i odabirom *Manage NuGet Packages*:



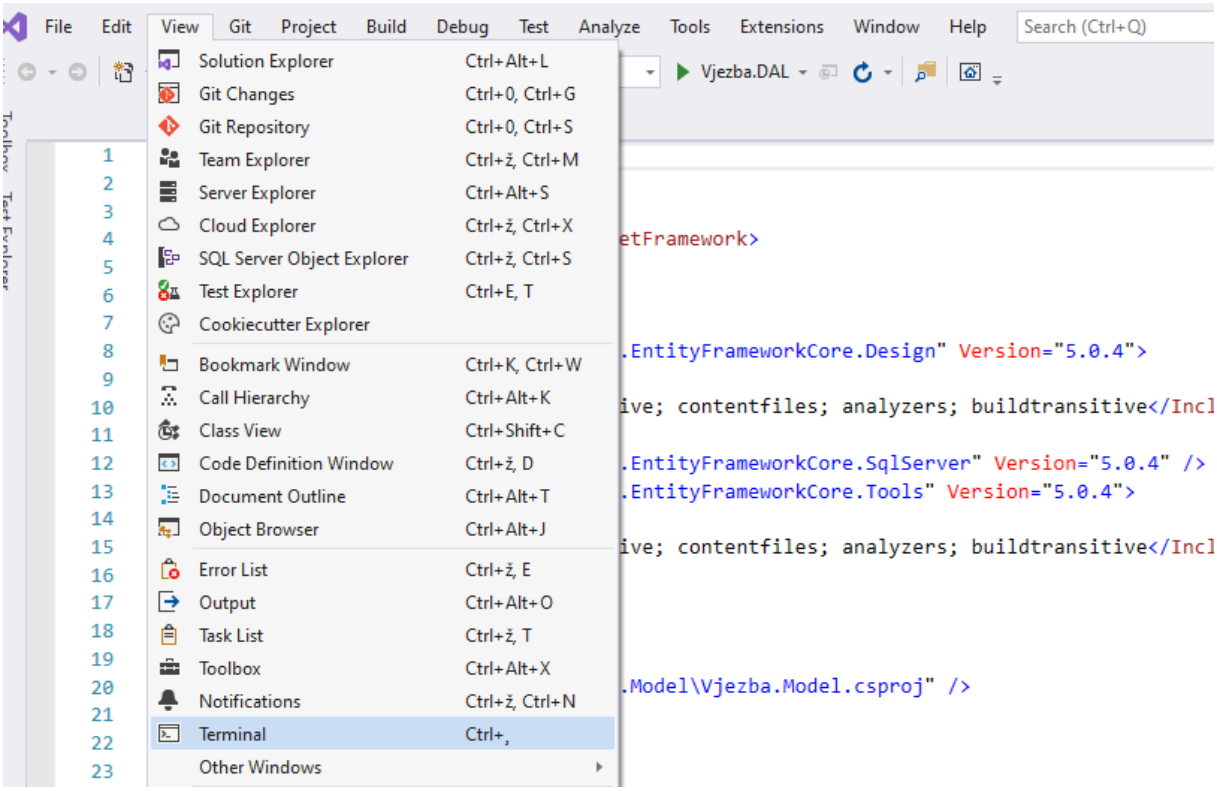
Također će nam biti potrebna konzola za izvršavanje skripti vezanih uz Entity Framework (EF):

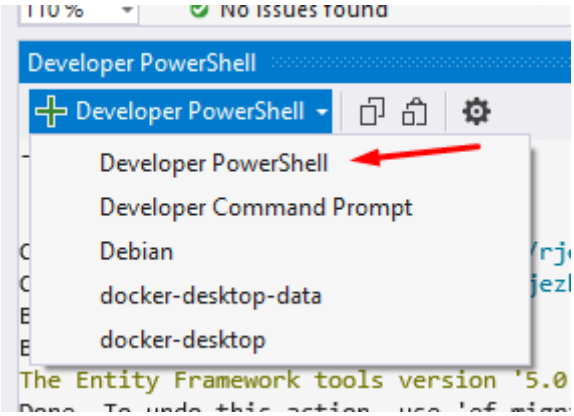


Svaka naredba koja se izvodi se izvodi nad nekim projektom, što se može vidjeti u konzoli:



