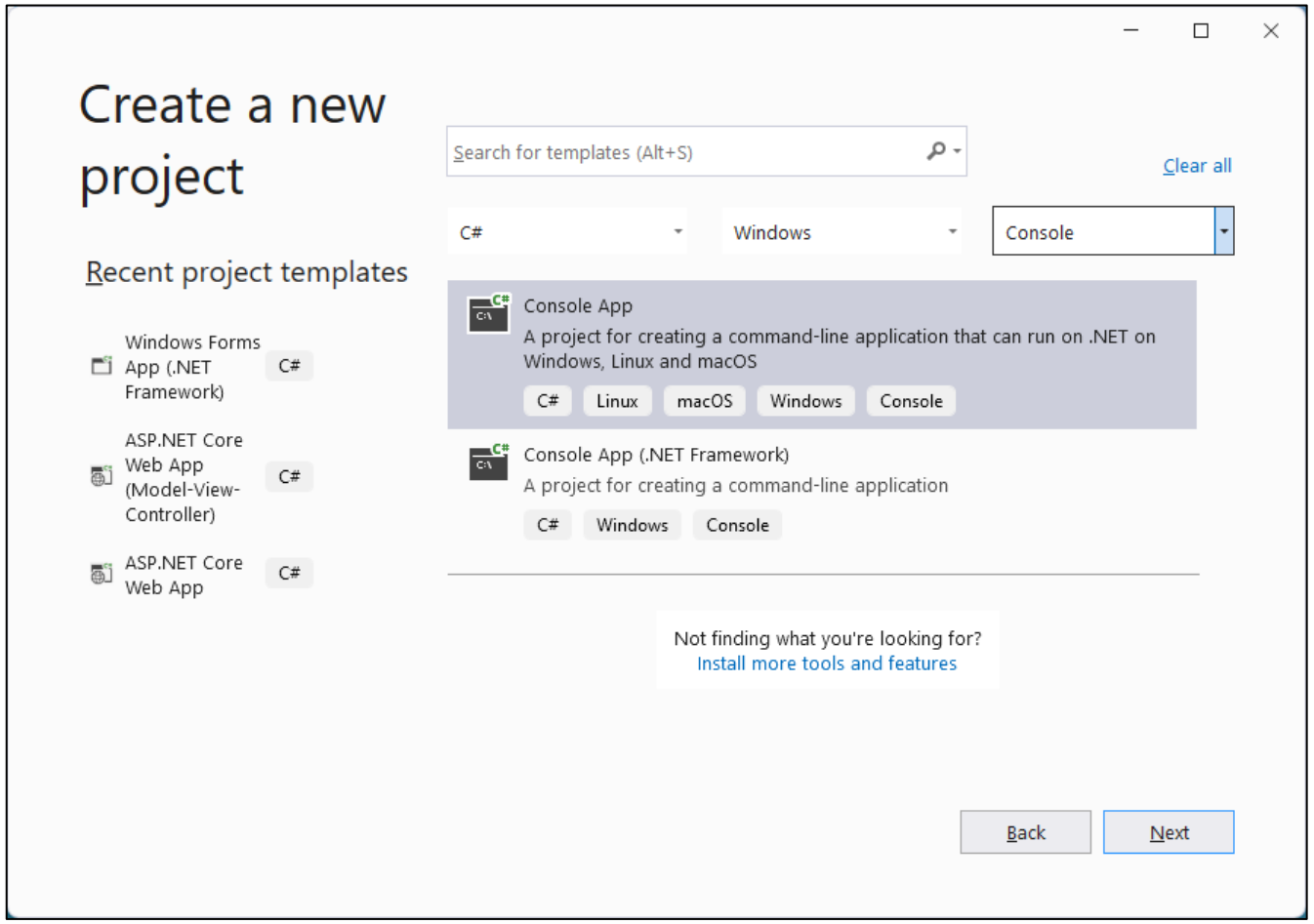
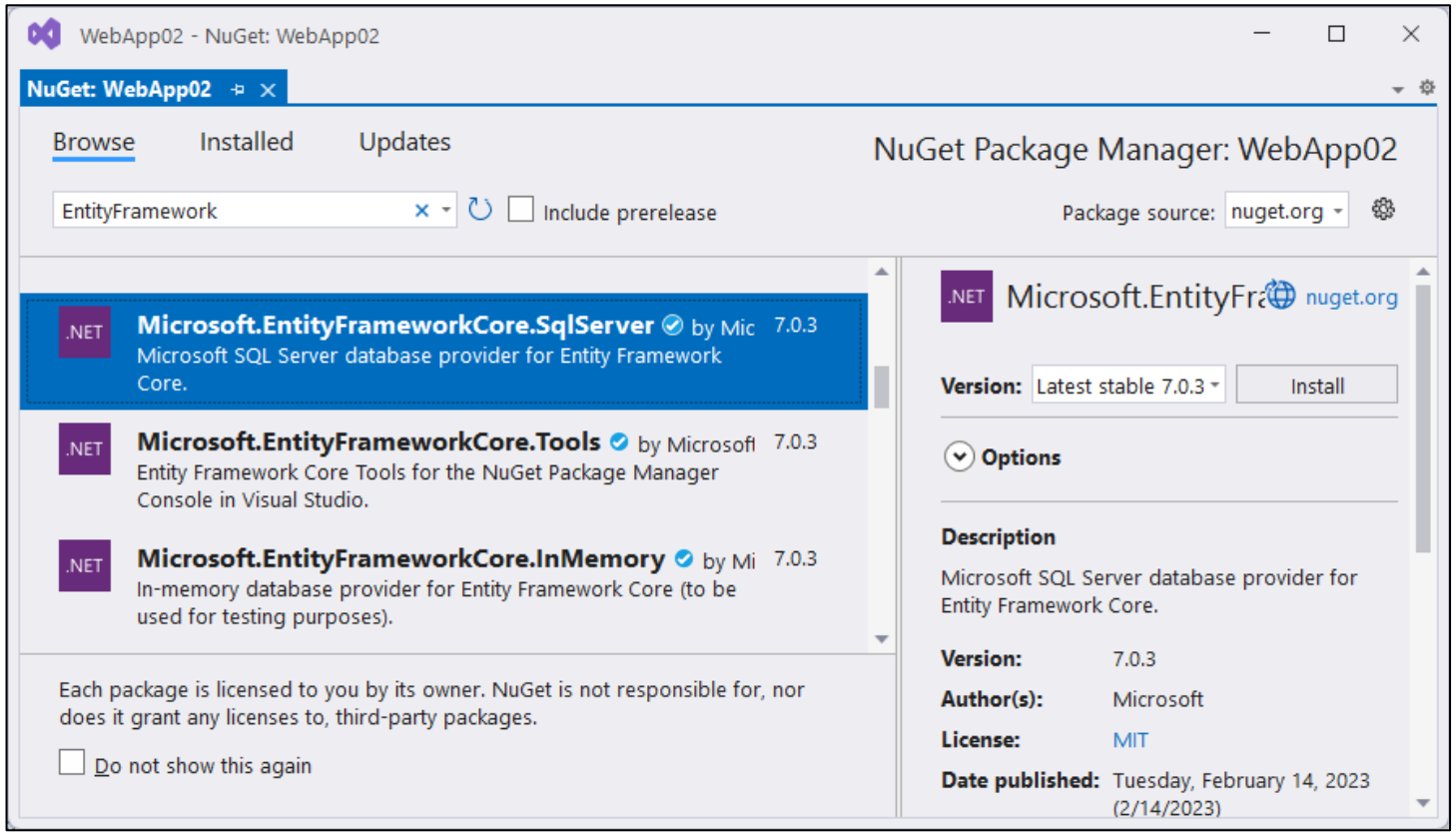
Entity Framework Core

