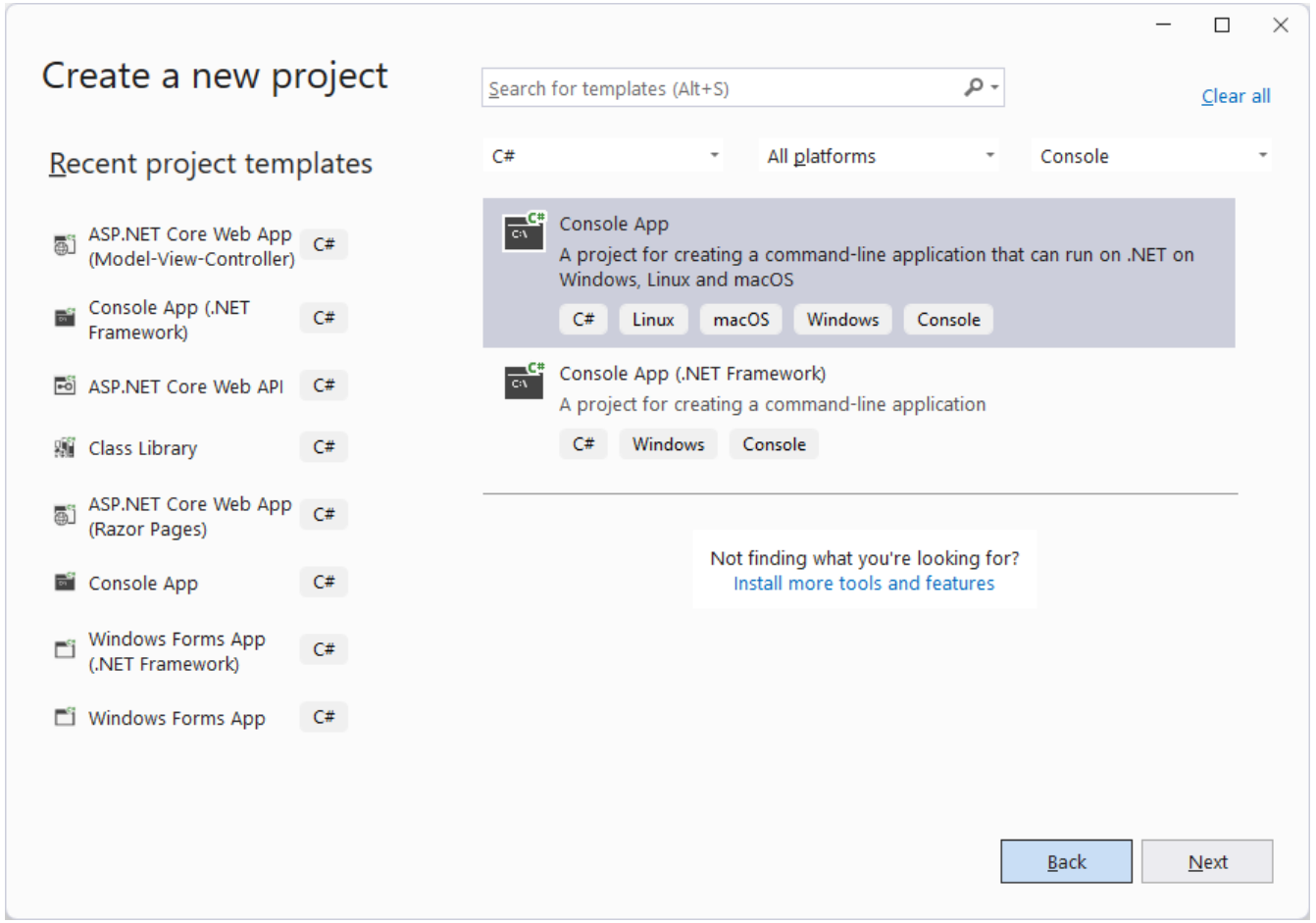
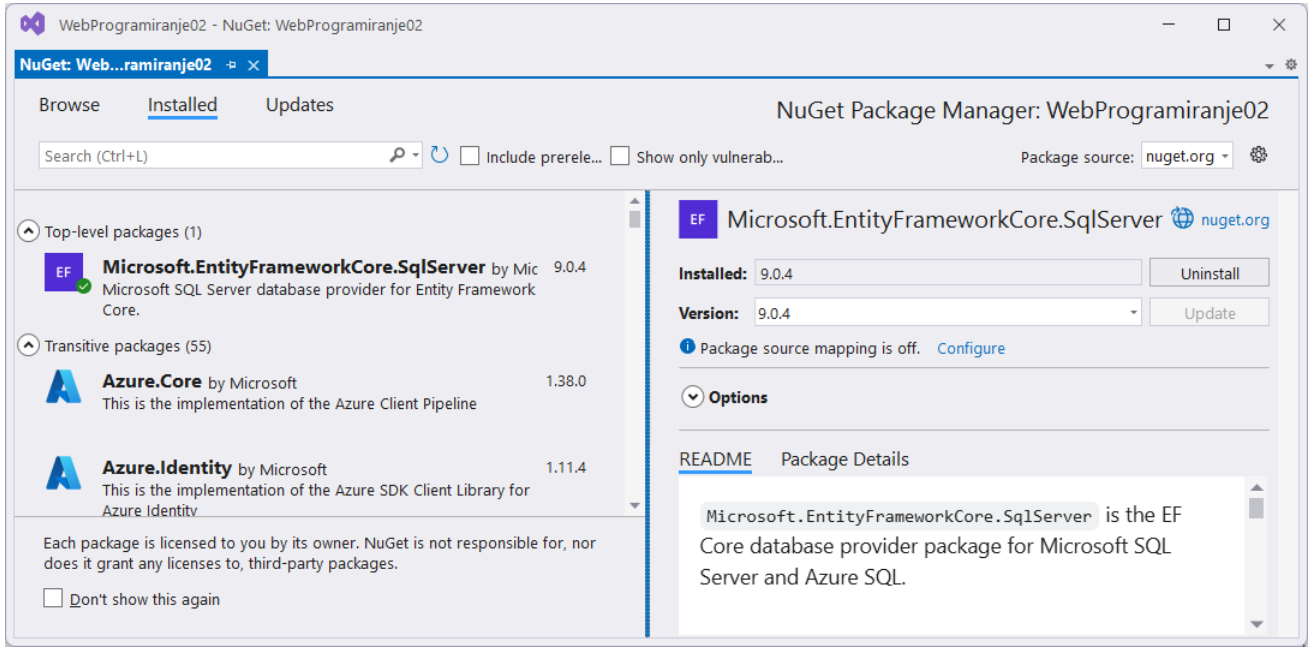
Entity Framework Core