Za rad s migracijama, koristit će se Developer PowerShell konzola:





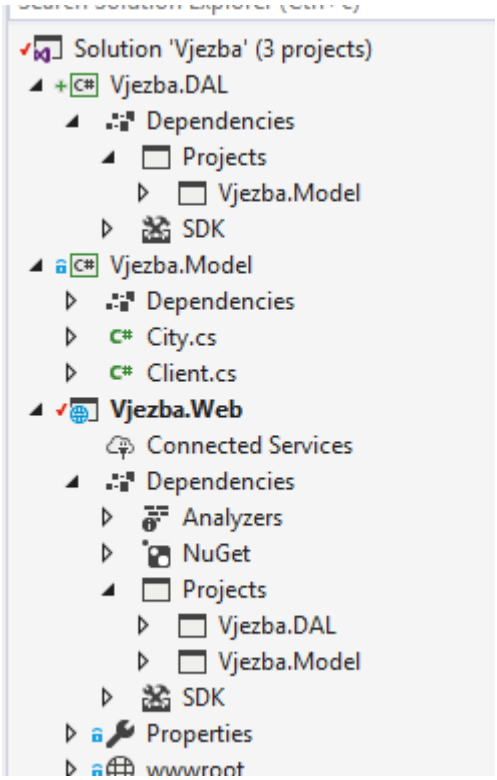
Prije nego se krene raditi sa Entity Frameworkom, dobro je resturkturirati projekt na smislene slojeve:

- **Model** sloj – sadržavat će klase koje predstavljaju tablice u bazi podataka – primjerice Client, Meeting i sl.
- **DAL** sloj – sadržavat će migracijske skripte i klase vezane za EF
- **Web** sloj – controlleri, view-ovi, statičke HTML stranice i ostale komponente potrebne za prikaz korisničkog sučelja

Zadatak 7.1

Premjestiti klase **Client** i **City** u **Vjezba.Model** projekt i maknuti Mock repozitorije.

- Kod premještanja klase paziti na promjenu namespace-a
  - Dodati referencu na Model projekt desnim klikom na Web projekt i odabirom Add reference
  - Napraviti ispravne promjene u view komponentama
    - Vjerojatno je potrebno izmjeniti i _ViewImports datoteku
  - Osigurati da se kod prevodi
- Izbrisati ~~MockClientRepository~~ i ~~MockCityRepository~~
  - Osigurati da se kod prevodi, mogu se zakomentirati dijelovi koda u controller klasi
- Kreirati **Vjezba.DAL** projekt
  - Kreirati kao Class Library tip
  - Referencirati Model projekt iz DAL projekta
  - Referencirati DAL projekt iz Web projekta
  - Instalirati paket: `install-package Microsoft.EntityFrameworkCore.SqlServer`
  - Instalirati paket: `install-package Microsoft.EntityFrameworkCore.Design`
  - Instalirati paket: `install-package Microsoft.EntityFrameworkCore.Tools`
  - Osigurati da je prilikom instalacije paketa naznačen ispravni projekt u konzoli (Vjezba.DAL)



Konačno rješenje zadatka treba izgledati kao na slici desno.

Konfiguracija EF

Kroz naredno poglavlje bit će navedeni koraci koje je potrebno napraviti kako bi se EF mogao koristiti u projektu.

Prije početka osigurati da je moguće spojiti se na lokalni MSSQL server (najčešće .\sqlexpress). Provjeriti u SQL Server Management Studio alatu je li moguće, te dodati bazu podataka naziva ClientManager.

DbContext klasa

Prvi korak je u DAL projekt dodati klasu koja predstavlja kontekst – bazu podataka. Klasa mora **naslijediti iz DbContext** klase. U kontekstu aplikacije za evidenciju klijenata to bi bila **ClientManagerDbContext** klasa (source kod 1).