.NET Core konzolna aplikacija



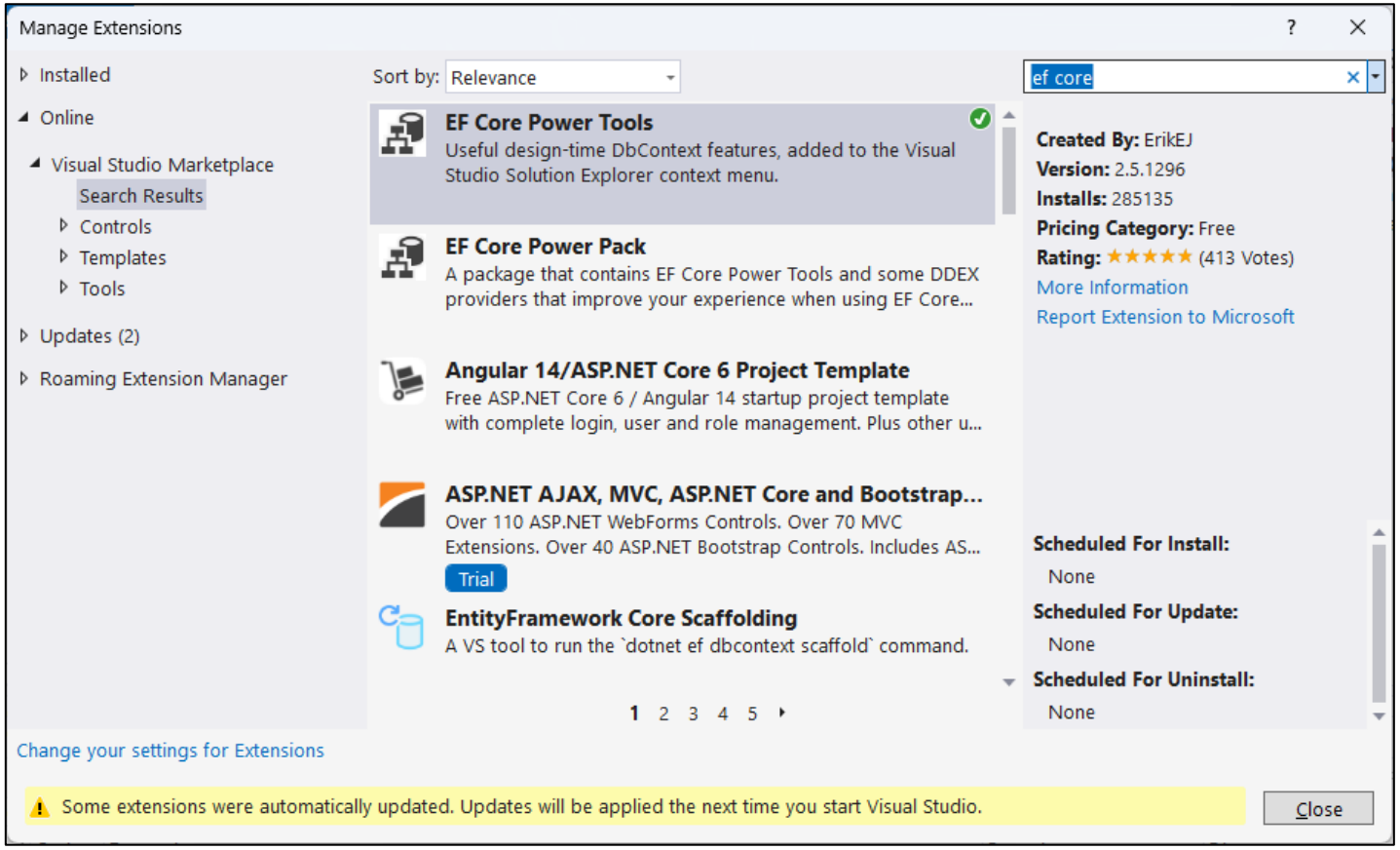
Microsoft.EntityFrameworkCore.SqlServer

- Project->Manage Nuget Packages

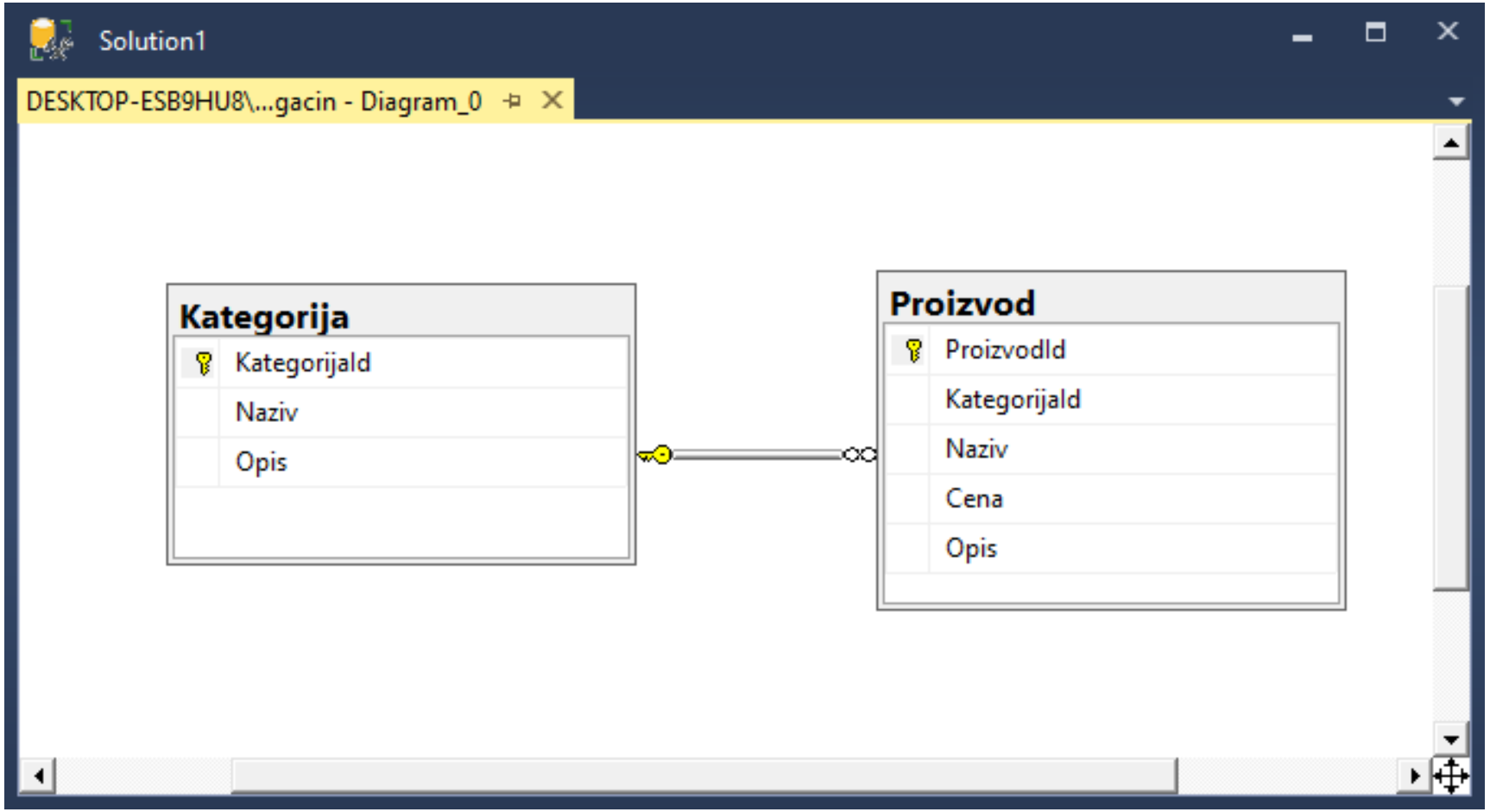


EF Core Power Tools

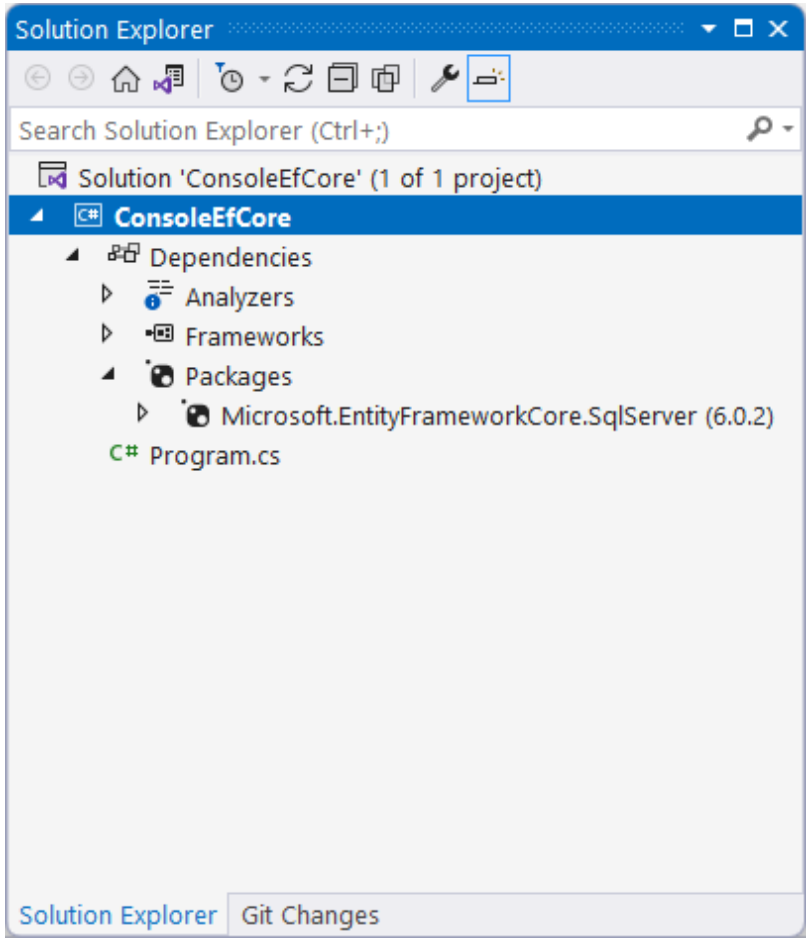
- Extensios -> Manage Extensios



Baza Magacin



Solution Explorer



Desni klik na projekat pa opcija Edit

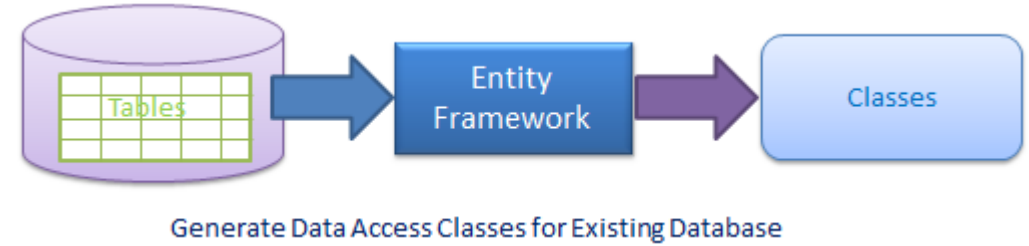
```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net7.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.3" />
  </ItemGroup>

</Project>
```

Entity Framework Core

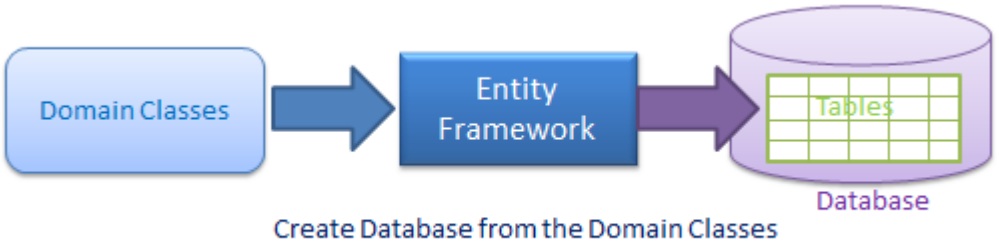
- EF Core je međuplatformska verzija Entity Frameworka
- EF Core je objektno-relacioni maper (O/RM) koji omogućava .NET programerima da rade sa bazom korišćenjem .NET objekata.

Strategije dizajna modela

- **Database-first design**, najpre se dizajnira i kreira baza podataka a zatim se generišu entiteti aplikacije
- **Model-first design**, najpre se dizajniraju entiteti za aplikaciju a zatim se kreira baza podataka na osnovu tih entiteta