.NET Core konzolna aplikacija



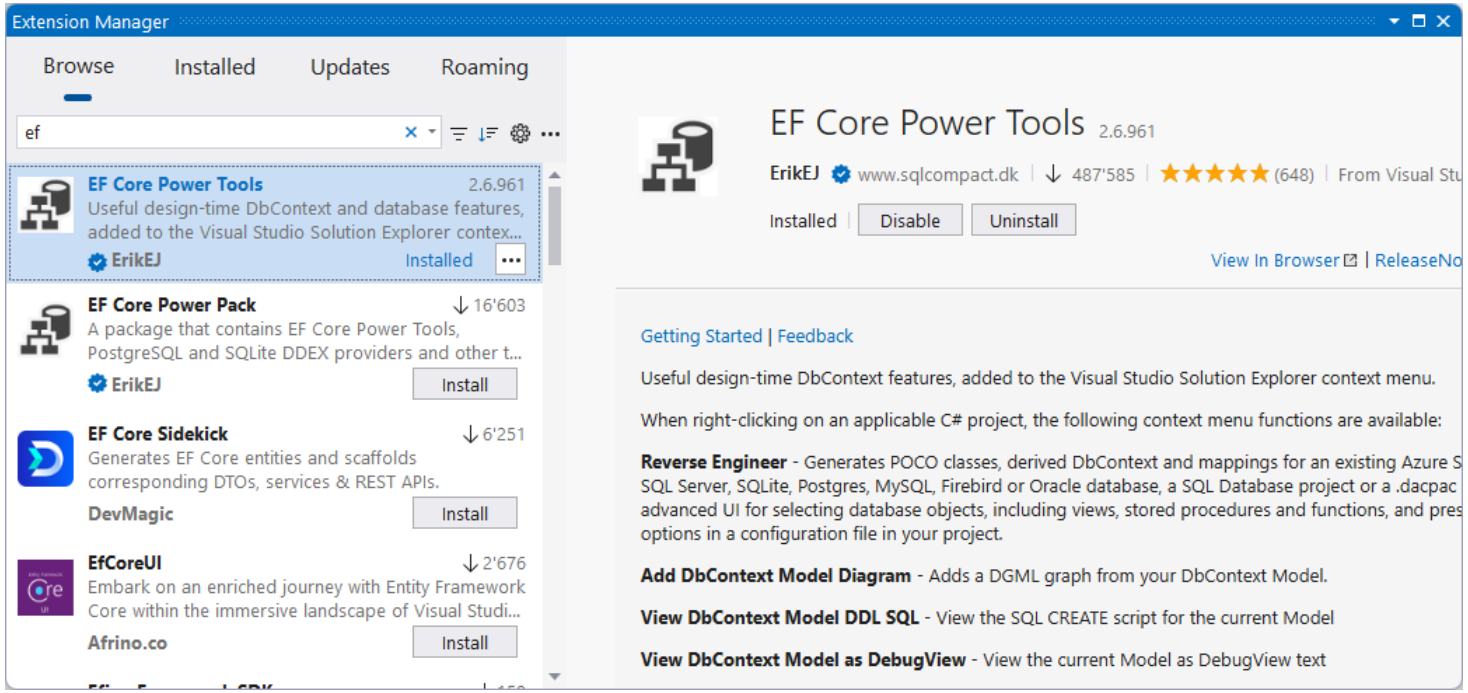
Microsoft.EntityFrameworkCore.SqlServer

Project->Manage Nuget Packages

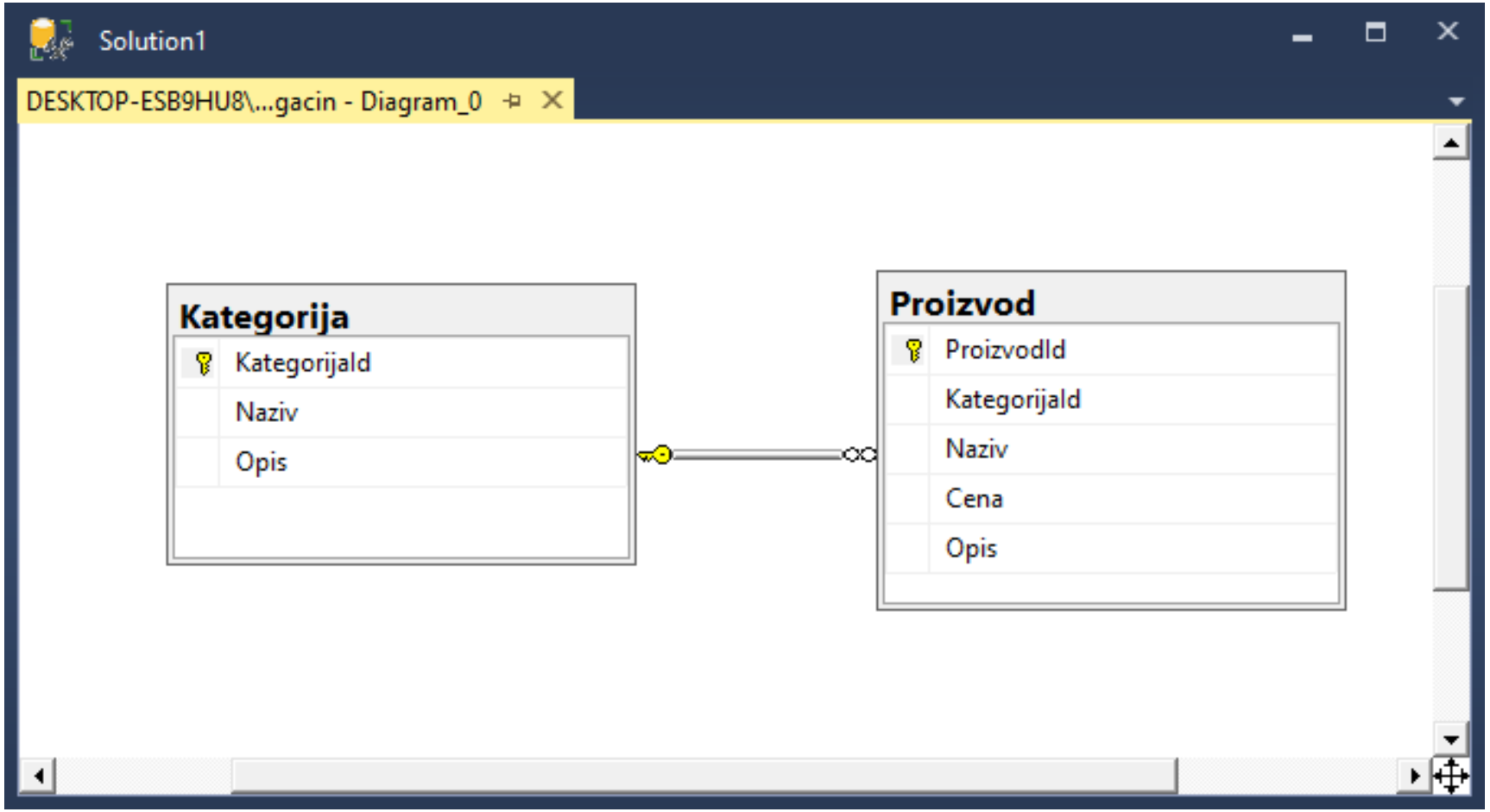


EF Core Power Tools

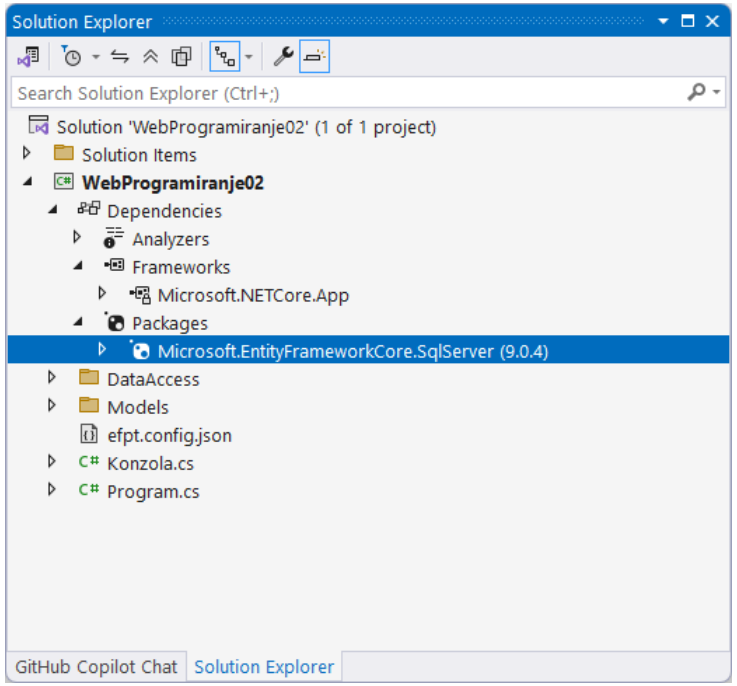
Extensios -> Manage Extensios



Baza Magacin



Solution Explorer



Desni klik na projekat pa opcija Edit

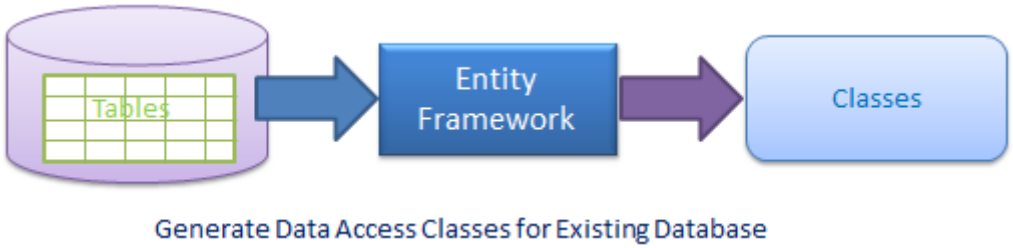
```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net9.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>disable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="9.0.4" />
  </ItemGroup>

</Project>
```

Entity Framework Core

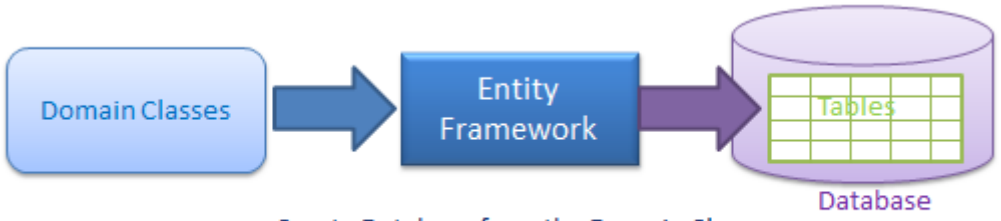
- EF Core je međuplatformska verzija Entity Frameworka
- EF Core je objektno-relacioni maper (O/RM) koji omogućava .NET programerima da rade sa bazom korišćenjem .NET objekata
- Omogućava pisanje upita pomoću LINQ-a
- Automatski prevodi C# kod u SQL upite
- Podržava različite baze podataka (SQL Server, PostgreSQL, SQLite…)
- Podržava migracije za upravljanje šemom baze

Strategije dizajna modela

- **Database-first pristup**
  - Najpre se dizajnira i kreira baza podataka, a zatim se na osnovu postojeće baze generišu entiteti i DbContext klase u aplikaciji
- **Model-first pristup**
  - Najpre se definišu entiteti (C# klase), a zatim EF Core generiše bazu podataka na osnovu modela



Generate Data Access Classes for Existing Database

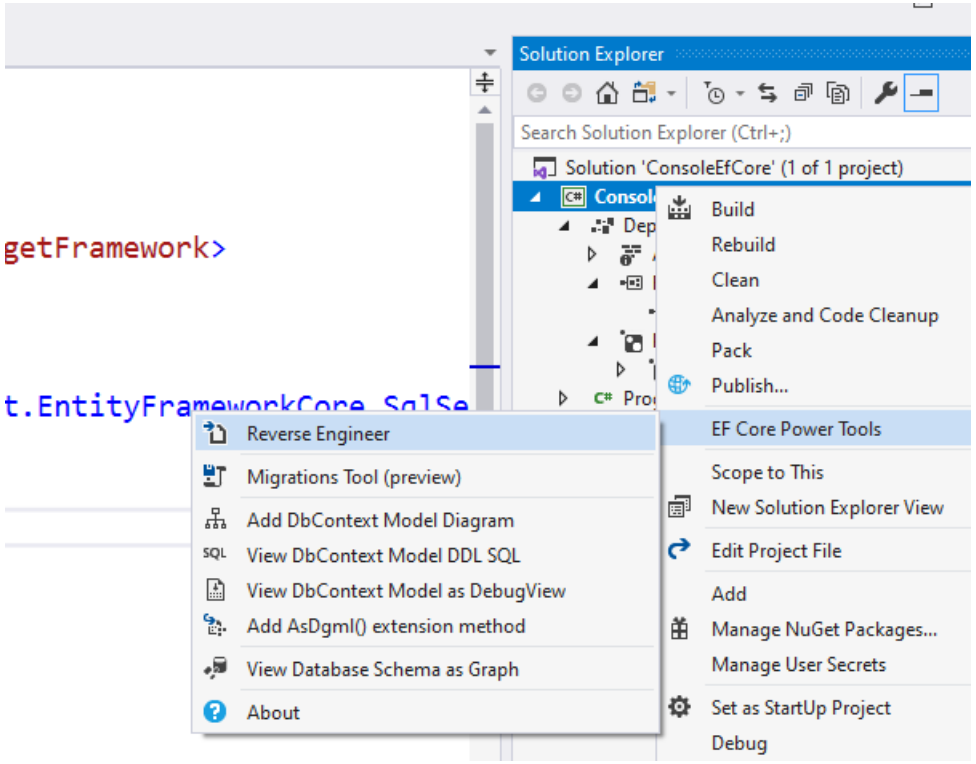


Create Database from the Domain Classes

EF Core Power Tools

- **EF Core Power Tools** je besplatan Visual Studio dodatak koji olakšava rad sa Entity Framework Core
- Instalira se preko Extensions → Manage Extensions → Browse → EF Core Power Tools → Install
- Omogućava generisanje entiteta i DbContext klase iz postojeće baze (Database-first)
- Pruža vizuelni prikaz modela (ER dijagram)
- Olakšava rad sa migracijama i SQL skriptama
- Pojednostavljuje konfiguraciju i analizu EF Core modela