DAL/ClientManagerDbContext.cs

Source kod 1

```
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Vjezba.DAL
{
    public class ClientManagerDbContext : DbContext
    {
        protected ClientManagerDbContext() { }

        public ClientManagerDbContext(DbContextOptions<ClientManagerDbContext> o
ptions) : base(options)
        { }
    }
}
```

Zadatak 7.2.A

Dodati DbContext klasu prema gornjem primjeru. Osigurati da se kod prevodi.

Zadatak 7.2.B

U **appsettings.json** u Web projektu dodati ispravne podatke za spajanje na bazu podataka. Kod može izgledati po prilici ovako, ali moguće je i da se ponešto razlikuje:

Web projekt/appsettings.json

Source kod 2

```
{
  "Logging": ...
  "AllowedHosts": "*",

  "ConnectionStrings": {
    "ClientManagerDbContext": "Server=.\\sqlexpress;Database=ClientManager;Trus
ted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Zadatak 7.2.C

Podesiti model na sljedeći način:

- Svaka klasa iz modela (koja predstavlja redak odgovarajuće tablice u bazi podataka) treba imati polje **Id** anotirano s **[Key]** atributom (source kod 3). Dodati ispravni using (boldano u source kod-u)
- Svaka kolekcija u modelu koja predstavlja **1-N** vezu treba biti definirana kao virtualna, tipa **ICollection<T>** (source kod 4)
- Svaki **strani ključ** (kod 1-N veze, reference iz kolekcije) treba imati definirano **Id** polje referenciranog objekta i anotaciju **[ForeignKey]** (source kod 5)
- Veze **N-N** se definiraju na način da se u **obje klase** koje su povezane doda **kolekcija** kako je opisano u točki 2 (source kod 6)

Donji primjeri koda su napisani za sličnu aplikaciju za evidenciju kvizova, te treba preuzeti sličan način anotacija i pisanja koda i primjeniti na aplikaciju za klijente.

Models/Quiz.cs

Source kod 3

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace Vjezba.Model
{
    public class Quiz
    {
        [Key]
        public int Id { get; set; }

        ...
    }
```

Models/Quiz.cs

Source kod 4

```
namespace Vjezba.Model
{
    public class Quiz
    {
        [Key]
        public int Id { get; set; }

        ...

        public virtual ICollection<Question> Questions { get; set; }
    }
}
```

Models/Question.cs

Source kod 5

```
namespace QuizManagerWeb.Models
{
    public class Question
    {
        [Key]
        public int Id { get; set; }

        ...

        [ForeignKey("Quiz")]
        public int QuizId { get; set; }

        public virtual Quiz Quiz { get; set; }
    }
}
```

Models/User.cs

Source kod 6

```
namespace QuizManagerWeb.Models
{
    public class User
    {
        [Key]
        public int Id { get; set; }

        public string UserName { get; set; }
        public string Password { get; set; }

        public virtual ICollection<Role> Roles { get; set; }
    }
}
```

Models/Role.cs

```
namespace QuizManagerWeb.Models
{
    public class Role
    {
        [Key]
        public int Id { get; set; }

        public string Name { get; set; }

        public virtual ICollection<User> Users { get; set; }
    }
}
```

Zadatak 7.2.D

U klasu **QuizManagerDbContext** potrebno je dodati informaciju o tome koje tablice tamo pripadaju, na način da se definira svojstvo tipa **DbSet<T>**, gdje je T tip klase koja predstavlja redak u nekoj tablici (source kod 7). Dodati **DbSet<T>** za svaku klasu/entitet koji želimo da bude tablica u bazi podataka.

/QuizManagerDbContext.cs

Source kod 7

```
using QuizManager.Model;
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace QuizManager.DAL
{
    public class QuizManagerDbContext : DbContext
    {
        public DbSet<Quiz> Quizzes { get; set; }
    }
}
```

Zadatak 7.2.E

Registrirati SQL Server i EF u Startup.cs klasi.