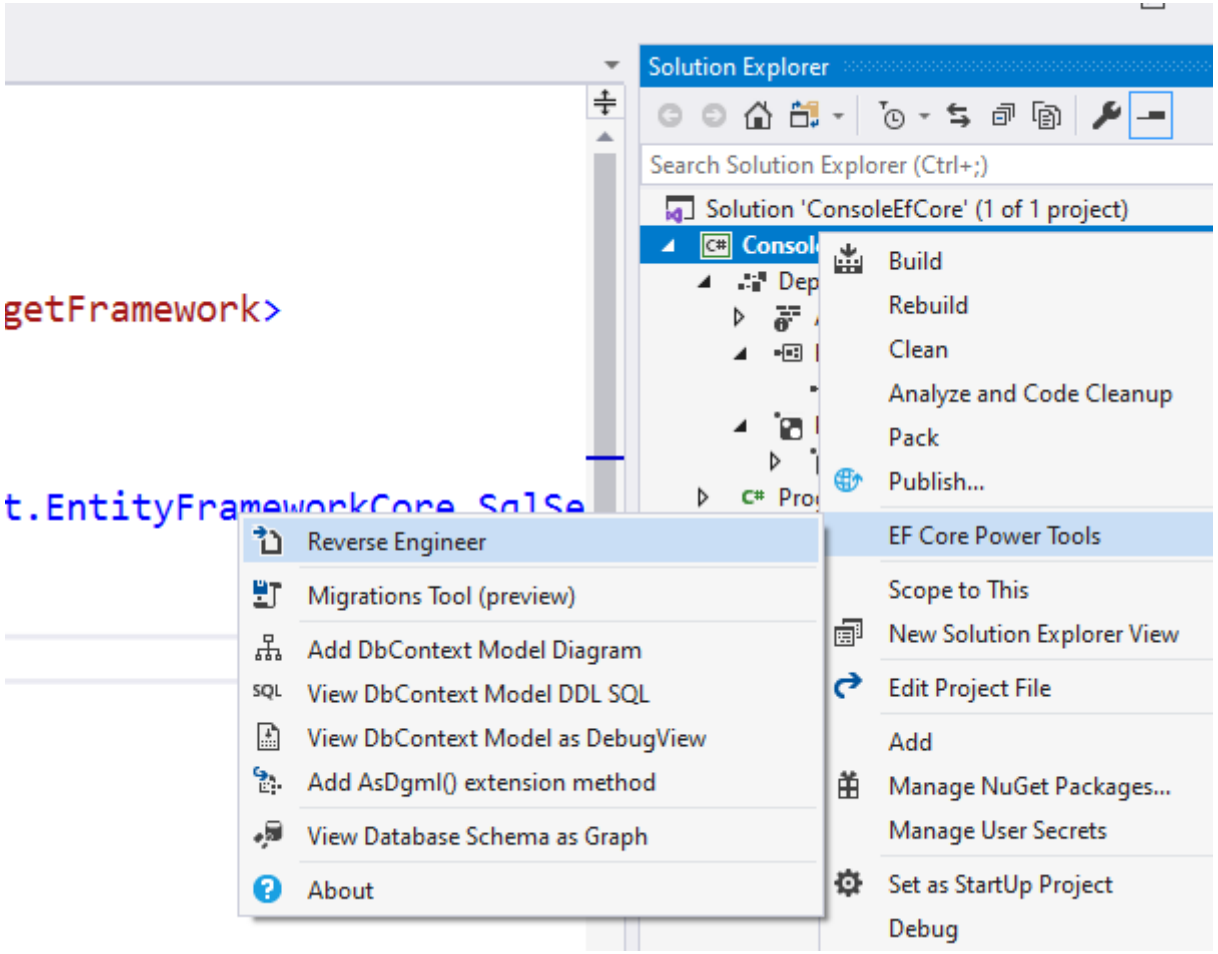


Generate Data Access Classes for Existing Database

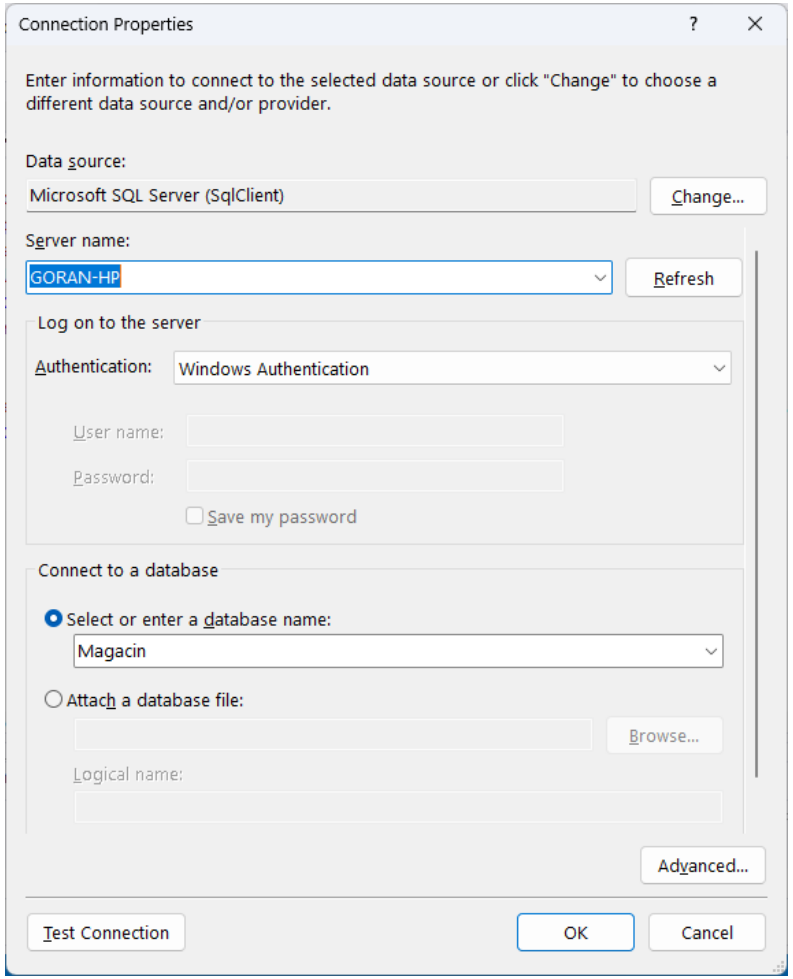


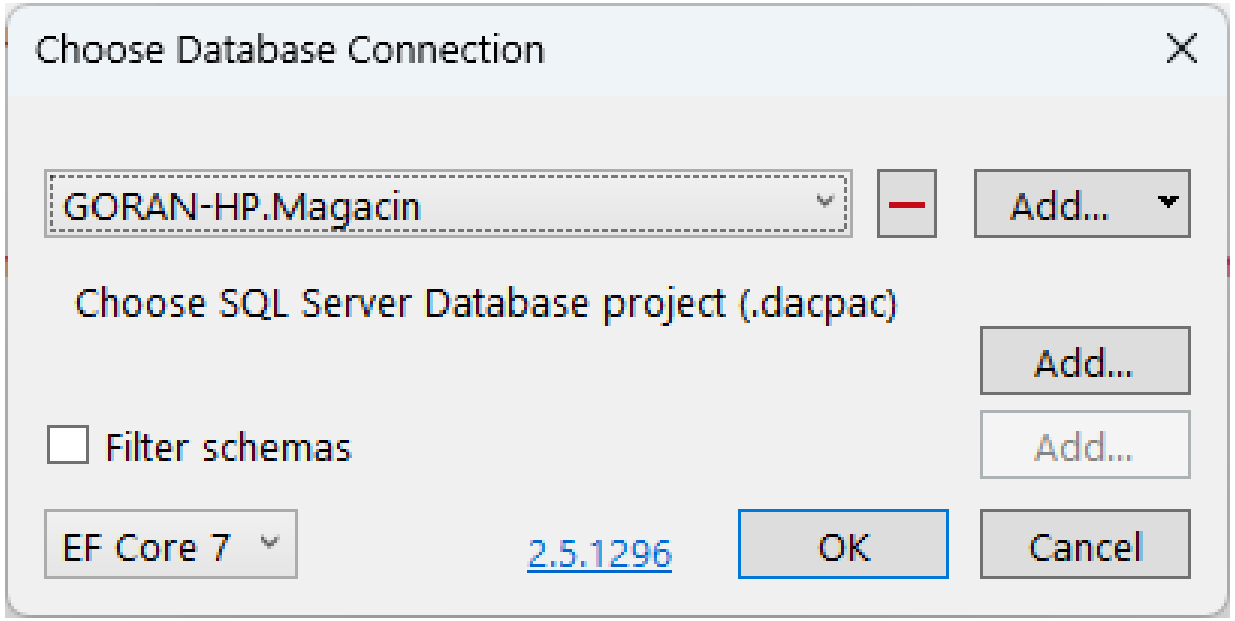
Create Database from the Domain Classes

Pokreni EF Core Power Tools

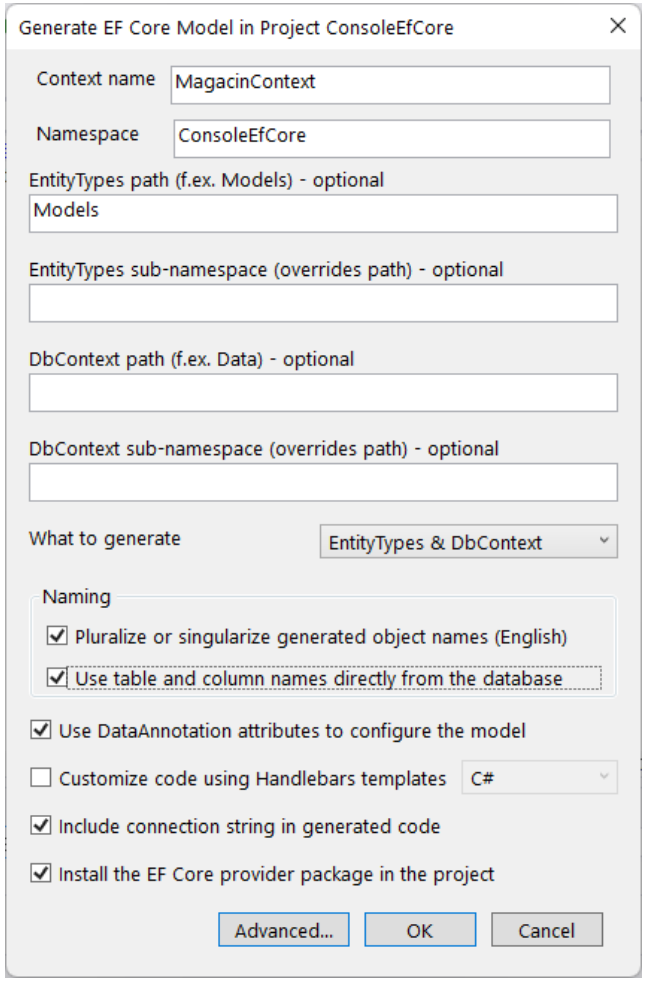


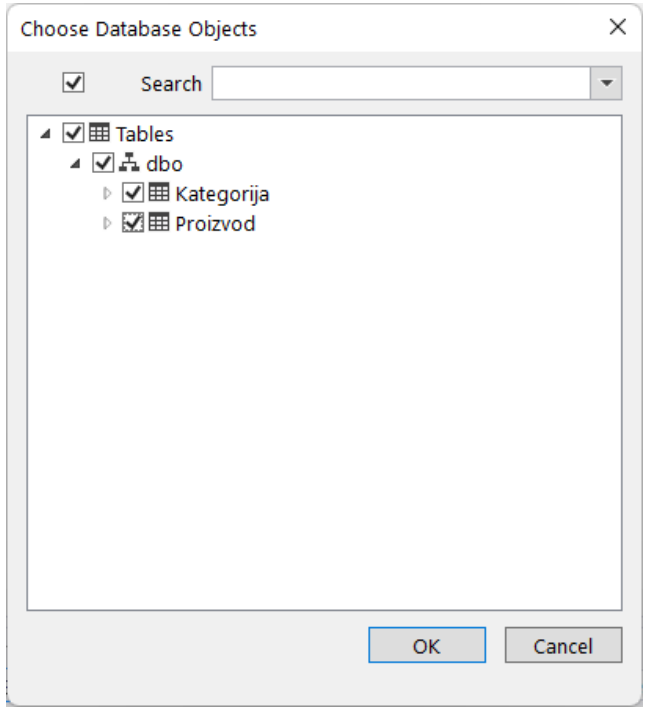
Odabir konekcije





Odabir tabela i kreiranje modela





Entitetska klasa Kategorija

```
namespace ConsoleEfCore.Models
{
    [Table("Kategorija")]
    public partial class Kategorija
    {
        public Kategorija()
        {
            Proizvodi = new HashSet<Proizvod>();
        }

        [Key]
        public int KategorijaId { get; set; }

        [Required]
        [StringLength(70)]
        public string Naziv { get; set; }

        [StringLength(120)]
        public string Opis { get; set; }


        public virtual ICollection<Proizvod> Proizvodi { get; set; }
    }
}
```

Entitetska klasa Proizvod

```
public partial class Proizvod
{
    [Key]
    public int ProizvodId { get; set; }

    public int KategorijaId { get; set; }

    [Required]
    [StringLength(120)]
    public string Naziv { get; set; }

    [Column(TypeName = "decimal(10, 2)")]
    public decimal Cena { get; set; }

    [StringLength(120)]
    public string Opis { get; set; }

    [ForeignKey(nameof(KategorijaId))]

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

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");


}
```

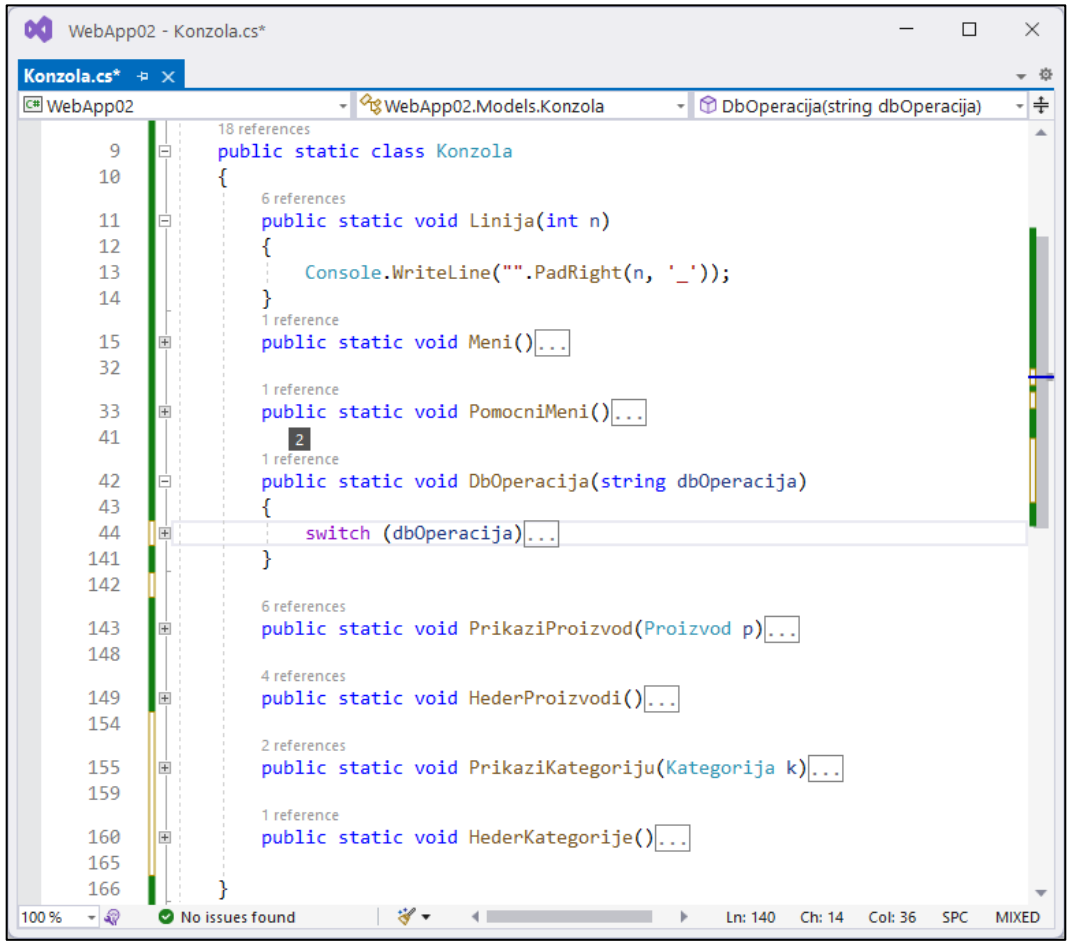
DbContext klasa

- DbContext klasa je most za interakciju 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 **DbSet<TEntity>** za sve entitetske klase koje su mapirane u tabele baze
- DbContext klasa:
  - upravlja konekcijom sa odgovarajućom bazom podataka
  - prati promene u objektima unutar DbSet ova
  - čuva promene iz DbSetova u bazi podataka

Klasa za komunikaciju sa bazom MagacinDal

```
using Microsoft.EntityFrameworkCore;

namespace WebApp02.Models{
    public static class MagacinDal
    {

    }
}
```