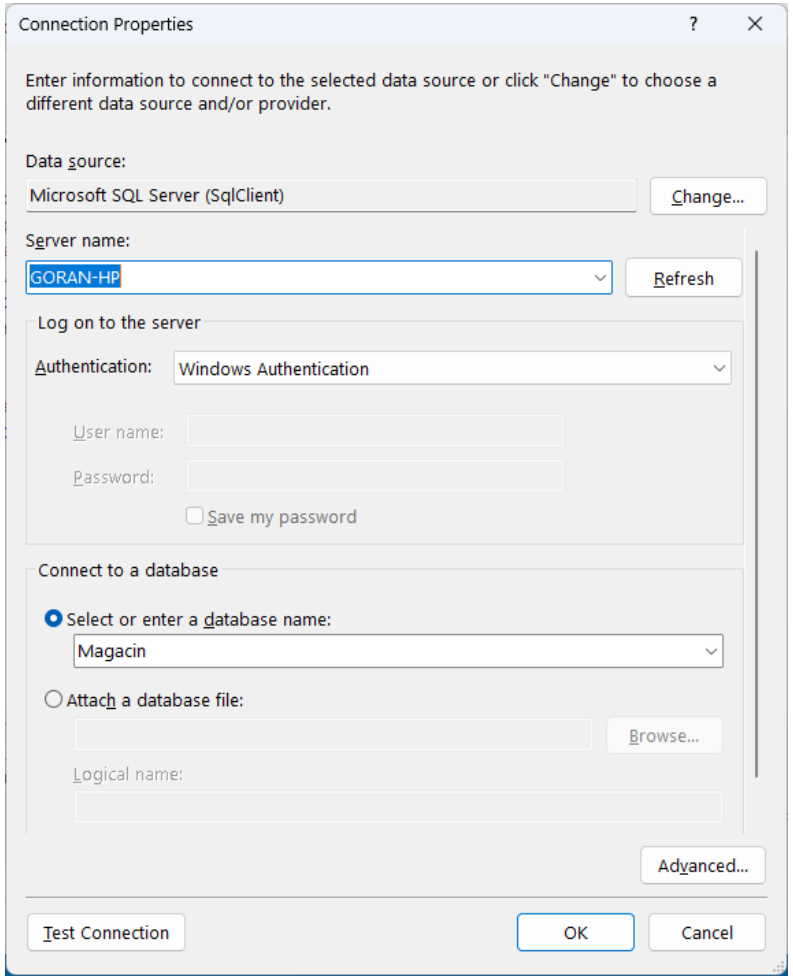
EF Core Power Tools

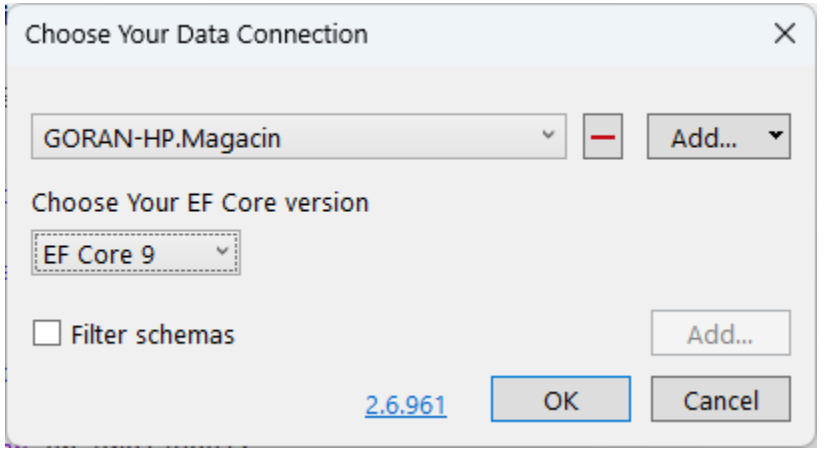


Reverse Engineer

- Generiše C# entitete i DbContext klasu iz postojeće baze podataka
- Koristi se u Database-first pristupu
- Automatski mapira tabele, kolone i relacije u klase
- Ubrzava razvoj kada već postoji gotova baza

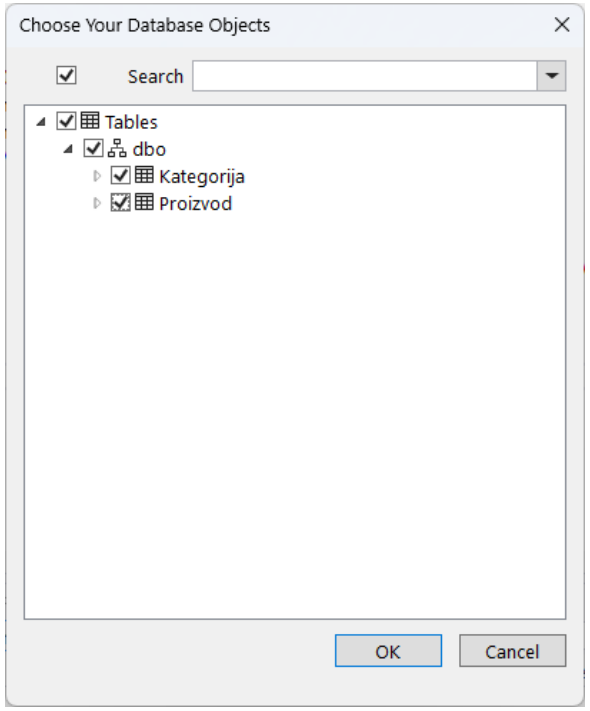
Odabir konekcije

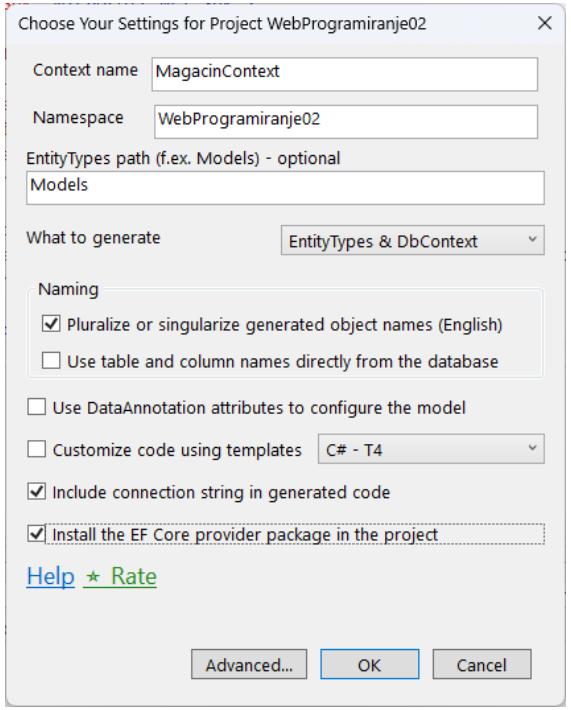




- Izbor servera i baze podataka
- Podešavanje autentifikacije
- Odabir EF Core verzije

Odabir tabela i kreiranje modela





- Izbor tabela iz baze podataka
- Definisanje naziva DbContext klase
- Odabir foldera za smeštanje model (entitet) klasa
- Generisanje entiteta i DbContext klase

Entitetska klasa Kategorija

```
[Table("Kategorija")]
public partial class Kategorija
{
    public int KategorijaId { get; set; }

    public string Naziv { get; set; }

    public string Opis { get; set; }

    public virtual ICollection<Proizvod> Proizvodi { get; set; } = new List<Proizvod>();
}
```

- **Proizvodi** je navigaciono svojstvo koje predstavlja vezu između Kategorije i njenih proizvoda
- Tipa je ICollection<Proizvod>, jer jedna kategorija može imati više proizvoda (1:N relacija)
- Omogućava pristup svim proizvodima koji pripadaju određenoj kategoriji

Entitetska klasa Proizvod

```
[Table("Proizvod")]
public partial class Proizvod
{
    public int ProizvodId { get; set; }

    public int KategorijaId { get; set; }

    public string Naziv { get; set; }

    public decimal Cena { get; set; }

    public string Opis { get; set; }

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

- Kategorija je navigaciono svojstvo koje predstavlja vezu proizvoda sa njegovom kategorijom (N:1 relacija)
- Omogućava pristup podacima povezane kategorije iz objekta Proizvod
- To je svojstvo tipa Kategorija, a ne definicija same klase

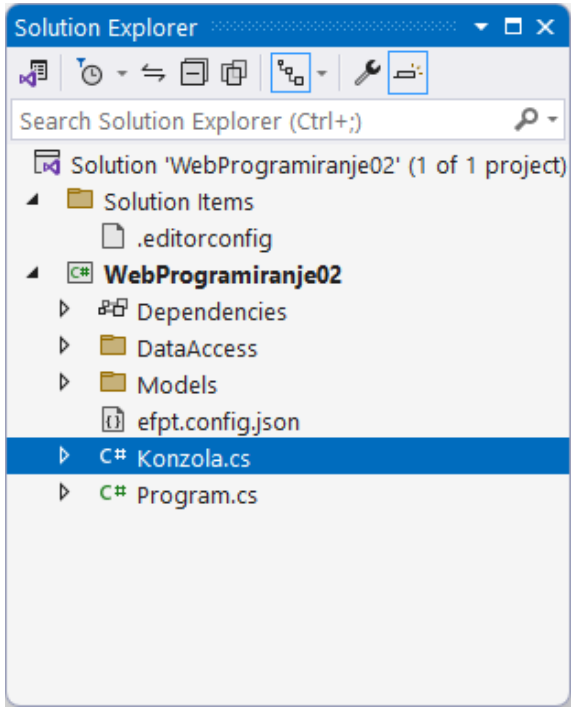
DbContext klasa

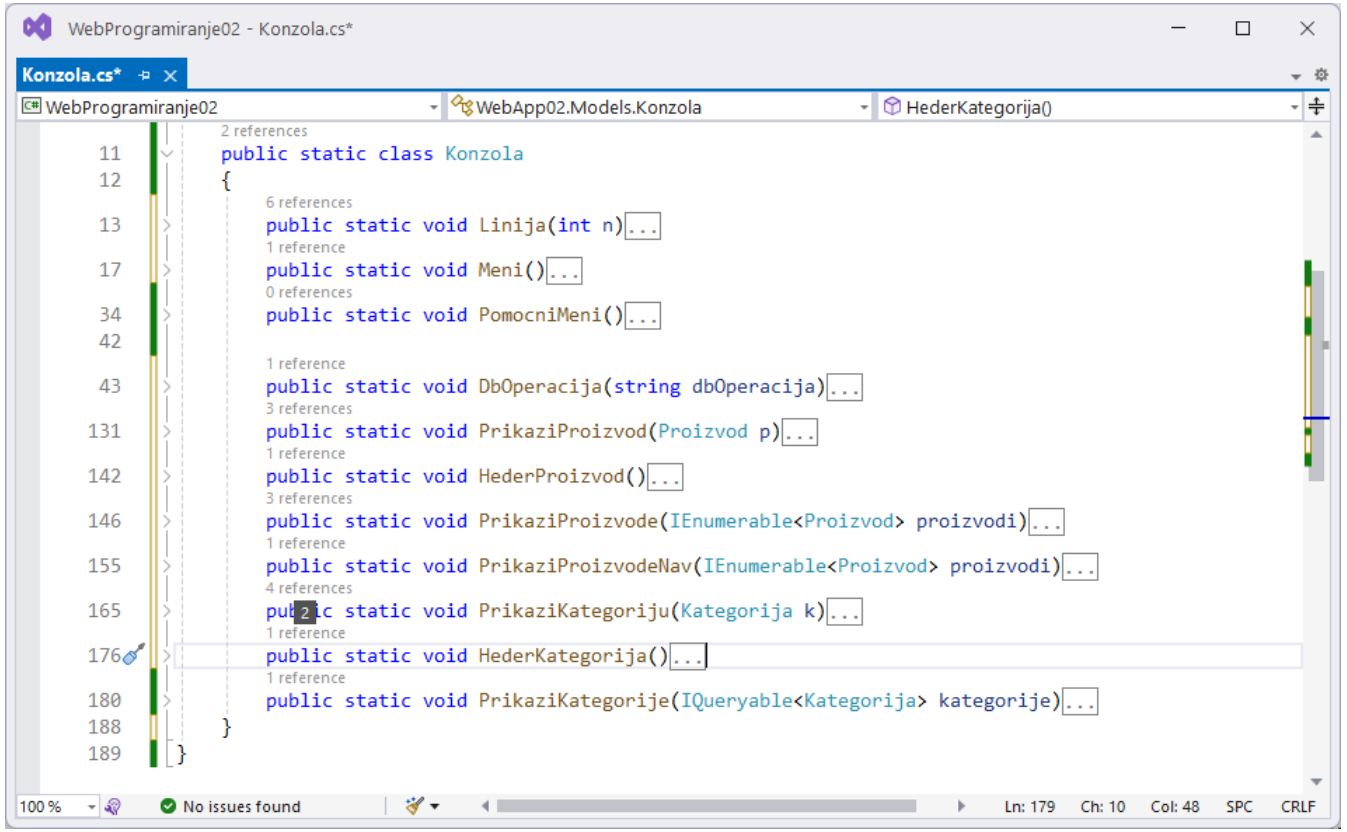
```
public partial class MagacinContext : DbContext
{
    public MagacinContext()
    {
    }