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        ...
    });

    services.AddDbContext<ClientManagerDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("ClientManagerDbContext") ,
                   opt => opt.MigrationsAssembly("Vjezba.DAL")));

    services.AddMvc().....
}
```

Migracijske skripte

Kako bi EF mogao ispravno funkcionirati, shema baze podataka mora biti u skladu s klasama definiranim u projektu. Održavanje sheme baza i modela klasa u sinkronicitetu nije uvijek jednostavan zadatak zbog kontinuiranih promjena modela, pogotovo ako se radi ručno, bez dodatne pomoći specijaliziranih alata. Srećom, EF sadrži alat za usklađivanje sheme baze i modela: *EF migrations*.

Općenito, pri svakoj promjeni modela (može biti više promjena odjednom), potrebno je provesti sljedeći proces kako bi se EF i baza uskladili:

1. Napraviti željene promjene u modelu (primjerice, dodati novo polje, novu klasu, itd.)
2. Dodati migraciju proizvoljnog opisnog imena naredbom „**dotnet ef migrations add NAZIV**“
3. Osvježiti bazu pozivom naredbe „**dotnet ef database update NAZIV**“
    a. Ukoliko se radi o bazi podataka koja je u produkcijskom okruženju, moguće je i pozvati naredbu „**dotnet ef migrations script FROM TO“** koja odgovarajuću skriptu generira u .sql datoteku, te se može naknadno izvesti ručno na željenoj instanci baze podataka putem MSSQL management studio alata ili ServerExplorer prozora.

Dodatno, treba paziti da se u višeslojnoj aplikaciji ispravno definiraju parametri koji određuju gdje se model nalazi, gdje se nalazi DB context klasa te gdje se nalaze podaci za spajanje na bazu (*connection string*). Za to su zaduženi sljedeći parametri:

- **--startup-project** XXXX – definira koji projekt sadrži ispravne *connection string*
- **--context** XXXX – koja klasa se uzima kao DB context
- Projekt u kojem želimo migracije mora biti „root“ projekt naredbe u developer konzoli.

Isti parametri koiste se kod svih gore navedenih naredbi. U praksi, kreiranje inicijalne migracije može izgledati ovako:

Developer PowerShell

```
C:\git-tt\net-ok-20-21\Vjezba.DAL> dotnet ef migrations add Initial --startup-project
../Vjezba.Web --context ClientManagerDbContext

Build started...

Build succeeded.

The Entity Framework tools version '5.0.2' is older than that of the runtime '5.0.4'.
Update the tools for the latest features and bug fixes.

Done. To undo this action, use 'ef migrations remove'
```

Osvježavanje baze podataka bi izgledalo ovako:

Developer PowerShell

```
C:\git-tt\net-ok-20-21\Vjezba.DAL> dotnet ef database update --startup-project
../Vjezba.Web --context ClientManagerDbContext
```

U „real-world“ projektima, česta je praksa definirati migrations-readme datoteku u kojoj se nalaze gornje naredbe s već definiranim svim potrebnim parametrima, te se novi članovi tima mogu lako snaći.

Zadatak 7.3

Dodati prvu migraciju i osvježiti bazu prema gornjim uputstvima. Migracije trebaju ići u DAL projekt.

Zadatak 7.4

Dodati u model novi entitet koji predstavlja sastanke sa klijentom.

- Novi entitet je potrebno nazvati **Meeting**
  - Sastanak može biti tipa InPerson, VideoCall
  - Potrebno je navesti datum početka i kraja (mogu biti null)
  - Definirati status: Scheduled, Cancelled
  - Treba moći definirati lokaciju (opcionalno) te komentare (opcionalno)
- Za nekog klijenta možemo imati nekoliko sastanaka
- Dodati automatski generiranu migracijsku skriptu, te generirati SQL skriptu koja izvršava promjene kako bi model u bazi odgovarao modelu definiranom u klasama
- U DbContext klasu dodati odgovarajuće DbSet polje

Inicijalni podaci

Pri razvoju projekata česta je situacija (pogotovo na početku) da su nam potrebni nekakvi podaci s kojima možemo testirati ostale funkcionalnosti aplikacije. Također, možemo napuniti i neke potrebne podatke kao što su razni tipovi, lookup vrijednosti i slično koji ostaju i u produkcijskom okruženju. U EF core alatu, inicijalni podaci se pune na način da se unutar modela proširi funkcionalnost metode OnModelCreating unutar koje se definiraju podaci za pojedine entitete.

ClientManagerDbContext

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

    modelBuilder.Entity<City>().HasData(new City { ID = 1, Name = "Zagreb" });
    ...
}
```