Klasa za komunikaciju sa konzolom - Kozola



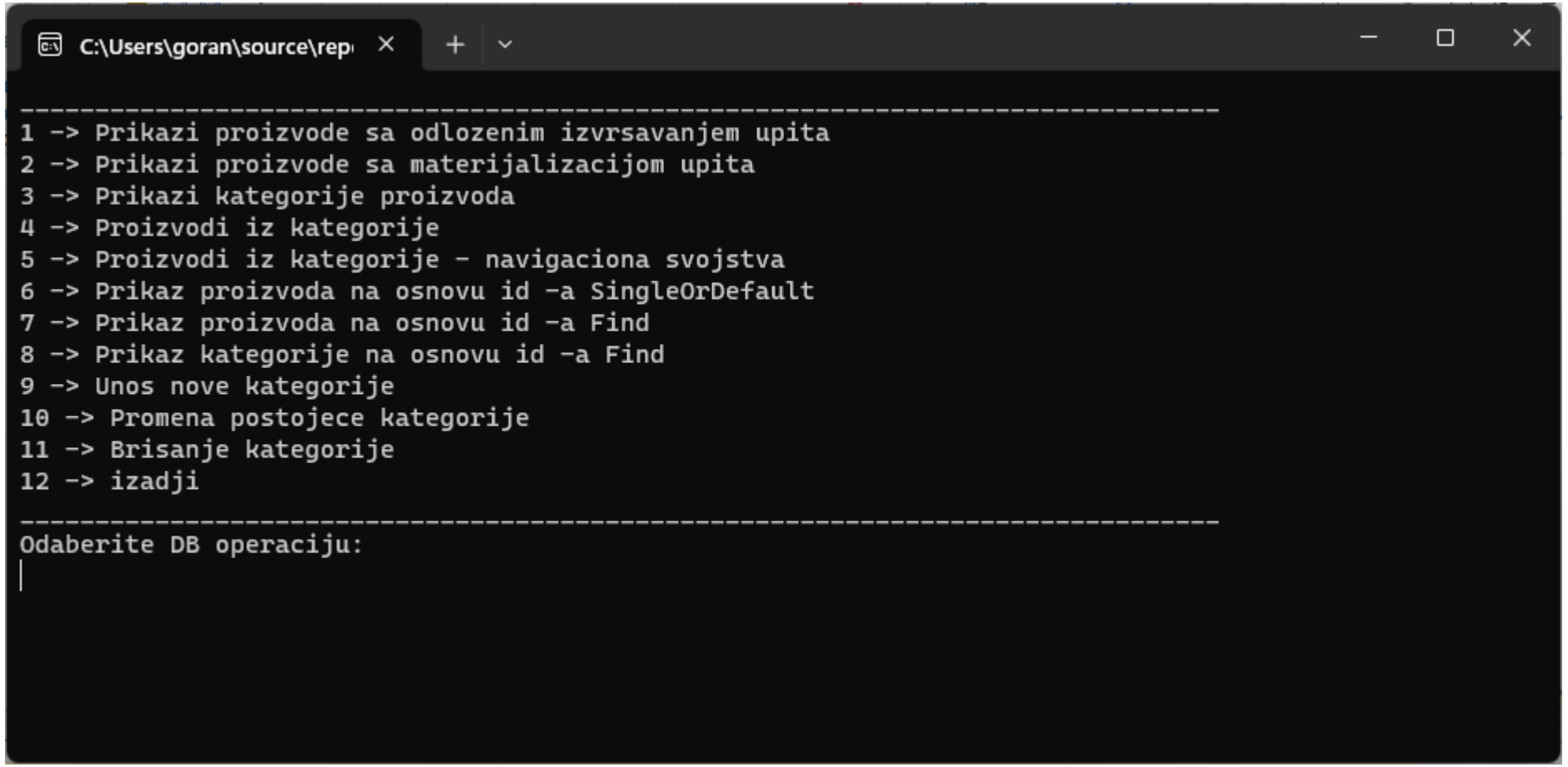
Klasa Konzola metoda DbOperacija

```
public static void DbOperacija(string dbOperacija)
{
    switch (dbOperacija)
    {
        case "1":
            //Prikazi proizvode sa odlozenim izvrsavanjem upita
            MagacinDal.PrikaziProizvode1();
            break;
        case "2":
            //Prikazi proizvode sa materijalizacijom upita
            MagacinDal.PrikaziProizvode2();
            break;
           ...
        default:
            //ne postoji Db operacija
            Console.WriteLine("Ne postojeca Db operacija");
            break;
    }
}
```

Fajl Program.cs

```
using WebApp02.Models;

string novaDbOperacija = "" ;
string dbOperacija;

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

Početni prozor aplikacije

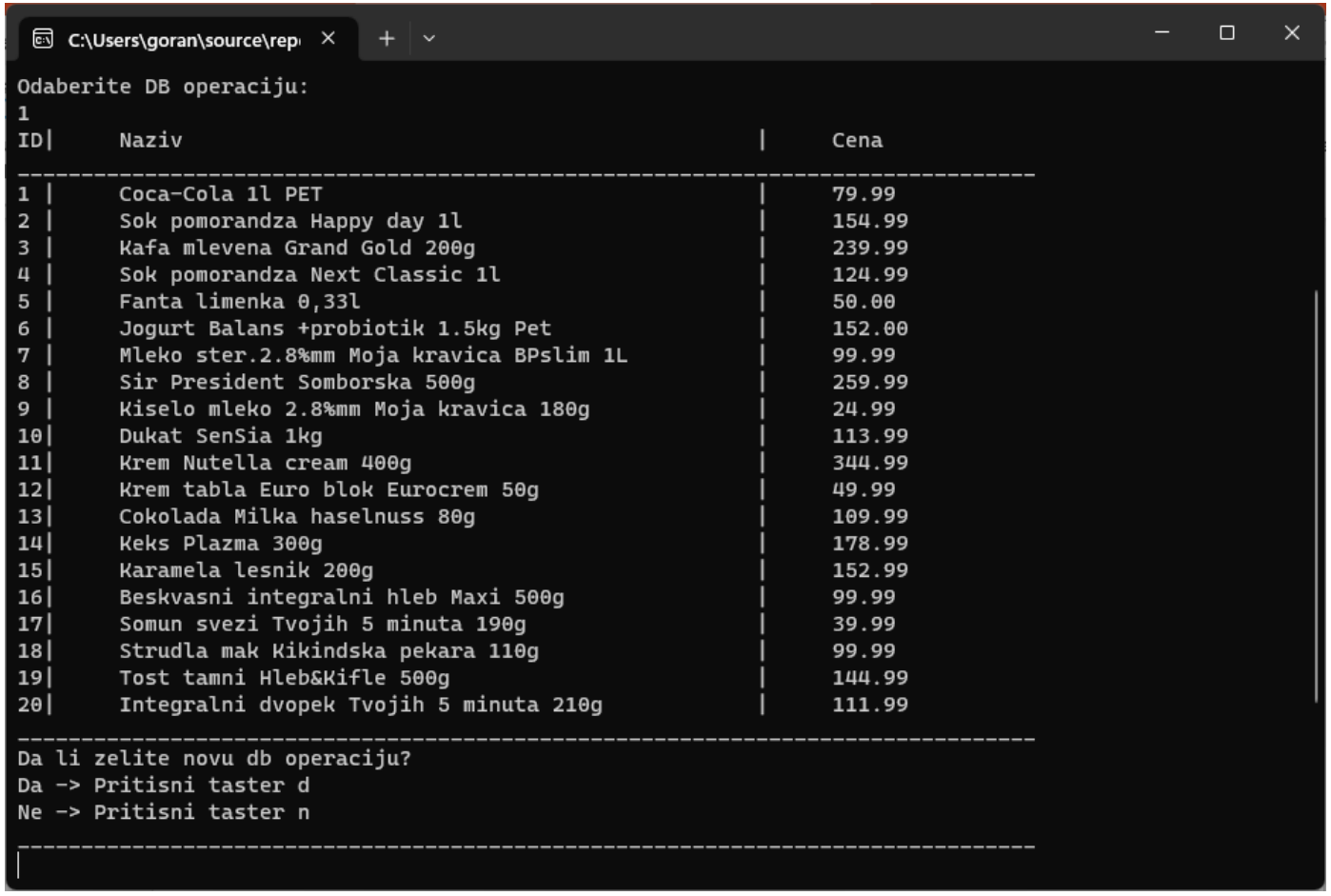


Čitanje podataka sa odloženim izvršavanjem upita

```
public static void PrikaziProizvode1()
{
    using (MagacinContext db = new MagacinContext())
    {

        DbSet<Proizvod> listaProizvoda = db.Proizvodi;

        Konzola.HederProizvodi();
        foreach (Proizvod p in listaProizvoda)
        {
            Konzola.PrikaziProizvod(p);
        }
    }
}
```

Metoda PrikaziProizvode1(), klasa MagacinDal

Prikaz proizvoda



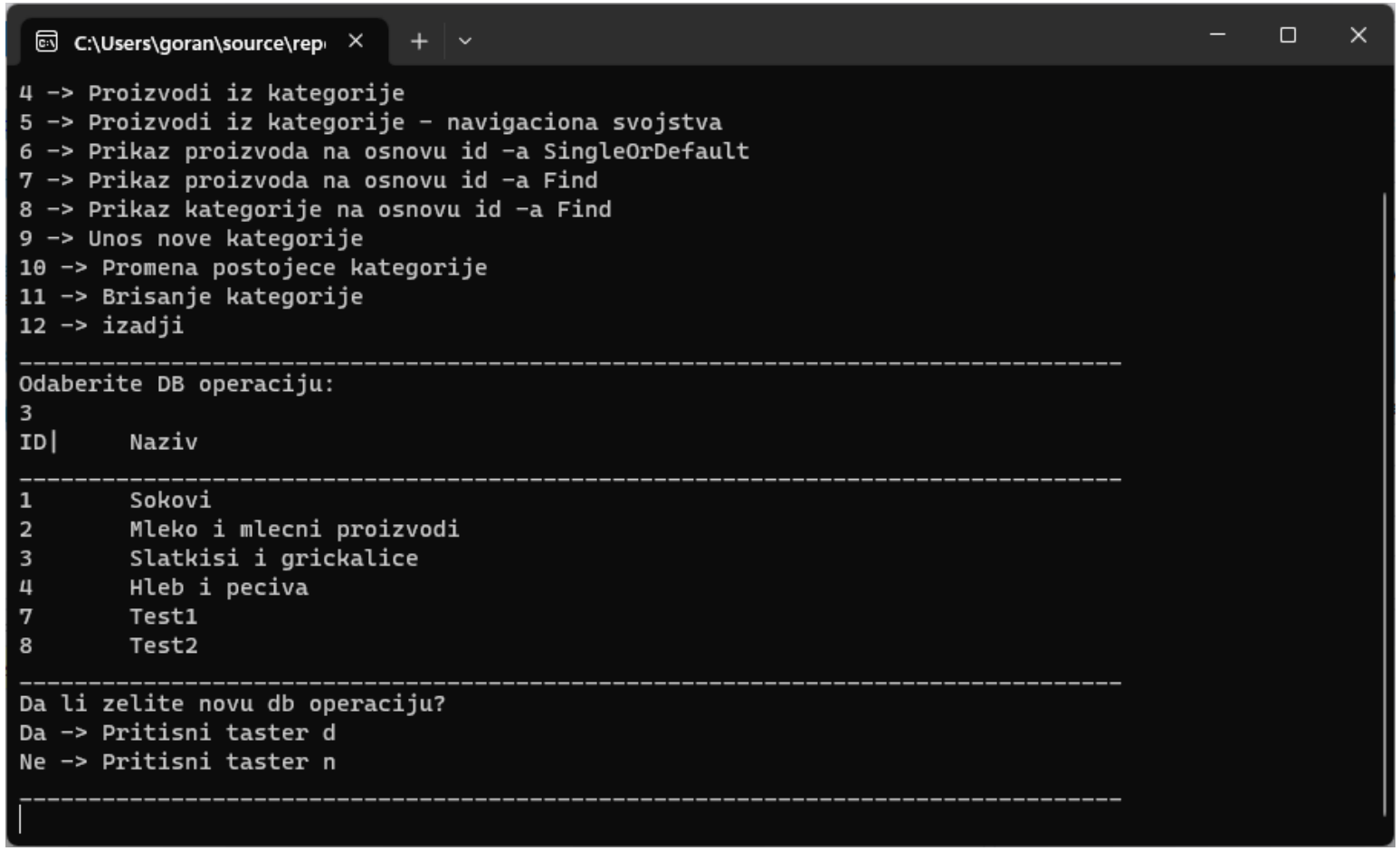
Čitanje podataka uz materijalizaciju upita

```
public static void PrikaziProizvode2()
{
    using (MagacinContext db = new MagacinContext())
    {

        List<Proizvod> listaProizvoda = db.Proizvodi.ToList();
        Konzola.HederProizvodi();
        foreach (Proizvod p in listaProizvoda)
        {
            Konzola.PrikaziProizvod(p);
        }
    }
}
```

Prikaz kategorija proizvoda

```
public static void PrikaziKategorije()
{
    using (MagacinContext db = new MagacinContext())
    {