    public MagacinContext(DbContextOptions<MagacinContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Kategorija> Kategorije { get; set; }

    public virtual DbSet<Proizvod> Proizvodi { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        => optionsBuilder.UseSqlServer("Data Source=GORAN-HP;
          Initial Catalog=Magacin;Integrated Security=True;Encrypt=False");
}
```

DbContext klasa

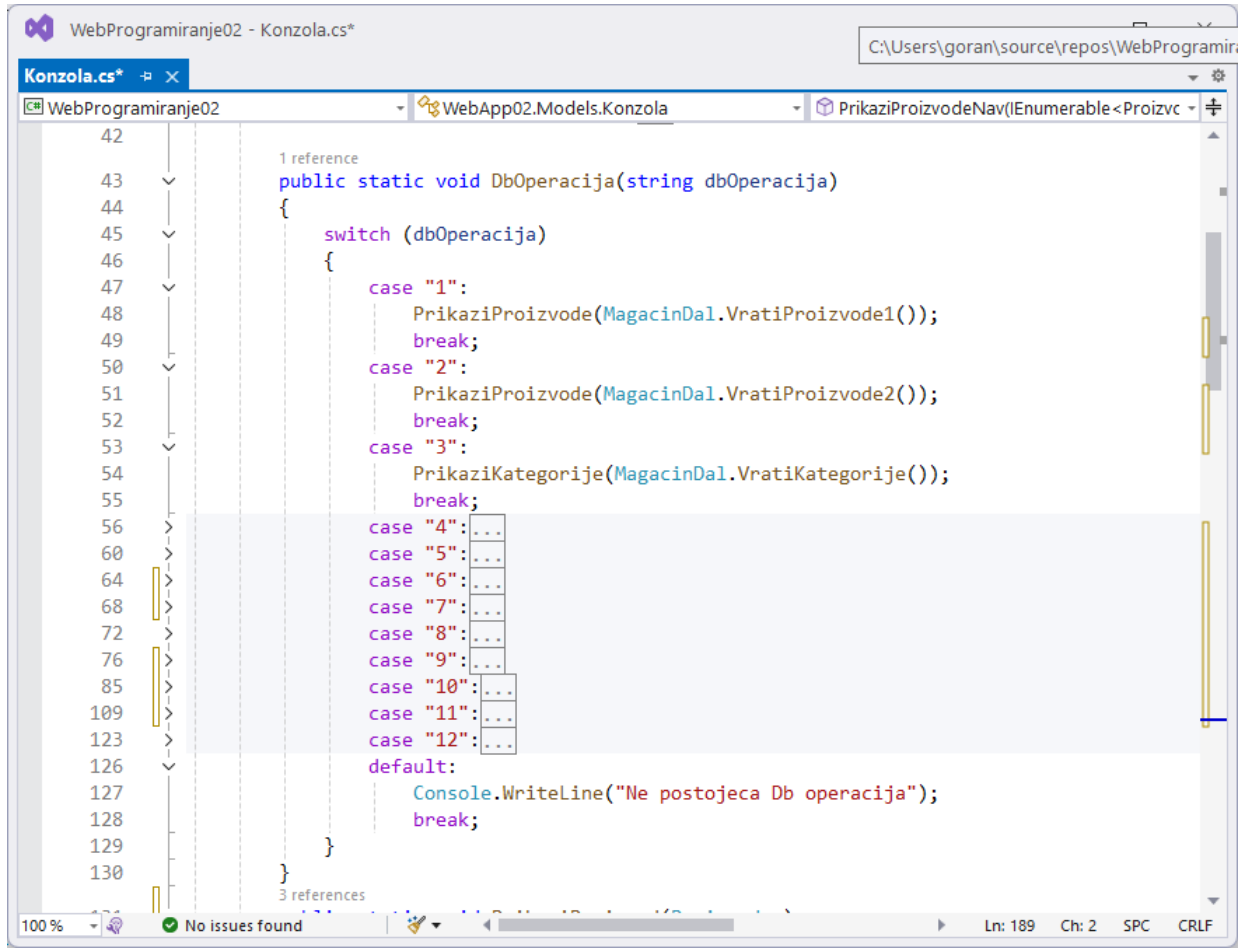
- **DbContext** klasa je most između entitetskih klasa i baze podataka
- **DbSet<TEntity>** predstavlja kolekciju entitetskih objekata koji odgovaraju redovima tabele u bazi podataka
- DbContext sadrži svojstva za pristup DbSet-ovima
- Postoji po jedan **DbSet** za svaku tabelu u bazi podataka
- DbContext klasa upravlja konekcijom sa bazom podataka
- DbContext klasa prati promene u objektima unutar DbSet-ova
- DbContext klasa čuva promene u bazi podataka pozivom metode **SaveChanges**()

Klasa Konzola za komunikaciju sa konzolom





Klasa Konzola metoda DbOperacija



Klasa Konzola metoda PrikaziProizvod

```
public static void PrikaziProizvod(Proizvod p)
{
    if (p != null)
    {
        Console.WriteLine(p.ProizvodId.ToString().PadRight(2, ' ') + "|\t" +
        p.Naziv.PadRight(50, ' ') + "|\t" + p.Cena);
    }
    else
    {
        Console.WriteLine("Ne postoji proizvod sa datim ID-om.");
    }
}
```

Klasa Konzola metoda PrikaziProizvode

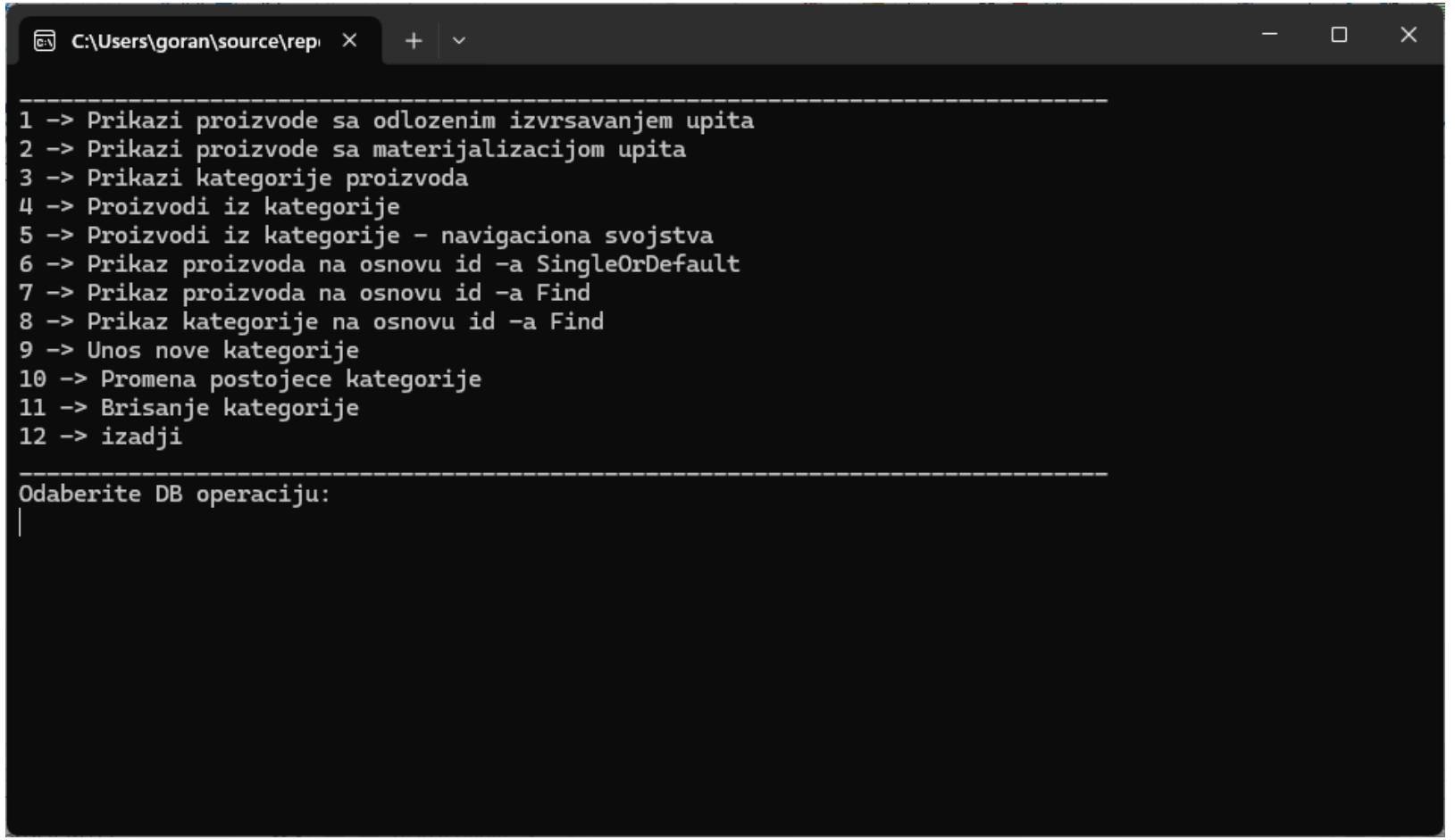
```
public static void HederProizvod()
{
    Console.WriteLine("ID|" + "\t" + "Naziv".PadRight(50, ' ') + "|\t" + "Cena");
}
```