Nakon toga, potrebno je izgenerirati novu migraciju i pokrenuti kako bi se podaci uspješno osvježili.

Korisni linkovi

- https://docs.microsoft.com/en-us/ef/core/modeling/data-seeding

Zadatak 7.5

Dodati podatke za 3 grada po želji i jednog klijenta po želji.

- Osigurati da se repetitivnim dodavanjem migracija ili pozivanjem **dotnet ef database update** podaci ne dupliraju (dodati testne 2 migracije)

- Pozvati **dotnet ef database update** više puta kako bi se uvjerili da se podaci ne dupliciraju

CRUD operacije

C: Create – stvaranje podatka u bazi
R: Read – čitanje podataka iz baze
U: Update – promjena postojećih podataka u bazi
D: Delete – brisanje odgovarajućeg podatka iz baze

U ovom poglavlju ćemo pogledati kako iskoristiti EF za svaku od pojedinih operacija nad bazom podataka.

Rukovanje kontekstom

Kako bi mogli koristiti EF, potrebna nam je instanca DB context klase, i upravo taj objekt prati promjene koje radimo nad entitetima i sprema ih u bazu podataka (kad dodajemo nove, mijenjamo postojeće i sl.). Jedan od izazova je definirati trajanje i životni ciklus upravo tog objekta.

Postoje dva načina kako se može kreirati instancu DB context klase: ručno, pozivom „new“ ili pustiti MVC radni okvir da ju kreira.

Kreiranje DB context klase ručno

```
var context = new QuizManagerDbContext();

//CRUD operacije nad kontekstom

context.Dispose();
```

Automatsko kreiranje instance

```
public class QuizController : Controller
{
    private QuizManagerDbContext _dbContext;

    public QuizController(QuizManagerDbContext dbContext)
    {
        this._dbContext = dbContext;
    }
```

Nakon toga koristimo **this._dbContext** kad god nam treba.

Preporučeni način korištenja je automatskim kreiranjem, i u ovom trenutku neće se dublje ulaziti u sami mehanizam po kojem to funkcionira.[1]

Create – stvaranje podataka

Kako bi kreirali podatak u bazi, koristimo sljedeći set naredbi (pod uvjetom da postoji ranije kreirani obični objekt tipa Quiz):

[1] Navedeni mehanizam je Dependency Injection, te ga je moguće samostalno istražiti, a detaljnije će biti objašnjen u sklopu kolegija Napredni web servisi .NET na specijalističkom studiju.

```
//Dodavanje novog kviza (pretpostavimo da postoji varijabla Quiz quiz
this._context.Quizes.Add(quiz);

//Spremanje promjena (commit)
this._context.SaveChanges();
```

Zadatak 7.6

Modificirati **ClientController** klasu da u akciji **Create** spremi klijenta u stvarnu bazu podataka, umjesto da koristi MockRepository.