        DbSet<Kategorija> listaKategorija = db.Kategorije;
        Konzola.HederKategorije();

        foreach (Kategorija k in listaKategorija)
        {
            Konzola.PrikaziKategoriju(k);
        }
    }
}
```

Prikaz kategorija



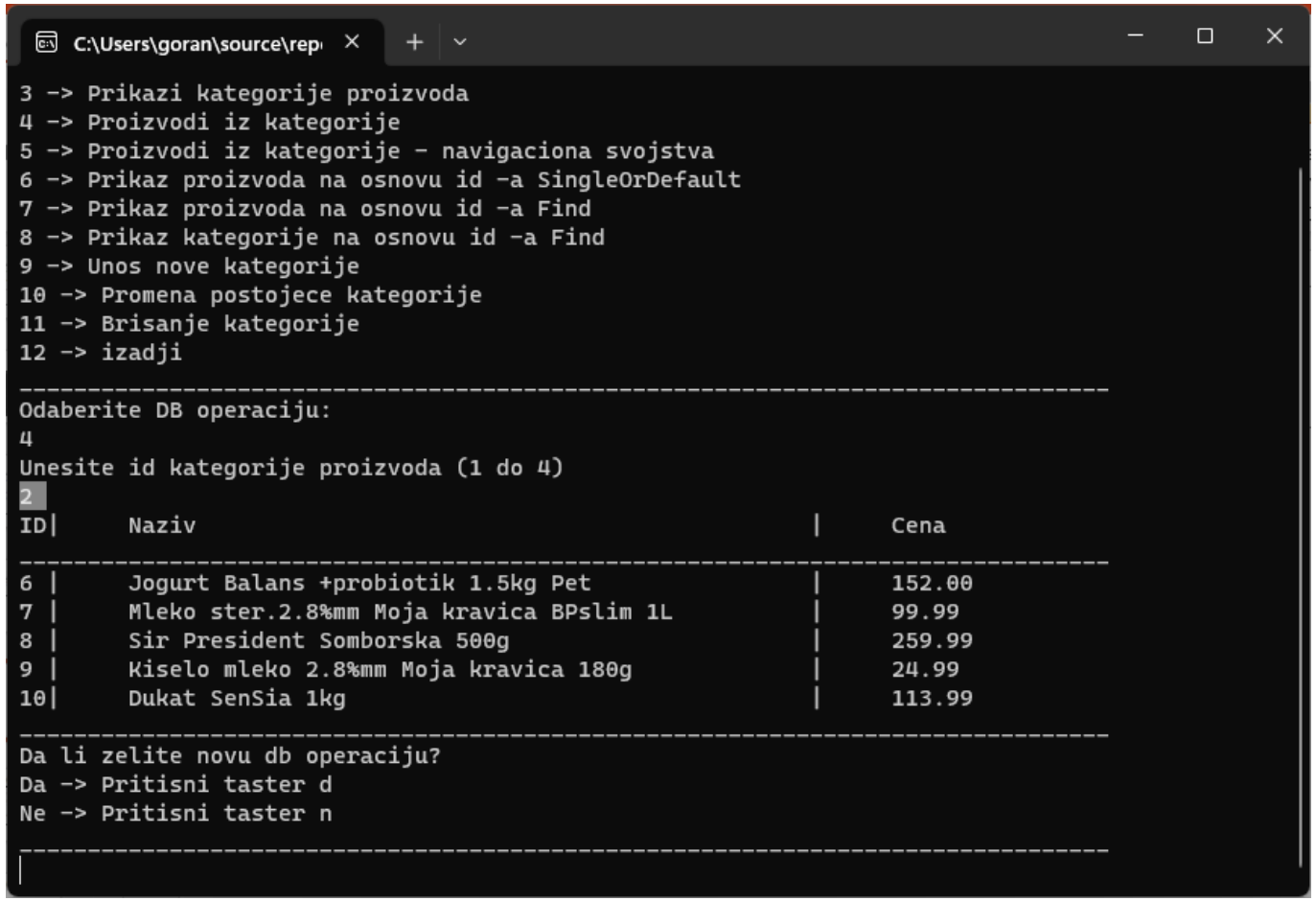
Filtriranje podataka

```
public static void ProizvodiIzKategorije1(int id)
{
    using (MagacinContext db = new MagacinContext())
    {

        IQueryable<Proizvod> listaProizvoda = db.Proizvodi
                    .Where(p => p.KategorijaId == id);
        Konzola.HederProizvodi();
        foreach (Proizvod p in listaProizvoda)
        {
            Konzola.PrikaziProizvod(p);
        }
    }
}
```

Filtriranje podataka

```
case "4":
    //Proizvodi iz kategorije
    Console.WriteLine("Unesite id kategorije proizvoda (1 do 4)");
    int id = int.Parse(Console.ReadLine());
    MagacinDal.ProizvodiIzKategorije1(id);
    break;

case "5":
    //Proizvodi iz kategorije - navigaciona svojstva
    Console.WriteLine("Unesite id kategorije proizvoda (1 do 4)");
    id = int.Parse(Console.ReadLine());
    MagacinDal.ProizvodiIzKategorije2(id);
    break;
```

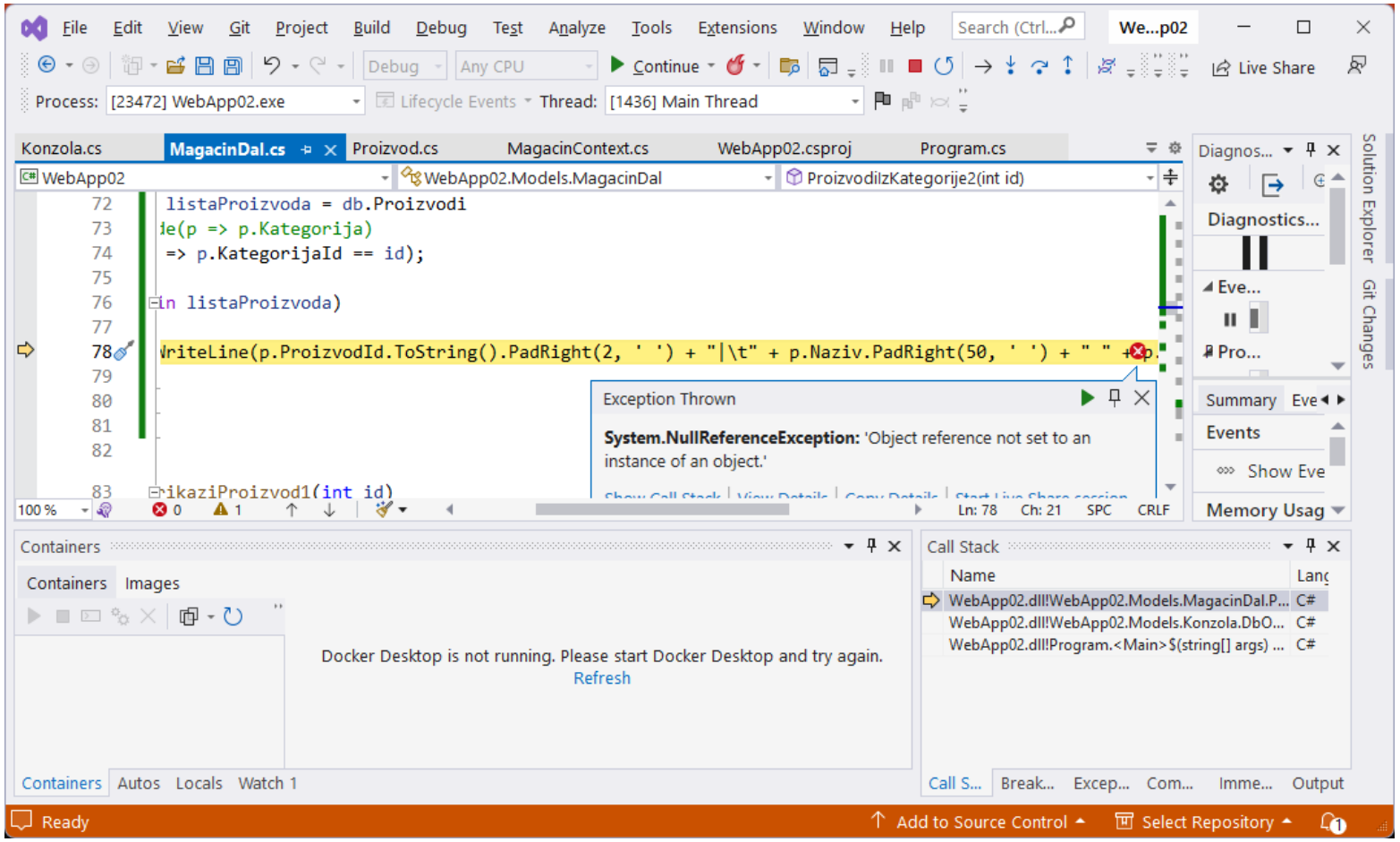
Prikaz proizvoda iz kategorije



Upotreba navigacionog svojstva

```
public static void ProizvodiIzKategorije2(int id)
{
    using (MagacinContext db = new MagacinContext())
    {

        IQueryable<Proizvod> listaProizvoda = db.Proizvodi
        .Where(p => p.KategorijaId == id);

        foreach (Proizvod p in listaProizvoda)
        {
                    Console.WriteLine(p.ProizvodId.ToString().PadRight(2, ' ') +
"|\t" + p.Naziv.PadRight(50, ' ') + " " + p.Kategorija.Naziv);
        }
    }
}
```

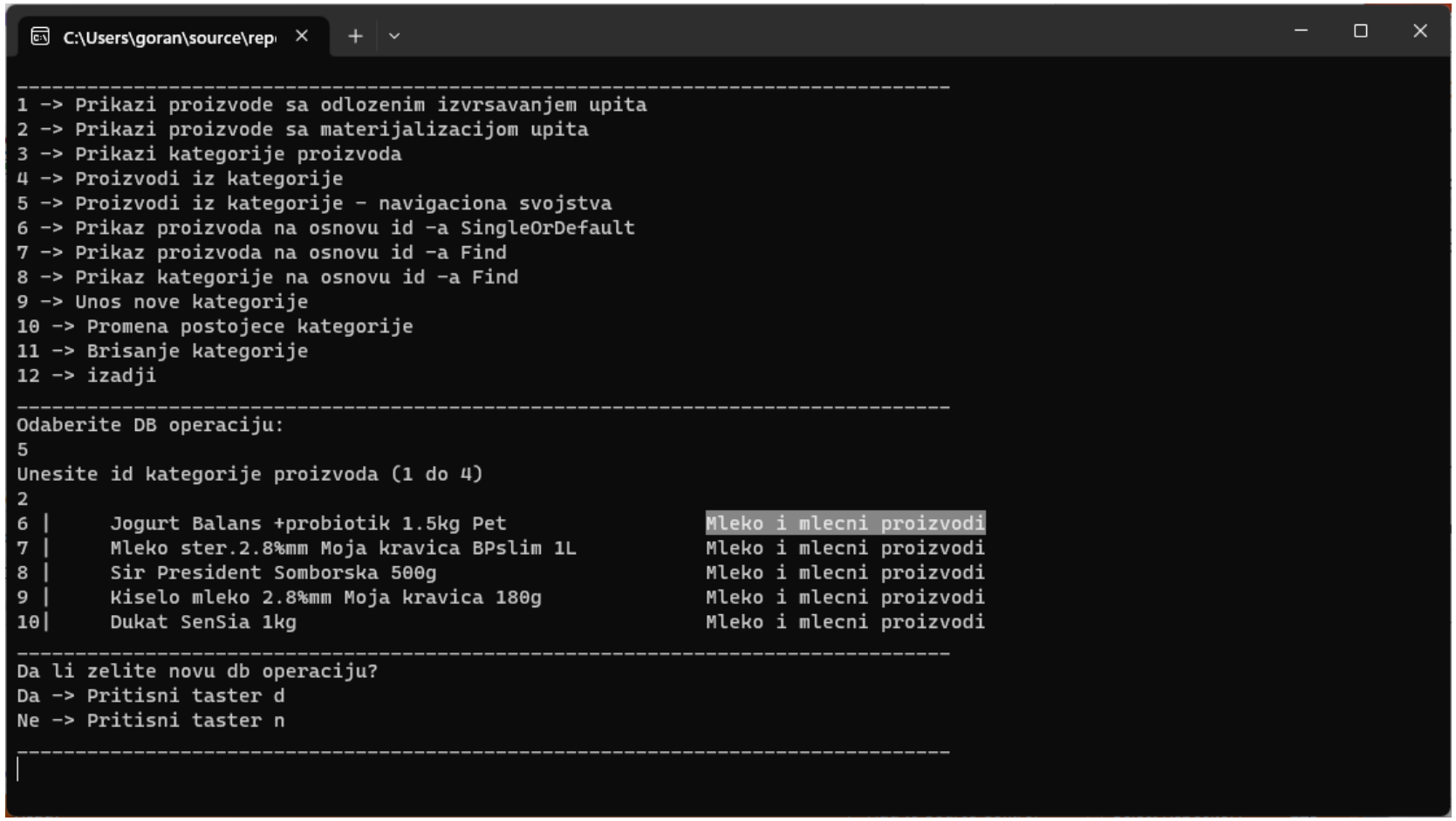
Upotreba navigacionog svojstva



Prikaz podataka iz povezane tabele – Include()

```
public static void ProizvodiIzKategorije2(int id)
{
    using (MagacinContext db = new MagacinContext())
    {

        IQueryable<Proizvod> listaProizvoda = db.Proizvodi
                    .Include(p => p.Kategorija)
                    .Where(p => p.KategorijaId == id);

        foreach (Proizvod p in listaProizvoda)
        {
                    Console.WriteLine(p.ProizvodId.ToString().PadRight(2, ' ') +
"|\t" + p.Naziv.PadRight(50, ' ') + " " + p.Kategorija.Naziv);
        }
    }
}
```

Prikaz podataka iz povezane tabele – Include()



Čitanje jednog reda