```
public static void PrikaziProizvode(IEnumerable<Proizvod> proizvodi)
{
    HederProizvod();
    Linija(80);
    foreach (var p in proizvodi)
    {
        PrikaziProizvod(p);
    }
}
```

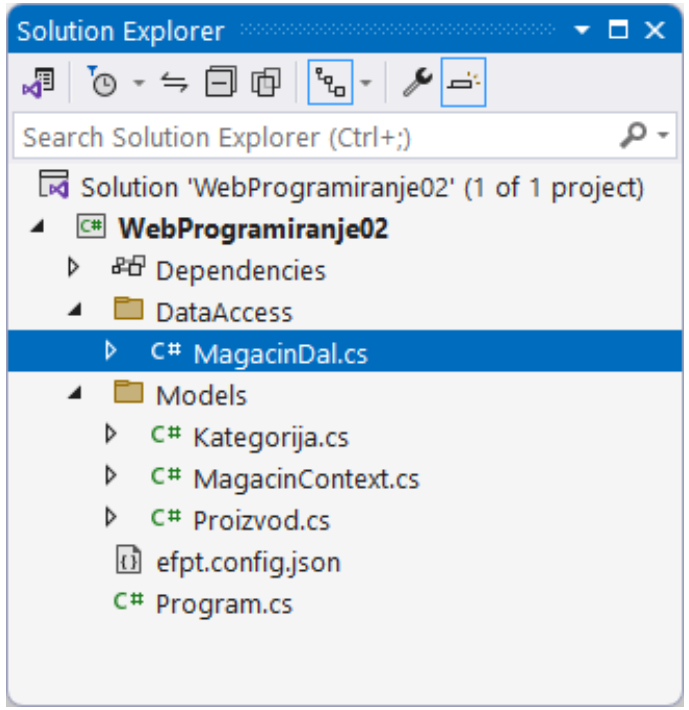
Fajl Program.cs

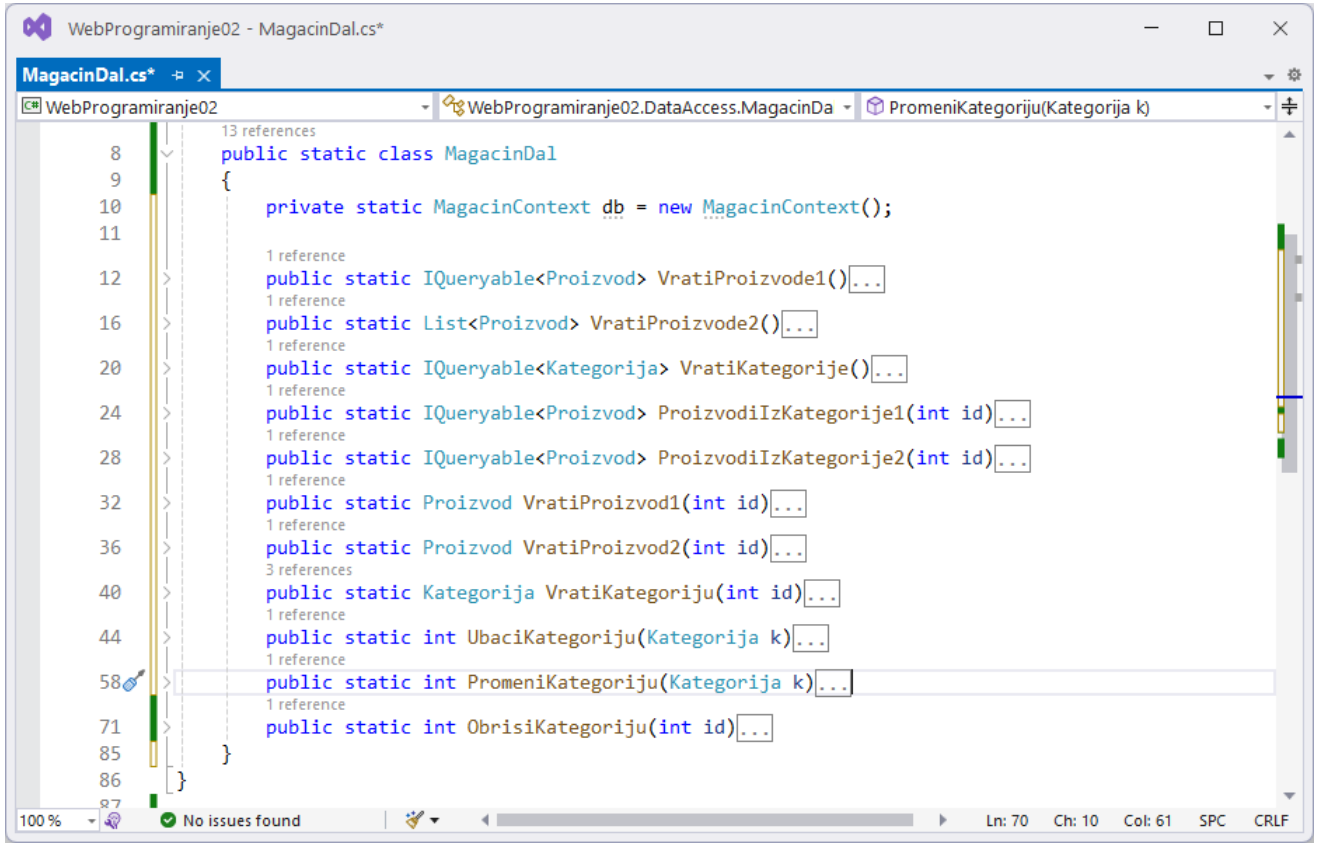
```
string dbOperacija;

do
{
    Konzola.Meni();
    Console.WriteLine("Odaberite DB operaciju:");
    dbOperacija = Console.ReadLine();
    Konzola.DbOperacija(dbOperacija);
} while (dbOperacija != "12");
```

Početni prozor aplikacije



Klasa za komunikaciju sa bazom folder DataAccess klasa MagacinDal





Čitanje podataka sa odloženim izvršavanjem upita

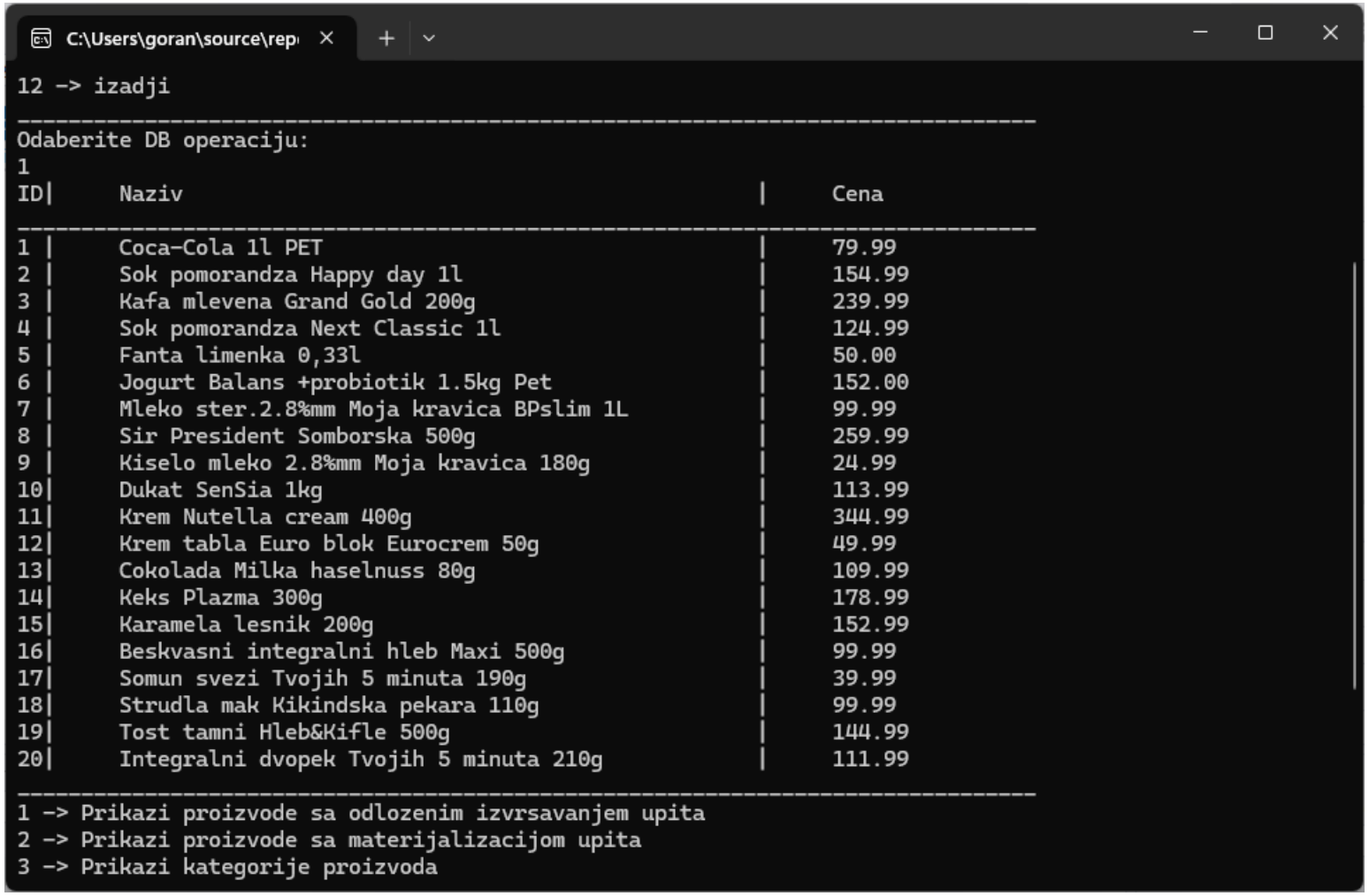
```
private static MagacinContext db = new MagacinContext();
```