- Postaviti da se dbContext objekt automatski napuni (pogledati primjer gore)
- Dodati **Client** objekt
    - Hard-codirati **CityID** na vrijednosti 1, 2 ili 3 (ne koristiti vrijednost iz padajućeg izbornika)
- Ne zaboraviti pozvati **SaveChanges**
- Uvjeriti se da nakon pozivanja **/Client/Create** i uspješnog dodavanja klijenta, je uistinu u bazi zapisana nova vrijednost (*ServerExplorer* prozor)
    - **Napomena**: pošto još nismo Index „spojili“ na pravu bazu, tamo se i dalje prikazuju klijenti samo iz mock repozitorija

Read – dohvaćanje i manipulacija podacima

Za dohvaćanje podataka koriste se LINQ upiti koji se pri upitu prevode u SQL naredbe, izvršavaju na bazi i kao rezultat vraćaju set objekata koji želimo dohvatiti. U načelu, naredbe za čitanje dijelimo na:

- naredbe za dohvat jednog podatka (po primarnom ključu)
- naredbe za dohvat više podataka (pretraga po kriteriju)

Za dohvaćanje jednog podatka možemo koristiti jedan od dva odsječka:

```
//U ovom slučaju ostavljamo mogućonst da quiz s željenim id-jem ne postoji
//Varijabla id sadrži vrijednost prema kojoj želimo dohvatiti kviz
Quiz result = this._context.Quizes
    .Where(p => p.Id == id)
    .FirstOrDefault();
```

```
//Dohvaćanje putem poziva funkcije 'find'
Quiz result = this._context.Quizes
    .Find(id);
```

Za dohvaćanje više podataka koristimo željene LINQ naredbe koje su obrađene u vježbi 3. Primjer dohvaćanja svih kvizova koji su nastali u 2013. godini:

```
//Dohvaćanje svih u 2013.
List<Quiz> result = this._context.Quizes
    .Where(p => p.DateCreated.Year == 2013)
    .ToList();
```

Tek pozivom **ToList()** naredbe se upit zaista prevodi u SQL i izvršava na bazi podataka.

Često postoji potreba da se iz baze automatski dohvate i relacije pojedine tablice. Primjerice, ukoliko kviz ima kategoriju (1-N veza), ovakav kod bi bacio iznimku, jer relacijsko svojstvo Category zahtjeva ili da se naknadno dohvati iz baze, ili da se učita automatski u samom upitu:

```
public IActionResult Details(int? id = null)
{
    var quiz = this._dbContext.Quizes
        .Where(p => p.ID == id)
        .FirstOrDefault();

    var categoryName = quiz.Category.Name;
    return View(null);
}
```

Kako bi to popravili, potrebno je već pri učitavanju kvizova dohvatiti i kategoriju (napuniti relacijsko svojstvo).:

```
//Dohvaćanje svih u 2013.
List<Quiz> result = this._context.Quizes
    .Include(p => p.Category)
    .Where(p => p.DateCreated.Year == 2013)
    .ToList();

var categoryName = result.First().Category.Name; // OK
```

Napomena: funkcija Include se nalazi u imenskom prostoru **Microsoft.EntityFremeworkCore**. Također, ne treba pretjerati sa uključivanjem puno relacija jer time upit postaje kompleksniji i performanse se mogu drastično smanjiti.

Zadatak 7.7

Modificirati akciju **Index, AdvancedSearch** i akciju **Details** u ClientController-u na način da se koristi ispravni poziv preko EntityFrameworka, umjesto dohvat iz MockRepository klase.

- Osigurati da se pozivom funkcije Include(…) dohvati i grad

Update i delete – izmjena i brisanje postojećeg podatka

U ovoj fazi neće se raditi update/delete primjeri, ali slijede odsječci koda pomoću kojih se navedene operacije mogu napraviti:

```
//Dohvat željenog podatka
Quiz result = this._context.Quizes
    .Find(id);

//Primjer izmjene
result.Title = "Izmjenjeni naslov";

//Spremanje promjena (commit)
This._context.SaveChanges();
```

```
//Dohvat željenog podatka
Quiz result = this._context.Quizes
    .Find(id);

//Postavljanje stanja na deleted - označavanje da je kviz obrisan
this._context.Entry(result).State = System.Data.Entity.EntityState.Deleted;

//Spremanje promjena (commit)
this._context.SaveChanges();
```