```
public static void PrikaziProizvod1(int id)
{
    using (MagacinContext db = new MagacinContext())
    {
        Proizvod p1 = db.Proizvodi
        .SingleOrDefault(p => p.ProizvodId == id);
        if (p1 != null)
        {
            Konzola.PrikaziProizvod(p1);
        }
        else
        {
            Console.WriteLine($"Ne postoji proizvod ciji je id = {id}");
        }
    }
}
```

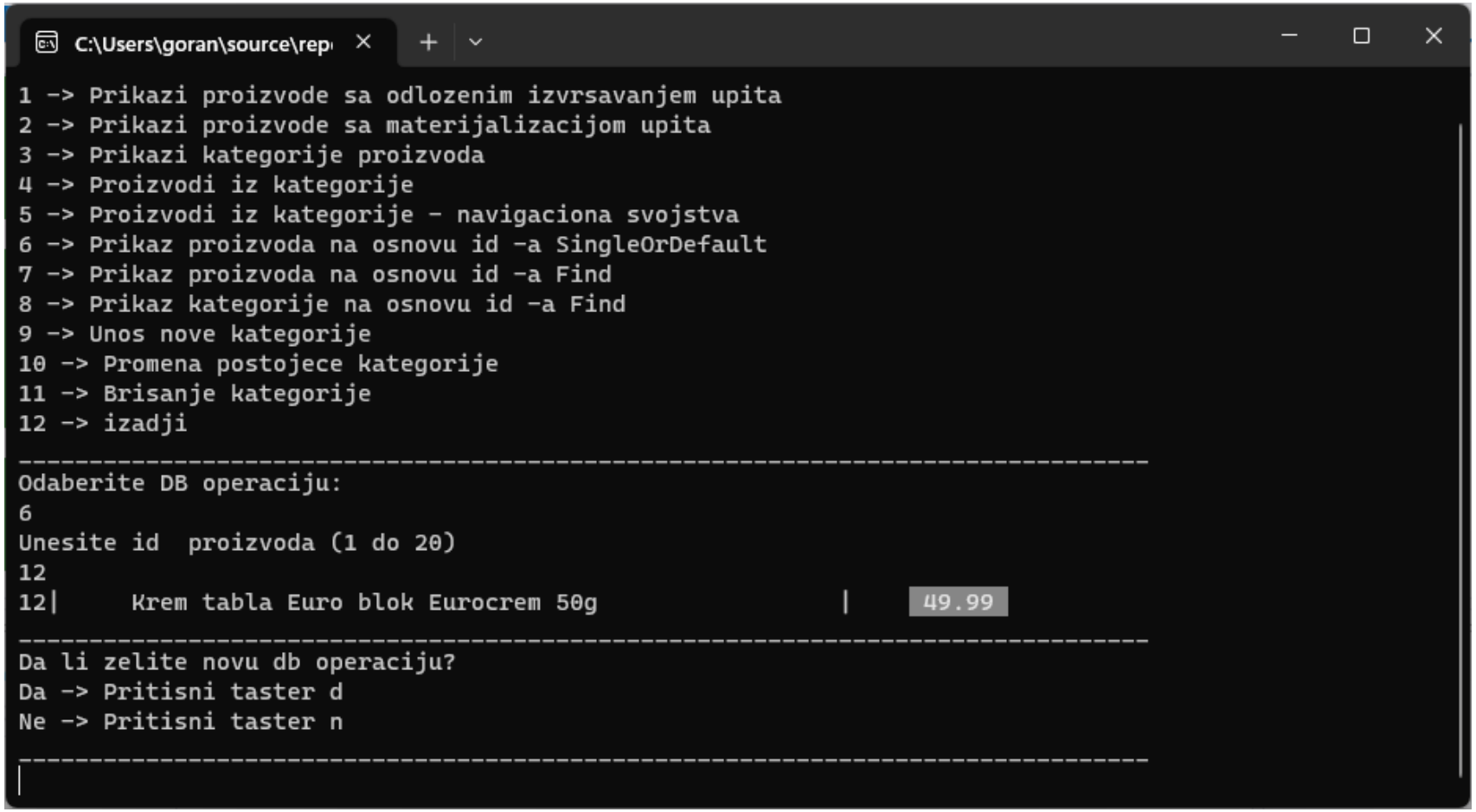
Čitanje jednog reda

```
public static void PrikaziProizvod2(int id)
{
    using (MagacinContext db = new MagacinContext())
    {

        Proizvod p1 = db.Proizvodi.Find(id);

        if (p1 != null)
        {
            Konzola.PrikaziProizvod(p1);
        }
        else
        {
            Console.WriteLine($"Ne postoji proizvod ciji je id = {id}");
        }
    }
}
```

Čitanje jednog reda

```
case "6":
    //Prikaz proizvoda na osnovu id - a SingleOrDefault
    Console.WriteLine("Unesite id  proizvoda (1 do 20)");
    id = int.Parse(Console.ReadLine());
    MagacinDal.PrikaziProizvod1(id);
    break;

case "7":
    //Prikaz proizvoda na osnovu id -a Find
    Console.WriteLine("Unesite id  proizvoda (1 do 20)");
    id = int.Parse(Console.ReadLine());
    MagacinDal.PrikaziProizvod2(id);
    break;
```

Čitanje jednog proizvoda



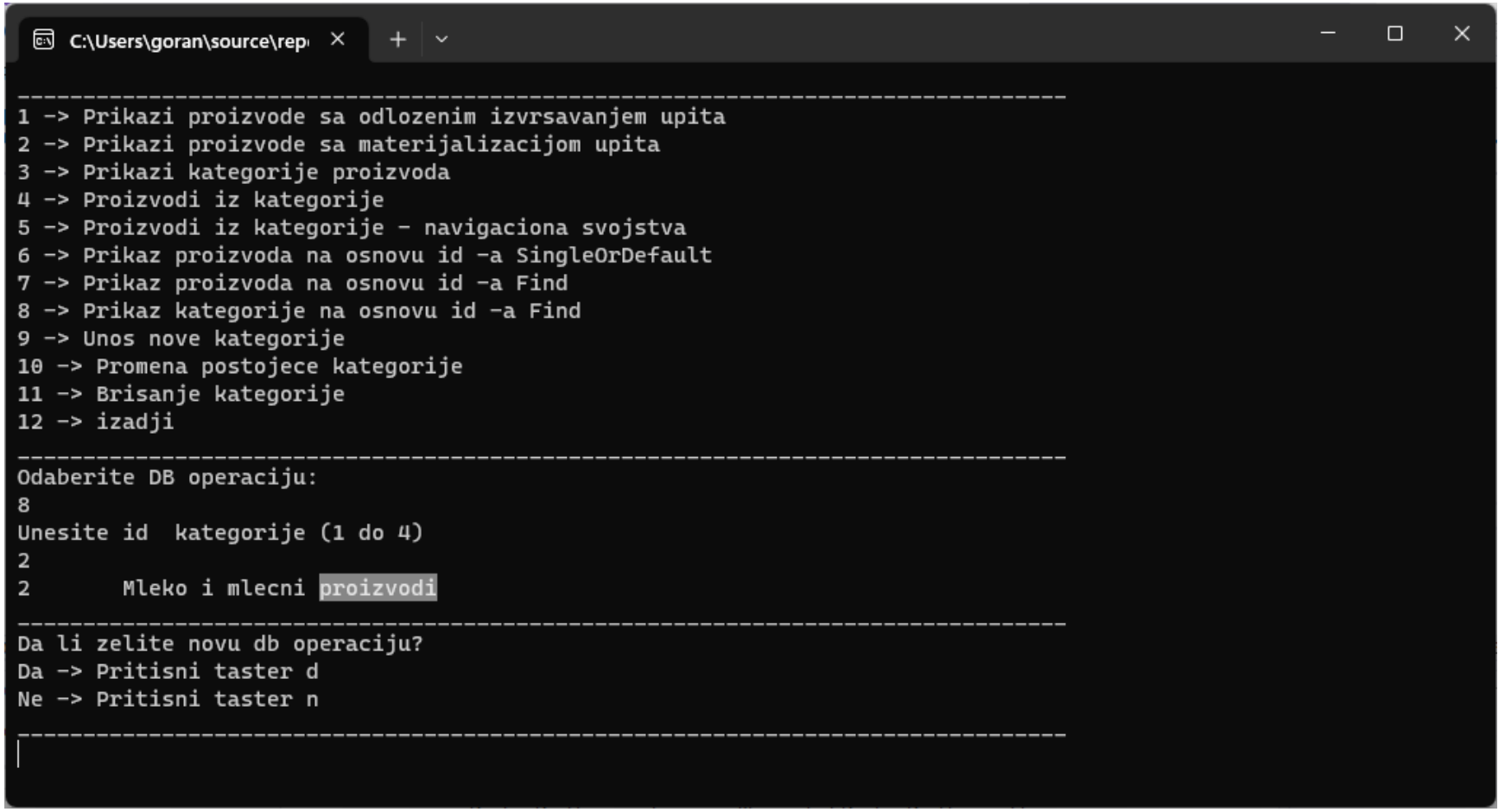
Čitanje jedne kategorije

```
public static void PrikaziKategoriju(int id)
{
    using (MagacinContext db = new MagacinContext())
    {
        Kategorija k1 = db.Kategorije.Find(id);