```
public static IQueryable<Proizvod> VratiProizvode1()
{
    return db.Proizvodi;
}
```

- Metoda vraća IQueryable<Proizvod>, što znači da vraća upit koji treba da se izvrši nad bazom podataka
- U trenutku kada se poziva ova metoda, upit se još uvek nije izvršio
- Upit se izvršava tek u trenutku enumeracije (npr. pozivom ToList(), First(), Count()...).

Prikaz proizvoda

```
PrikaziProizvode(MagacinDal.VratiProizvode1());
```



Čitanje podataka uz materijalizaciju upita

```
public static List<Proizvod> VratiProizvode2()
{
    return db.Proizvodi.ToList();
}
```

- Upit se izvršava odmah i rezultati se vraćaju kao lista
- Kada se metoda pozove, izvršava se upit nad bazom podataka i svi rezultati se odmah učitavaju u memoriju

Čitanje kategorija proizvoda

```
public static IQueryable<Kategorija> VratiKategorije()
{
    return db.Kategorije;
}
```

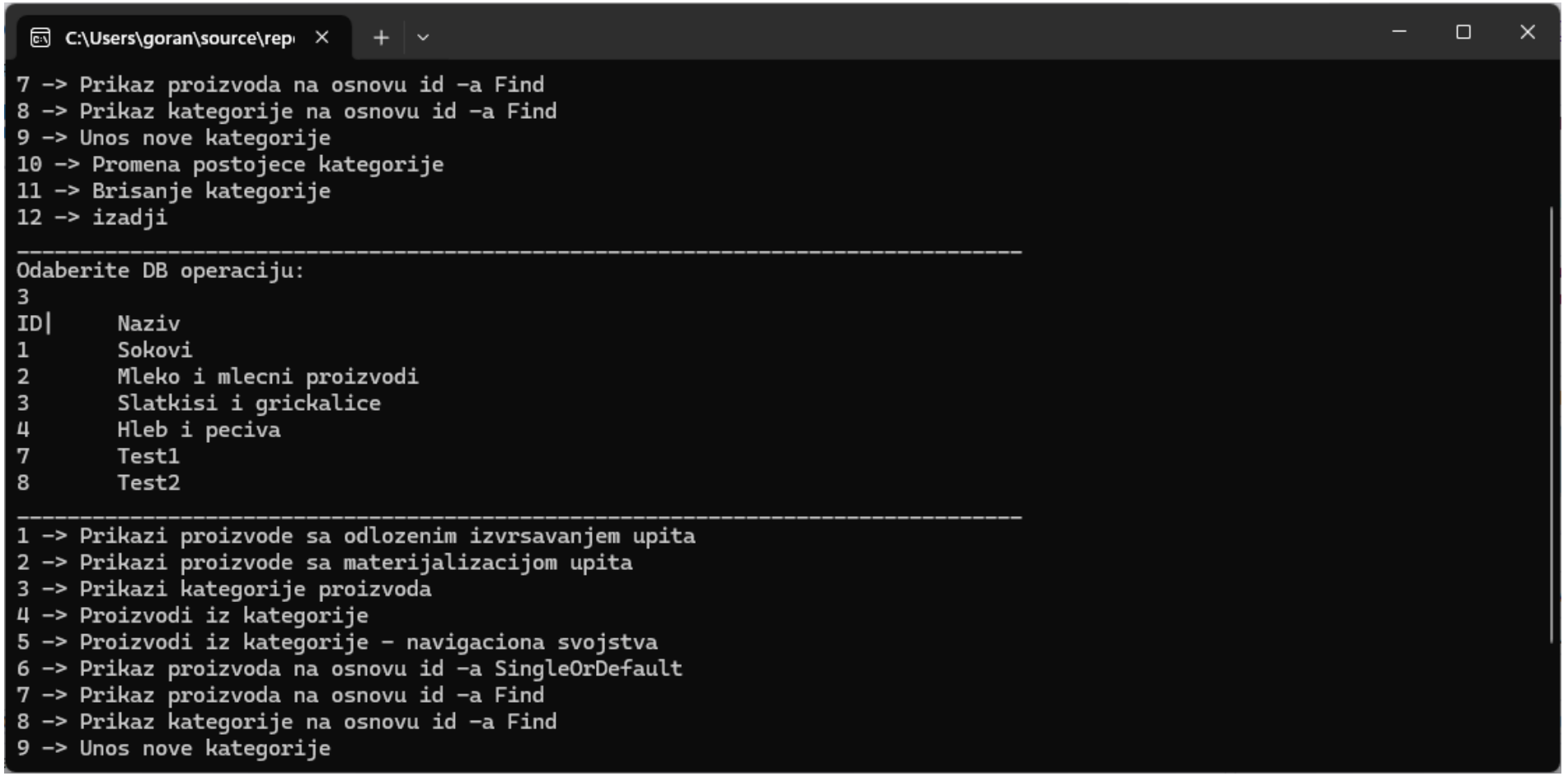
Prikaz kategorije u klasi Konzola

```
public static void PrikaziKategoriju(Kategorija k)
{
    if (k != null)
    {
        Console.WriteLine(k.KategorijaId + "\t" + k.Naziv);
    }
    else
    {
        Console.WriteLine("Ne postoji kategorija.");
    }
}
```

Prikaz kategorija u klasi Konzola

```
public static void HederKategorija()
{
    Console.WriteLine("ID|" + "\t" + "Naziv");
}
public static void PrikaziKategorije(IQueryable<Kategorija> kategorije)
{
    HederKategorija();
    foreach (var k in kategorije)
    {
        PrikaziKategoriju(k);
    }
}
```

Prikaz kategorija



Filtriranje podataka

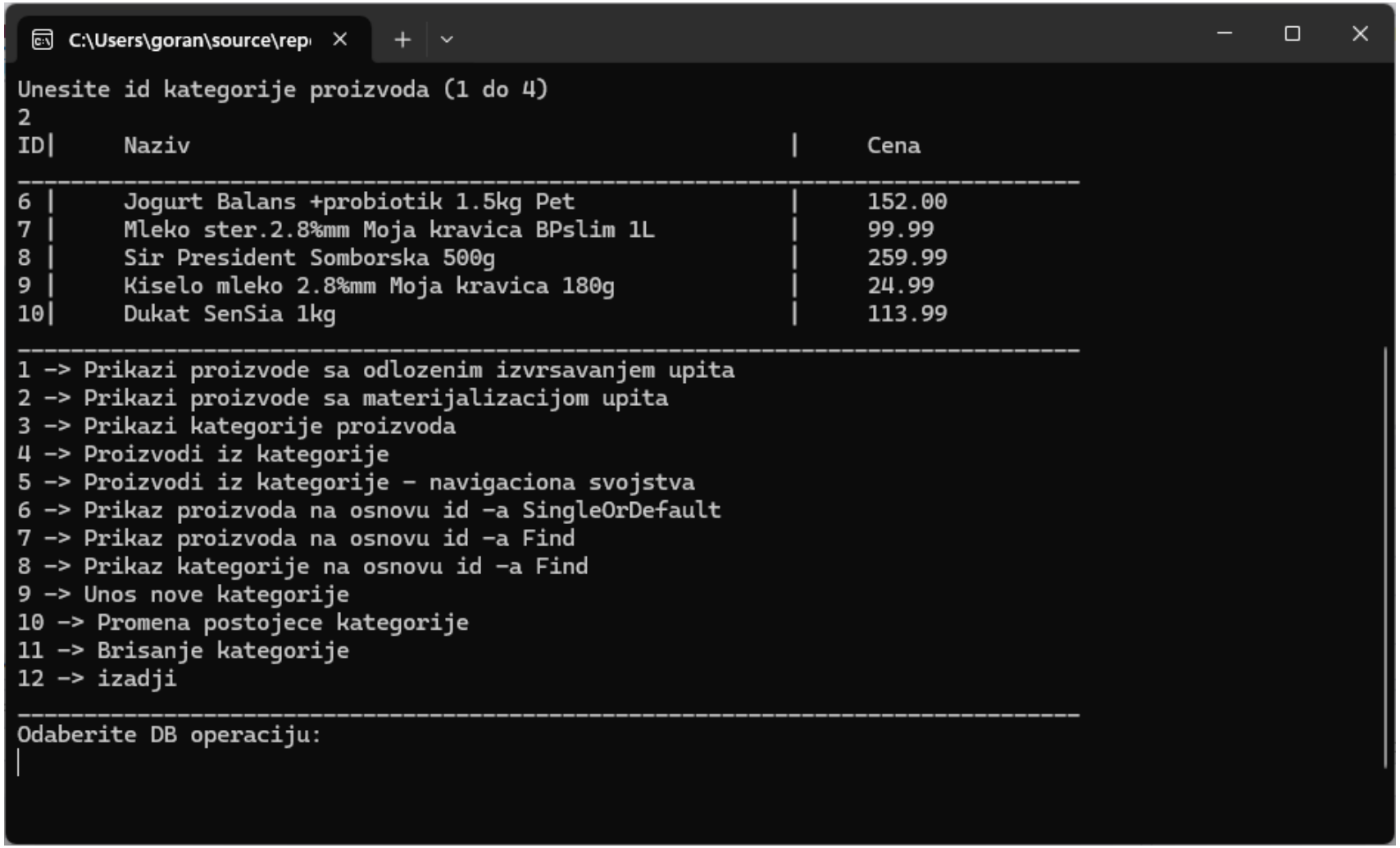
```
public static IQueryable<Proizvod> ProizvodiIzKategorije1(int id)
{
    return db.Proizvodi.Where(p => p.KategorijaId == id);
}
```

- Metoda vraća upit tipa IQueryable<Proizvod>
- Upit sadrži uslov filtriranja definisan pomoću metode Where()
- Upit se izvršava tek kada se pozove npr. ToList()

Filtriranje podataka

```
case "4":
    Console.WriteLine("Unesite id kategorije proizvoda (1 do 4)");

PrikaziProizvode(MagacinDal.ProizvodiIzKategorije1(int.Parse(Console.ReadLine())));
 break;
```

Prikaz proizvoda iz kategorije



Upotreba navigacionog svojstva

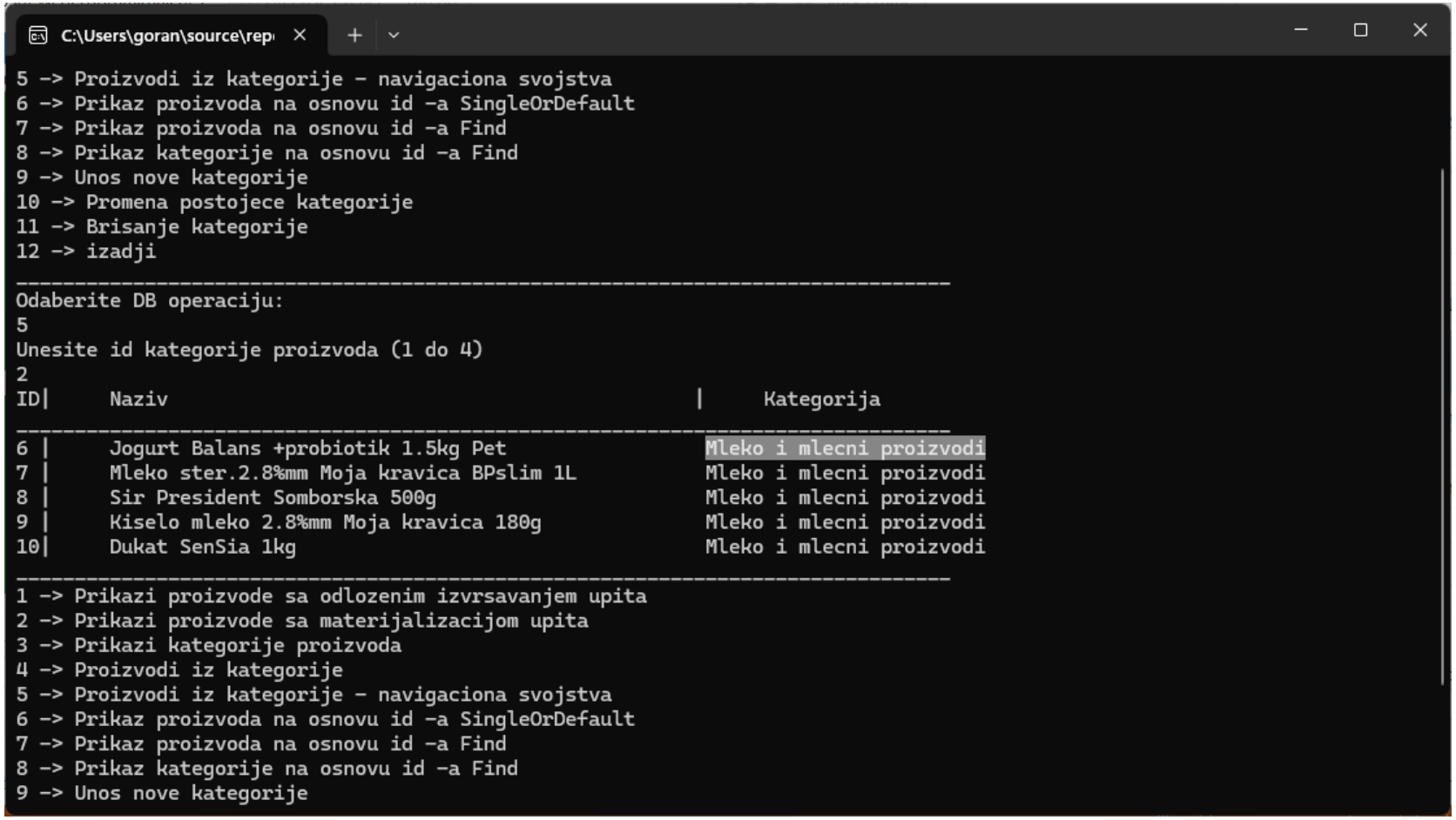
```
public static IQueryable<Proizvod> ProizvodiIzKategorije2(int id)
{
    return db.Proizvodi.Include(p => p.Kategorija).Where(p => p.KategorijaId == id);
}
```

- Include() je metoda koja služi za učitavanje povezanih entiteta
- Koristi se nad navigacionim svojstvima
- Omogućava da se povezani podaci učitaju zajedno sa glavnim entitetom
- EF Core generiše JOIN u SQL upitu

Prikaz podataka iz povezane tabele - klasa Konzola

```
public static void PrikaziProizvodeNav(IEnumerable<Proizvod> proizvodi)
{
    Console.WriteLine("ID|" + "\t" + "Naziv".PadRight(50, ' ') + "|\t" + "Kategorija");
    Linija(80);
    foreach (Proizvod p in proizvodi)
    {
        Console.WriteLine(p.ProizvodId.ToString().PadRight(2, ' ') + "|\t" +
        p.Naziv.PadRight(50, ' ') + " " + p.Kategorija.Naziv);
    }

}
```

Prikaz podataka iz povezane tabele



Čitanje jednog reda

```
public static Proizvod VratiProizvod1(int id)
{
    return db.Proizvodi.SingleOrDefault(p => p.ProizvodId == id);
}
```

- Metoda vraća jedan objekat tipa Proizvod na osnovu primarnog ključa
- Koristi se metoda **SingleOrDefault**() za pronalaženje jednog reda
- Upit se izvršava odmah (nema odloženog izvršavanja)
- Ako ne postoji odgovarajući red, metoda vraća null

Čitanje jednog reda

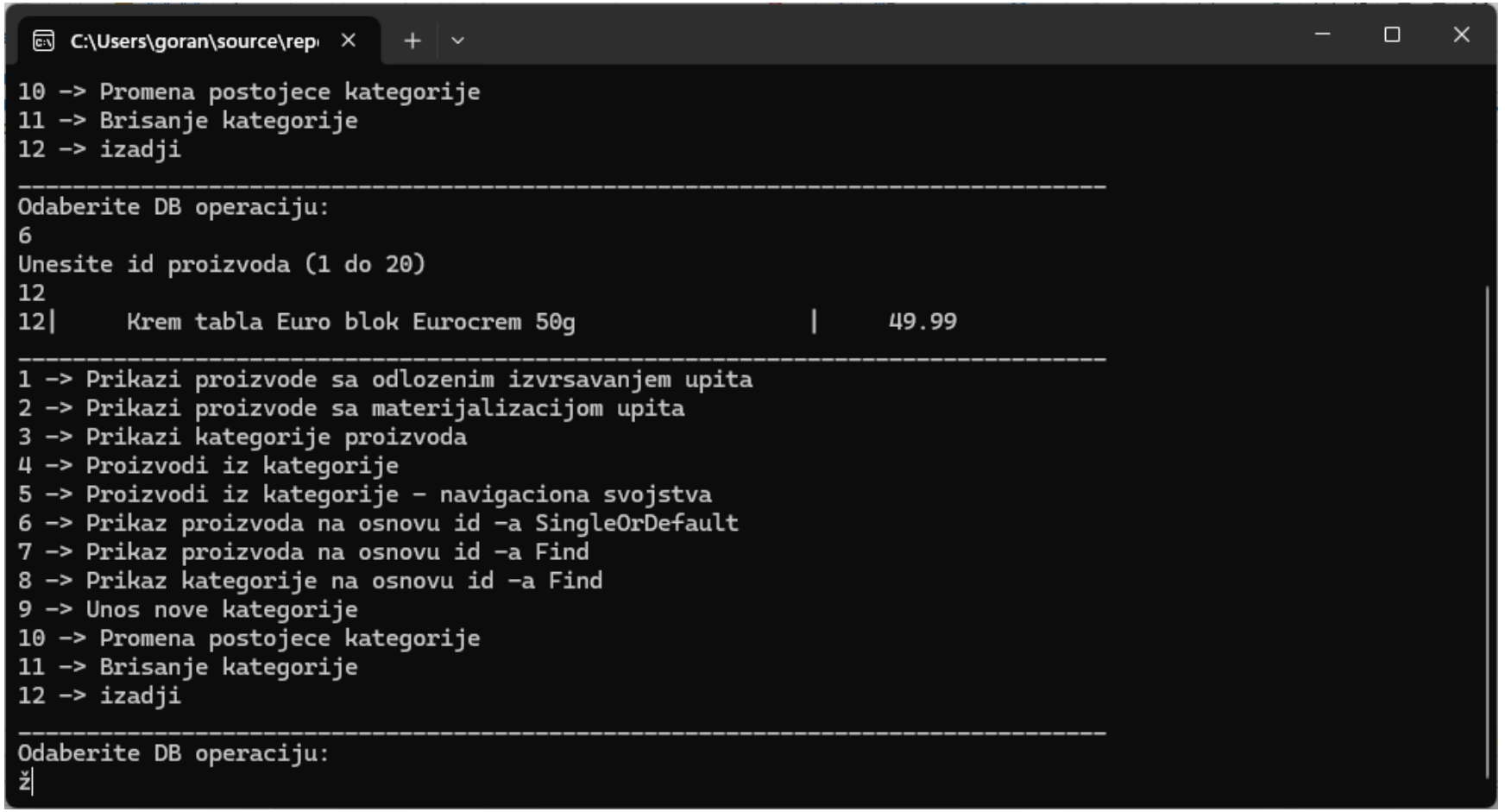
```
public static Proizvod VratiProizvod2(int id)
{
    return db.Proizvodi.Find(id);
}
```

- Find() je metoda klase DbSet<TEntity>
- Koristi se za pronalaženje entiteta po primarnom ključu
- Ako je entitet već učitan u kontekst, vraća se iz memorije
- U suprotnom, šalje se upit ka bazi podataka
- Ako entitet sa datim ID-jem ne postoji, metoda vraća null

Čitanje jednog reda