        if (k1 != null)
        {
            Konzola.PrikaziKategoriju(k1);
        }
        else
        {
            Console.WriteLine("Ne postoji kategorija");
        }
    }
}
```

Čitanje jedne kategorije

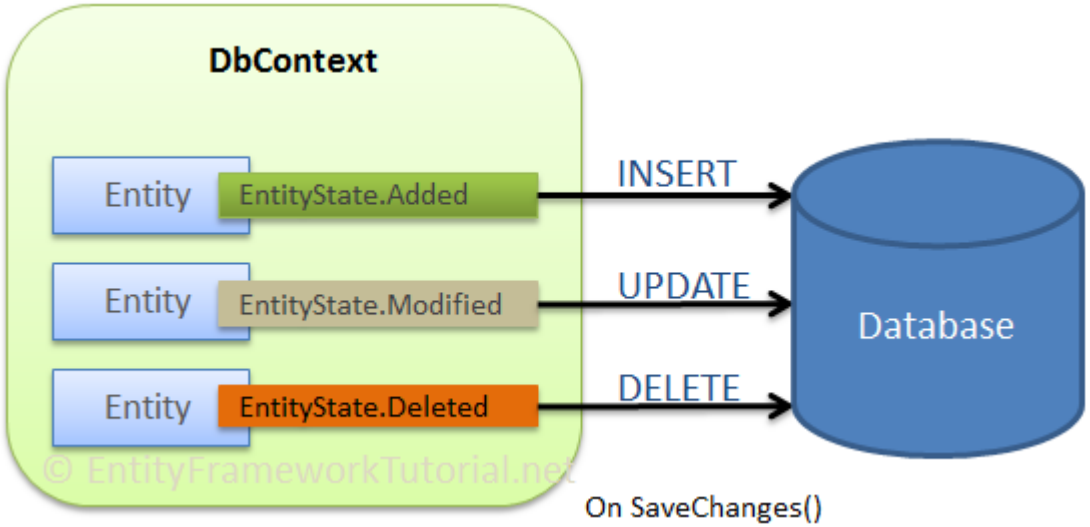
```
case "8":
    //Prikaz kategorije na osnovu id -a Find
    Console.WriteLine("Unesite id  kategorije (1 do 4)");
    id = int.Parse(Console.ReadLine());
    MagacinDal.PrikaziKategoriju(id);
    break;
```

Čitanje jedne kategorije

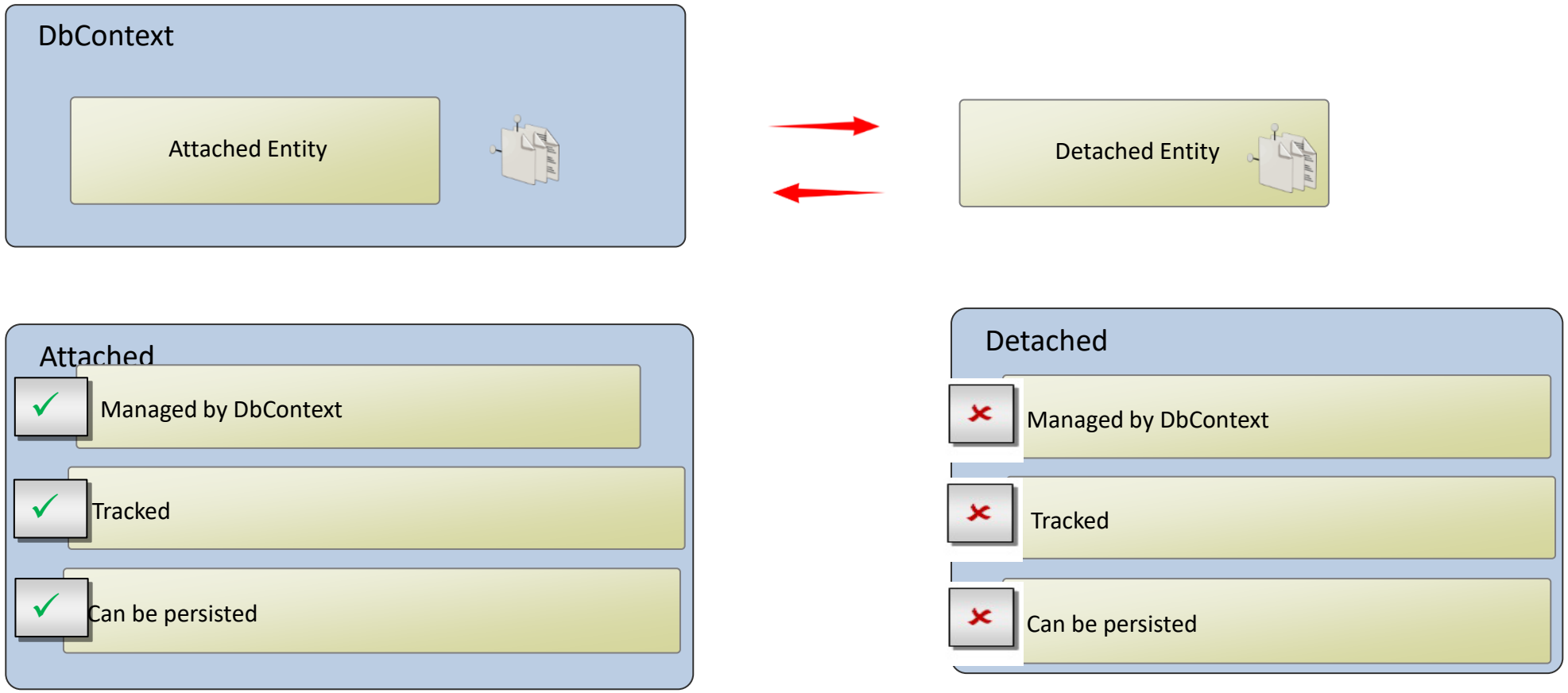


Ažuriranje podataka korišćenjem EF

- DbContext objekat sadrži entity objekte koji predstavljaju podatke u odgovarajućoj bazi podataka
- Ukoliko želimo da modifikujemo podatke u bazi prvo se vrši promena sadržaja DbContext objekta a zatim se od njega traži da sačuva promene u bazi podataka



Atačovani i detačovani objekti



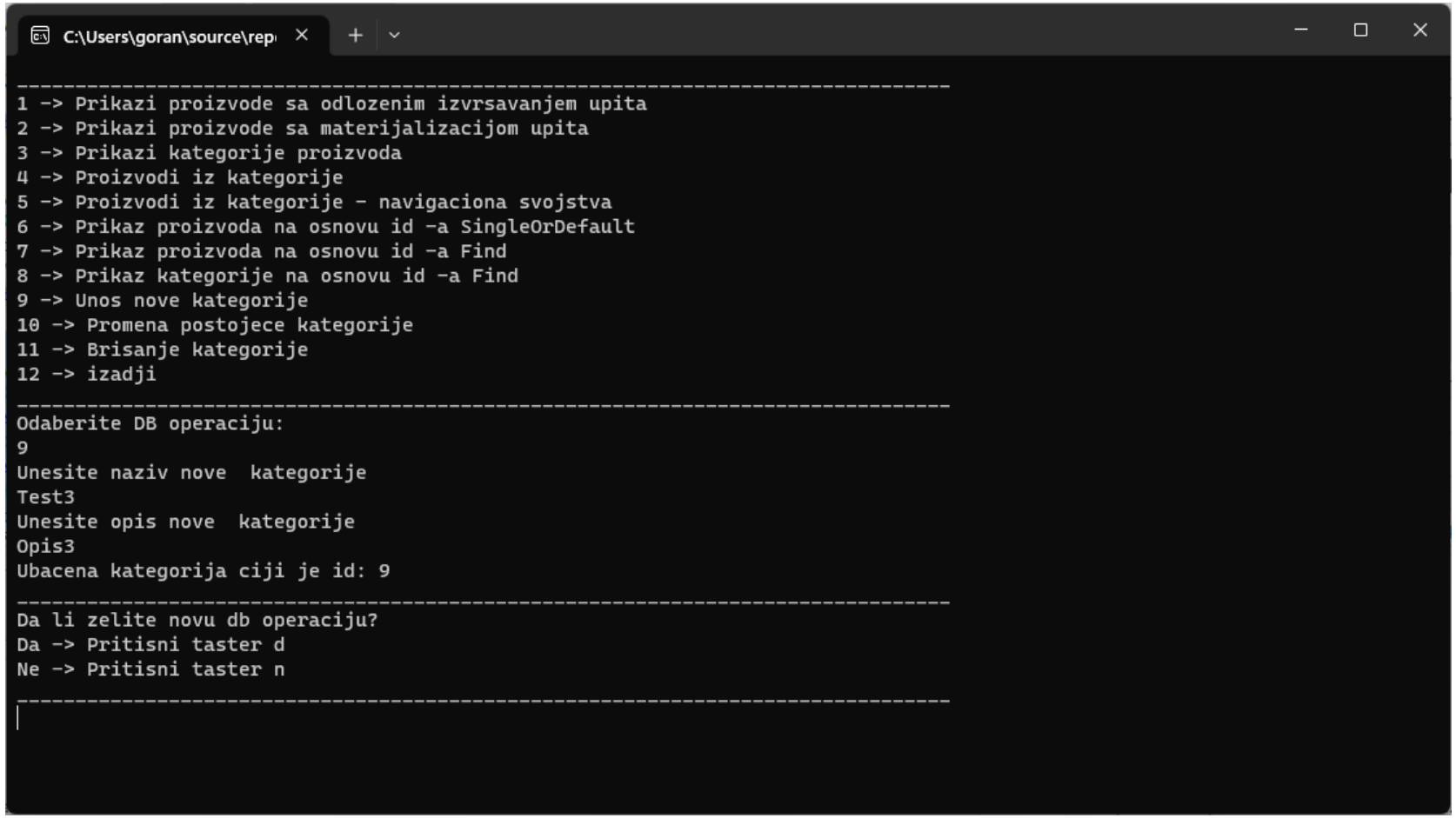
Unos podataka

```
public static int UbaciKategoriju(Kategorija k)
{
    using (MagacinContext db = new MagacinContext())
    {
        try
        {

            db.Add(k);
            db.SaveChanges();
            return k.KategorijaId;
        }
        catch (Exception xcp)
        {
            Console.WriteLine(xcp.Message);
            return -1;
        }

    }
}
```

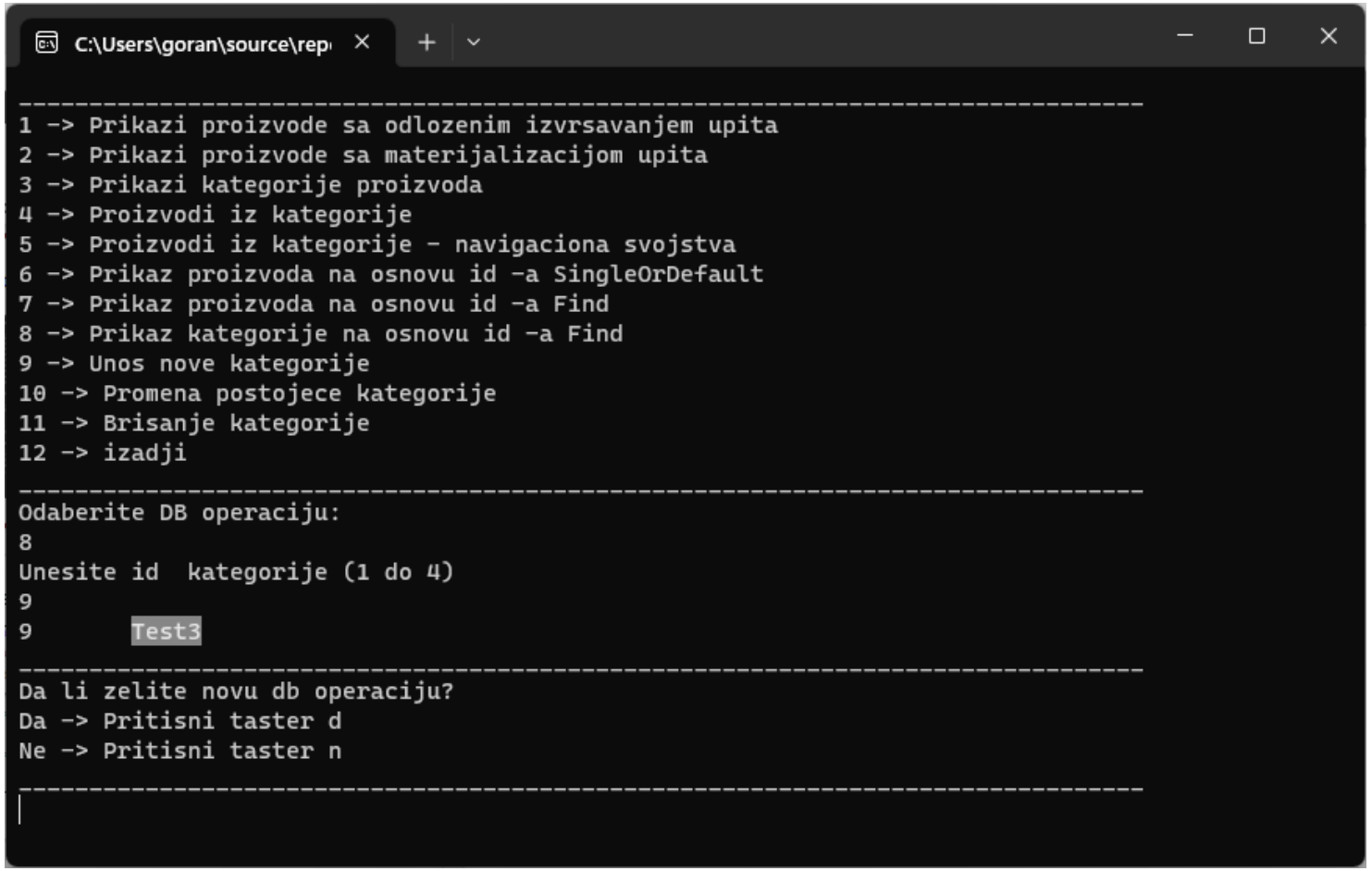
Unos nove kategorije

```
case "9":
    //Unos nove kategorije
    Console.WriteLine("Unesite naziv nove  kategorije");
    string naziv = Console.ReadLine();
    Console.WriteLine("Unesite opis nove  kategorije");
    string opis = Console.ReadLine();
    Kategorija k1 = new Kategorija { Naziv = naziv, Opis = opis };
    id = MagacinDal.UbaciKategoriju(k1);
    Console.WriteLine("Ubacena kategorija ciji je id: " + id);
    break;
```

Unos nove kategorije



Prikaz unetog reda



Metoda VratiKategoriju()