```
case "6":
    Console.WriteLine("Unesite id proizvoda (1 do 20)");
    PrikaziProizvod(MagacinDal.VratiProizvod1(int.Parse(Console.ReadLine())));
    break;
case "7":
    Console.WriteLine("Unesite id proizvoda (1 do 20)");
    PrikaziProizvod(MagacinDal.VratiProizvod2(int.Parse(Console.ReadLine())));
    break;
```

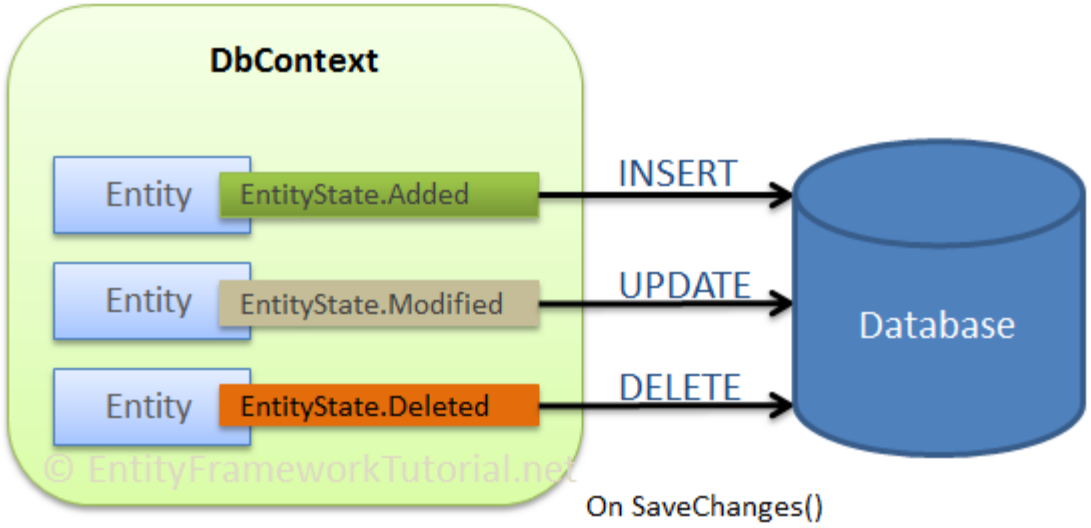
Čitanje jednog proizvoda



Ažuriranje podataka korišćenjem EF

- DbContext sadrži entitetske objekte koji predstavljaju podatke iz baze
- Izmene se prvo vrše nad entitetima unutar DbContext-a
- DbContext prati stanje entiteta (EntityState)
- Promene se upisuju u bazu pozivom metode SaveChanges()
- Pri pozivu metode SaveChanges(), u zavisnosti od stanja entiteta generiše se:
  - INSERT komanda ako je stanje Added
  - UPDATE komanda ako je stanje Modified
  - DELETE komanda ako je stanje Deleted

Ažuriranje podataka korišćenjem EF



Atačovani (Attached) i detačovani (Detached) entiteti

- **Attached** (atačovan) entitet je povezan sa DbContext-om
  - DbContext ga prati (Change Tracking)
  - Izmene nad njim mogu se sačuvati pozivom SaveChanges()
- **Detached** (detačovan) entitet nije povezan sa DbContext-om
  - DbContext ga ne prati
  - Izmene se neće sačuvati dok se entitet ponovo ne prikači na kontekst

Unos podataka

```
public static int UbaciKategoriju(Kategorija k)
{
    try
    {
        db.Add(k);
        db.SaveChanges();
        return k.KategorijaId;
    }
    catch (Exception xcp)
    {
        Console.WriteLine(xcp.Message);
        return -1;
    }
}
```

- **Add()** je metoda DbContext-a koja dodaje entitet u kontekst
- Entitet dobija stanje **Added**
- Pozivom **SaveChanges()** izvršava se INSERT komanda

Unos nove kategorije

```
case "9":
    Console.WriteLine("Unesite naziv nove kategorije");
    string naziv = Console.ReadLine();
    Console.WriteLine("Unesite opis nove kategorije");
    string opis = Console.ReadLine();
    Kategorija novaKategorija = new Kategorija { Naziv = naziv, Opis = opis };
    int idNoveKategorije = MagacinDal.UbaciKategoriju(novaKategorija);
    Console.WriteLine("Ubacena kategorija ciji je id: " + idNoveKategorije);
    break;
```

Ažuriranje reda

```
public static int PromeniKategoriju(Kategorija k)
{
    try
    {
        Kategorija postojeca = db.Kategorije.Find(k.KategorijaId);
        if (postojeca == null)
            return -1;
        postojeca.Naziv = k.Naziv;
        postojeca.Opis = k.Opis;

        db.SaveChanges();
        return 0;
    }
    catch (Exception)
    {
        return -1;
    }
}
```

- Pomoću Find() pronalazi se postojeći entitet na osnovu Id-a
- Menjaju se svojstva već atačovanog objekta
- SaveChanges() izvršava UPDATE u bazi

Promena postojeće kategorije

```
case "10":
    Console.WriteLine("Unesite id kategorije koju menjate");
    int id = int.Parse(Console.ReadLine());
    Kategorija kategorijaZaIzmenu = MagacinDal.VratiKategoriju(id);
    PrikaziKategoriju(kategorijaZaIzmenu);
    if (kategorijaZaIzmenu != null)
    {
        Console.WriteLine("Unesite novi naziv kategorije");
        naziv = Console.ReadLine();
        Console.WriteLine("Unesite novi opis kategorije");
        opis = Console.ReadLine();
        kategorijaZaIzmenu.Naziv = naziv;
        kategorijaZaIzmenu.Opis = opis;
        int rezultatIzmene = MagacinDal.PromeniKategoriju(kategorijaZaIzmenu);
        if (rezultatIzmene == 0)
        {
            Console.WriteLine("Promenjena kategorija");
        }
        else
        {
            Console.WriteLine("Greska pri promeni");
        }
    }
    break;
```

Brisanje reda

```
public static int ObrisiKategoriju(int id)
{
    try
    {
        Kategorija k = db.Kategorije.Find(id);
        db.Remove(k);    // metoda DbContext-a
        db.SaveChanges();
        return 0;
    }
    catch (Exception)
    {
        return -1;
    }
}
```

- Metoda najpre pronalazi entitet pomoću Find() metode klase DbSet<Kategorija>
- Pozivom Remove() metode klase DbContext entitet dobija stanje Deleted
- Pozivom SaveChanges() metode klase DbContext izvršava se DELETE komanda u bazi
- Metoda vraća 0 ako je brisanje uspešno, odnosno -1 u slučaju greške

Brisanje reda

```
case "11":
    Console.WriteLine("Unesite id kategorije koju brisete");
    id = int.Parse(Console.ReadLine());
    int rezultatBrisanja = MagacinDal.ObrisiKategoriju(id);
    if (rezultatBrisanja == 0)
    {
        Console.WriteLine("Obrisana kategorija ciji je id: " + id);
    }
    else
    {
        Console.WriteLine("Greska pri brisanju kategorije");
    }
    PrikaziKategoriju(MagacinDal.VratiKategoriju(id));
    break;
```

Klase Task i Task<T>

- Klasa **Task** se nalazi u biblioteci **System.Threading.Tasks**
- Klasa Task se koristi za izvršavanje više poslova istovremeno
- Vremenski zahtevni zadaci se obično izvršavaju u posebnoj programskoj niti
- **Asinhrona** operacija je operacija koja se izvršava nezavisno od glavnog toka programa, bez blokiranja njegove daljnje izvršavanja
- Klasa **Task** predstavlja asinhronu operaciju koja ne vraća vrednost
- Klasa **Task<T>** predstavlja asinhronu operaciju koja vraća rezultat

Vraćanje vrednosti iz zadatka

- Koristi se klasa **Task<TResult>**
- Rezultat se dobija pomoću **Result** svojstva
- Zadatak t1 se izvršava u drugoj niti
- Svojstvo **Result** blokira trenutnu nit dok se zadatak ne završi

```
private void button1_Click(object sender, EventArgs e)
{
    Task<string> t1 = Task.Run(
    () =>
    {
        Thread.Sleep(5000);
        return DateTime.Now.ToLongTimeString();
    }
    );

    // UI je blokitan dok se zadatak ne zavrsi
    label1.Text = t1.Result;
}
```

Korišćenje operatora **async** i **await**

- Modifikator **async** označava da je metoda asinhrona
- Unutar asinhrone metode koristi se operator **await** koji pauzira izvršavanje metode dok se zadatak ne završi
- Nakon završetka zadatka izvršavanje metode se nastavlja od mesta gde je korišćen **await**
- Operatori async i await omogućavaju pozivanje asinhronih metoda i čekanje rezultata bez blokiranja programske niti

Izvršavanje asinhrone operacije u UI programskoj niti

```
private async void button1_Click(object sender, EventArgs e)
{
    Task<string> t1 = Task.Run(
        () =>
        {
            Thread.Sleep(5000);
            return DateTime.Now.ToLongTimeString();
        }
        );

    // Linija koda se izvrsava kada je rezultat raspoloziv
    // UI nije blokiran

    label1.Text = await t1;
}
```

Kreiranje awaitable metode

- Ako metoda vraća void, njen asinhroni ekvivalent vraća **Task**
- Ako metoda vraća tip T, njen asinhroni ekvivalent vraća **Task<T>**
- Metode sa povratnim tipom void mogu biti označene sa **async**, ali se to koristi samo za event handler metode

Asinhrona metoda

```
private async Task<string> CitajAsinhrono()
{
    Task<string> task1 = Task.Run(() =>
    {
        //simulacija dugotrajnog procesa
        Thread.Sleep(3000);
        return DateTime.Now.ToLongTimeString();
    });

    return await task1; // čeka završetak zadatka i vraća njegov rezultat
}
```

```
private async void button2_Click(object sender, EventArgs e)
{
    label1.Text = await CitajAsinhrono();
}
```

Pitanje 1

Kod Entity Frameworka Core tehnologije osnovna klasa koja se koristi kao bazna klasa za manipulaciju sa entitetskim objektima naziva se:

a. DbContext klasa
b. DataSet klasa
c. EntityContext klasa

Odogovor: a

Pitanje 2

Ako je **MagacinContext** klasa izvedena iz **DbContext klase.** Neka je **db** instanaca **MagacinContext** klase. Klasa **MagacinContext** ima svojstvo **Kategorije** pomoću koga se pristupa DbSet objektu tipa **DbSet<Kategorija>** koji se sastoji od objekata entitetske klase **Kategorija**.
Kako se pronalazi atačovani objekat čiji id ima vrednost 1:

a. Kategorija k1 = db.Kategorije.Find(k=>k.id=1);
b. Kategorija k1 = db.Kategorije.Select(1);
c. Kategorija k1 = db.Kategorije.Find(1);

Odogovor: c

Pitanje 3

Da bi se promene unutar tipiziranog DbContext objekta označenog sa db sačuvale u bazi podataka pišemo sledeću liniju koda:

a. db.Save();
b. db. SaveToDatabase();
c. db.SaveChanges();

Odogovor: c

Pitanje 4

Za kreiranje asinhrone operacije koja ne vraća vrednost koristi se klasa:

a. Thread
b. Task
c. Task<T>

Odogovor: b

Pitanje 5

Za asinhronu metodu koja vraća rezultat tipa string kao povratni tip se koristi:

a. Thread<string>
b. Task
c. Task<string>

Odogovor: c

Pitanje 6

Data je metoda klasa sledećeg potpisa:
private async Task<string> Citaj()

Na koji način se ova metoda pravilno poziva kako bismo dobili njen rezultat?

a. Citaj()
b. async Citaj()
c. await Citaj()

Odogovor: c