```
public static Kategorija? VratiKategoriju(int id)
{
    using (MagacinContext db = new MagacinContext())
    {
        Kategorija k1 = db.Kategorije.Find(id);
        return k1;
    }
}
```

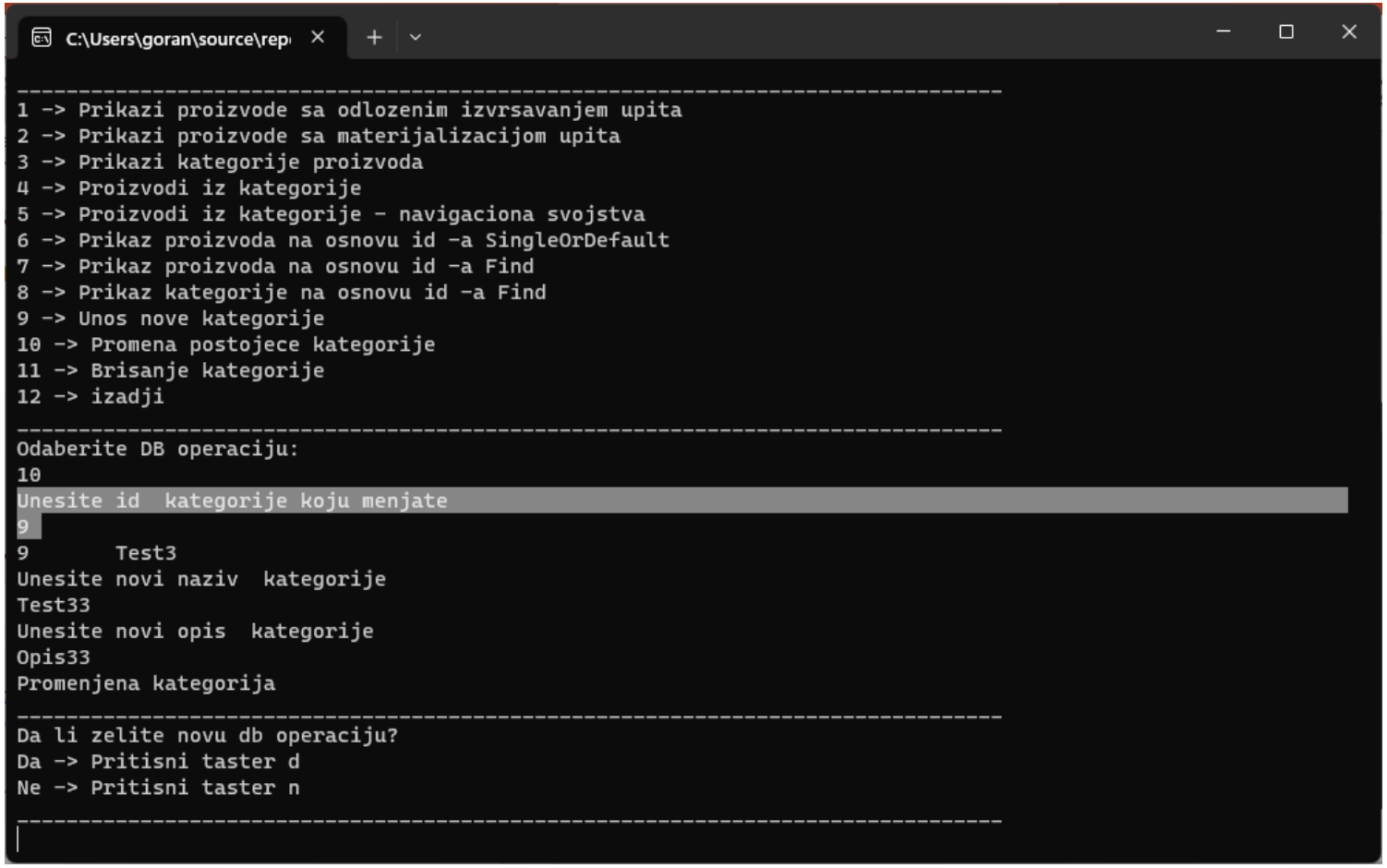
Ažuriranje reda

```
public static int PromeniKategoriju(Kategorija k)
{
    using (MagacinContext db = new MagacinContext())
    {
        try
        {
            db.Update(k);
            db.SaveChanges();
            return 0;
        }
        catch (Exception )
        {
            return -1;
        }

    }
}
```

Promena postojeće kategorije

```
case "10":
    //Promena postojece kategorije
    Console.WriteLine("Unesite id  kategorije koju menjate");
    id = int.Parse(Console.ReadLine());
    MagacinDal.PrikaziKategoriju(id);
    k1 = MagacinDal.VratiKategoriju(id);
    int r = 0;
    if (k1 != null)
    {
        Console.WriteLine("Unesite novi naziv  kategorije");
        naziv = Console.ReadLine();
        Console.WriteLine("Unesite novi opis  kategorije");
        opis = Console.ReadLine();
        k1.Naziv = naziv;
        k1.Opis = opis;
        r = MagacinDal.PromeniKategoriju(k1);
        if (r == 0)
        {
            Console.WriteLine("Promenjena kategorija");
        }
        else
        {
            Console.WriteLine("Greska pri promeni");
        }
    }

    break;
```

Promena postojeće kategorije



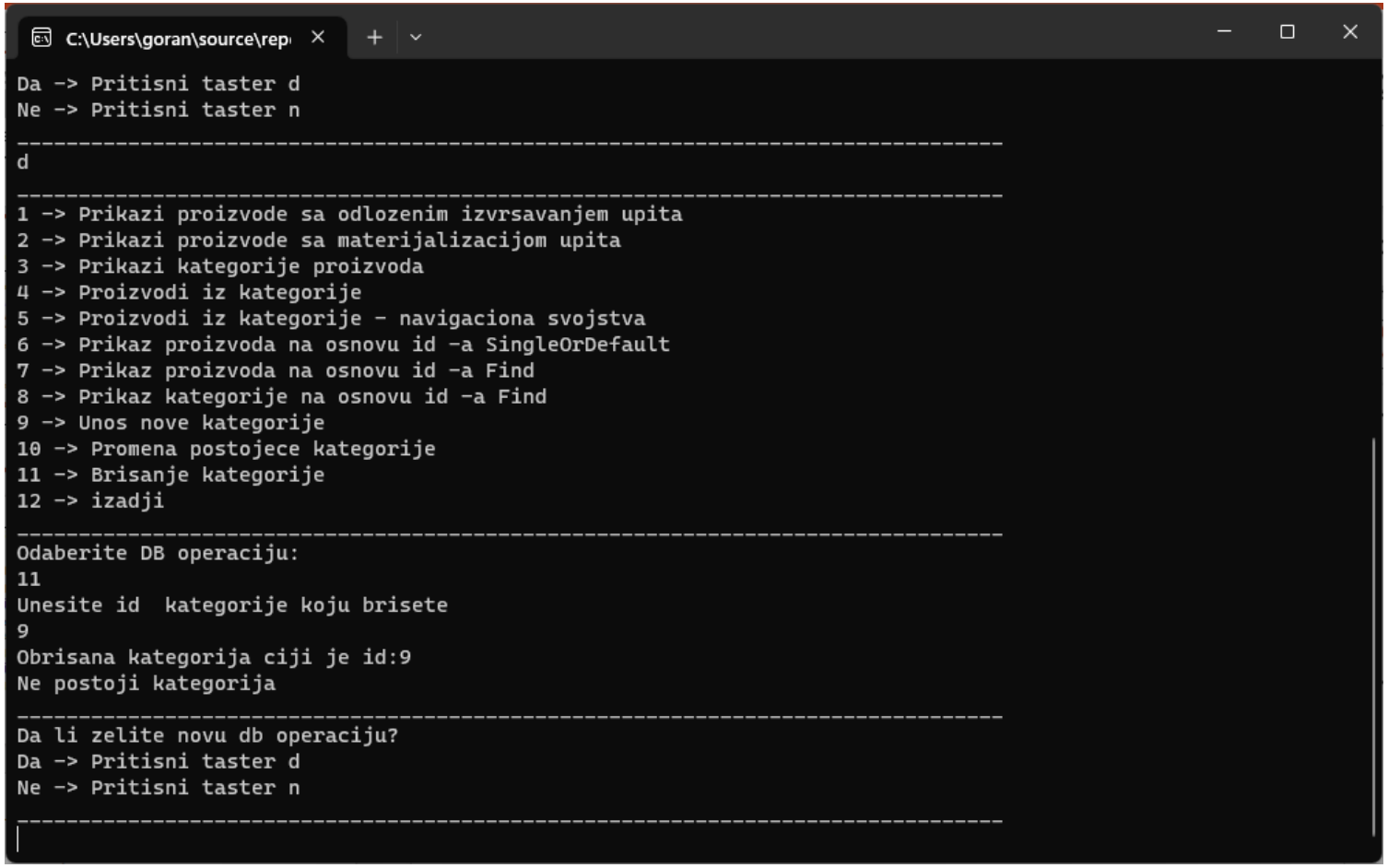
Brisanje reda

```
public static int ObrisiKategoriju(int id)
{
    using (MagacinContext db = new MagacinContext())
    {
        try
        {
            Kategorija k = db.Kategorije.Find(id);
            db.Kategorije.Remove(k);
            db.SaveChanges();
            return 0;
        }
        catch (Exception)
        {

            return -1;
        }
    }
}
```

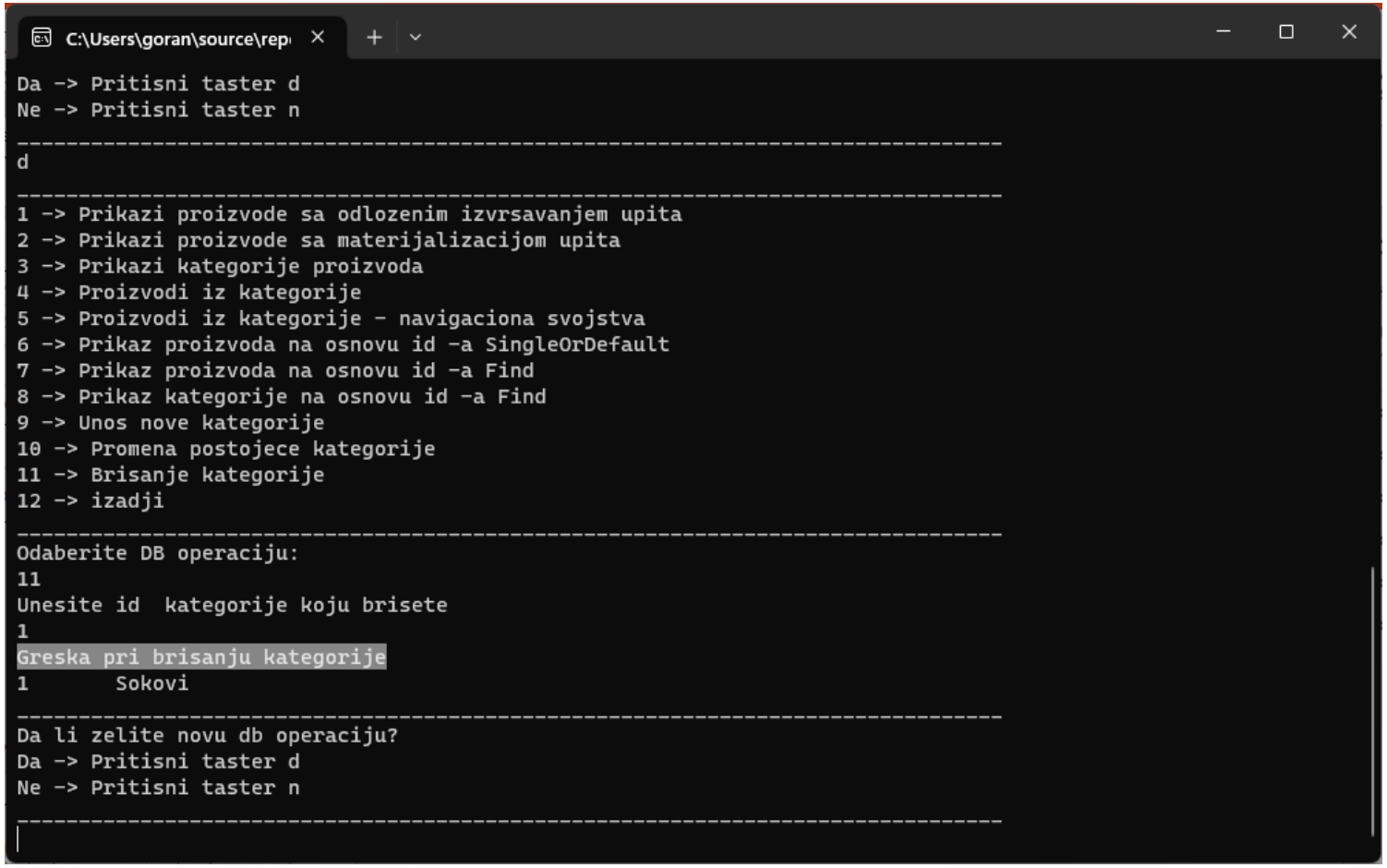
Brisanje reda

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

Brisanje reda



Neuspešno brisanje reda



Klase Task i Task<T>

- Klasa Task se nalazi u biblioteci **System.Threading.Tasks**
- Klasa Task se koristi za izvršavanje više poslova istovremeno, svaki u posebnoj programskoj niti
- Obično se vremenski zahtevni zadaci izvršavaju u posebnoj programskoj niti
- Klasa **Task** predstavlja metodu koja ne vraća vrednost, izvršava se u posebnoj programskoj niti i to izvršavanje je asinhrono
- Klasa **Task<T>** predstavlja asinhronu operaciju koja vraća rezultat

Vraćanje vrednosti iz zadatka

- Koristi se Task<TResult> klasa
- Koristi se Result svojstvo

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

    // UI je blokitan
    label1.Text = t1.Result;
}
```

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

- Modifikator async označava da je metoda asinhrona
- Unutar asinhrone metode se koristi operator await koji suspenduje izvršavanje metode sve dok se zadatak ne završi
- Omogućavaju poziv asinhrone metode 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 sinhrona metoda vraća **void** asinhroni ekvivalent vraća **Task**
- Ako sinhrona metoda vraća tip **T**, asinhroni ekvivalent vraća **Task<T>**

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;
}
```

```
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 operacije koja se izvršava u posebnoj programskoj niti i koja ne vraća vrednost koristi se klasa:

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

Odogovor: b

Pitanje 5

Za asinhronu metodu koja treba da vrati string kao povratni tip se specificira:

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

Odogovor: c

Pitanje 6

Kreirana je metoda klasa sledećeg potpisa:
private async Task<string> Citaj()
Na koji način pozivamo ovu metodu:

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

Odogovor: c