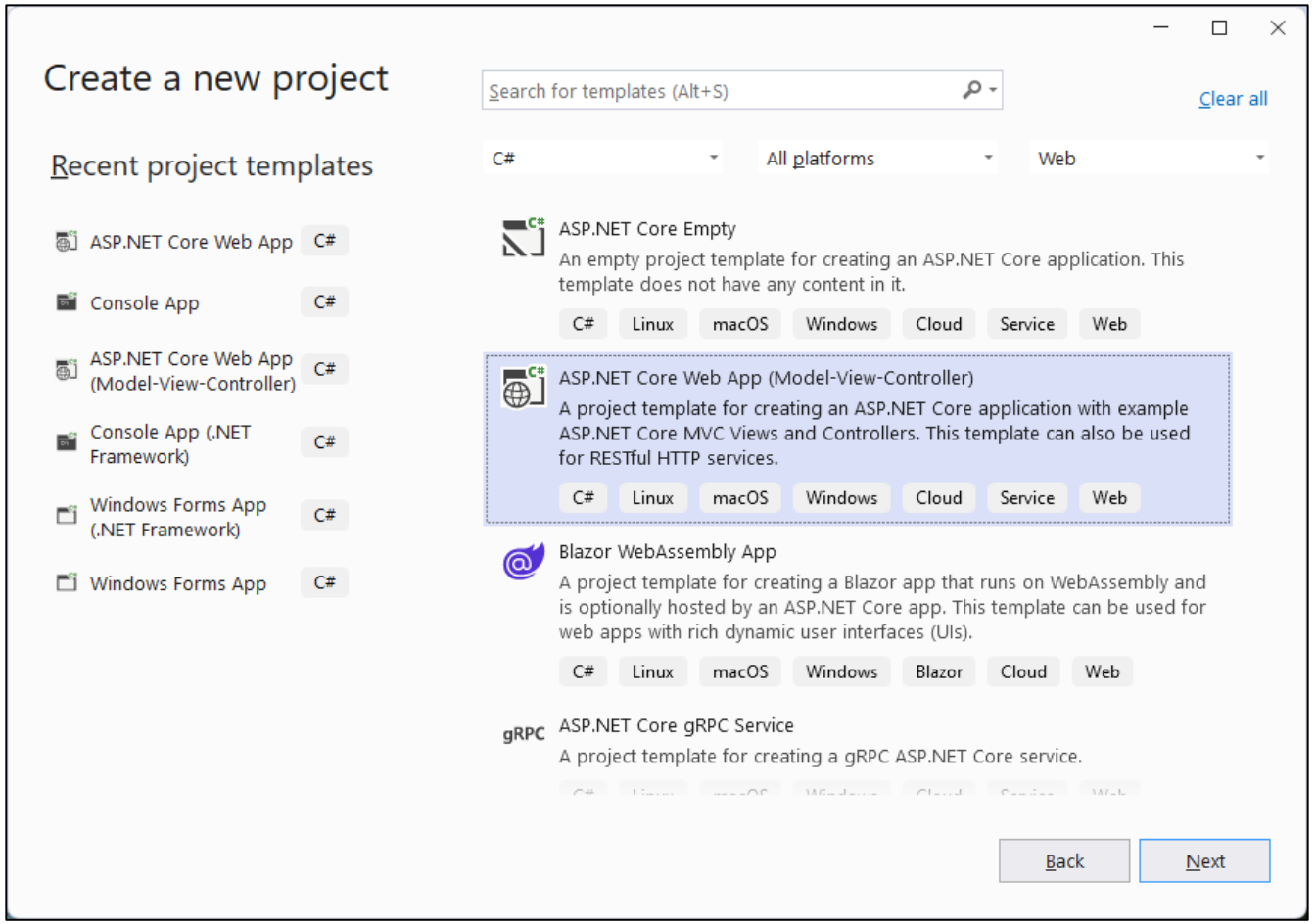
MVC modeli

MVC Model

- Model predstavlja domen aplikacije
- Svrha modela je da da smisao podacima
- Model se može koristiti kao kontejner za podatke koji se prikazuju na web stranici
- Kao model podataka najčešće se koristi entitetski model

Kreiranje ASP.NET Core web aplikacije



Model klasa(folder Models)

```
using System.ComponentModel.DataAnnotations.Schema;
```

```
[Table("Osoba")]
public class Osoba
{
    public int OsobaId { get; set; }
    public string Ime { get; set; }
    public string Prezime { get; set; }
    public string Adresa { get; set; }
}
```

- Model se definiše kao klasa u folderu Models
- Svojstva klase odgovaraju kolonama u tabeli
- Anotacija **[Table("Osoba")]** određuje naziv tabele u bazi podataka
- Anotacione klase nalaze se u biblioteci **System.ComponentModel.DataAnnotations.Schema**

Interfejs ISkladiste

```
namespace WebApp03.Models
{
    public interface ISkladiste
    {
        IEnumerable<Osoba> Osobe { get; }
    }
}
```

- Interfejs **Iskladiste** definiše pristup podacima o osobama
- Svojstvo Osobe vraća kolekciju objekata tipa **Osoba**
- Kolekcija je tipa **IEnumerable**, što omogućava iteraciju kroz podatke

Models folder

```
public class OsobaSkladiste : Iskladiste
{
    public IEnumerable<Osoba> Osobe
    {
        get
        {
            List<Osoba> listaOsoba = new List<Osoba>()
            {
            new Osoba { OsobaId = 1, Ime = "Pera", Prezime = "Peric", Adresa = "Prvomajska 4" },
            new Osoba { OsobaId = 2, Ime = "Mika", Prezime = "Mikic", Adresa = "Glavna 12" },
            new Osoba { OsobaId = 3, Ime = "Laza", Prezime = "Lazic", Adresa = "Marije Bursac 2" }
            };
            return listaOsoba;
        }
    }
}
```

- Klasa OsobaSkladiste implementira interfejs Iskladiste
- Vraća kolekciju objekata Osoba
- Podaci su u ovom primeru definisani u memoriji (List<Osoba>)

Dodavanje servisa u kontejner

```
using WebApp03.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddTransient<Iskladiste, OsobaSkladiste>();

var app = builder.Build();
```

- Interfejs Iskladiste je registrovan u **DI kontejneru**
- Kao implementacija je registrovana klasa **OsobaSkladiste**
- Servis je tipa transient, pa se nova instanca klaseservisa kreira pri svakom zahtevu

Modifikovana klasa HomeController

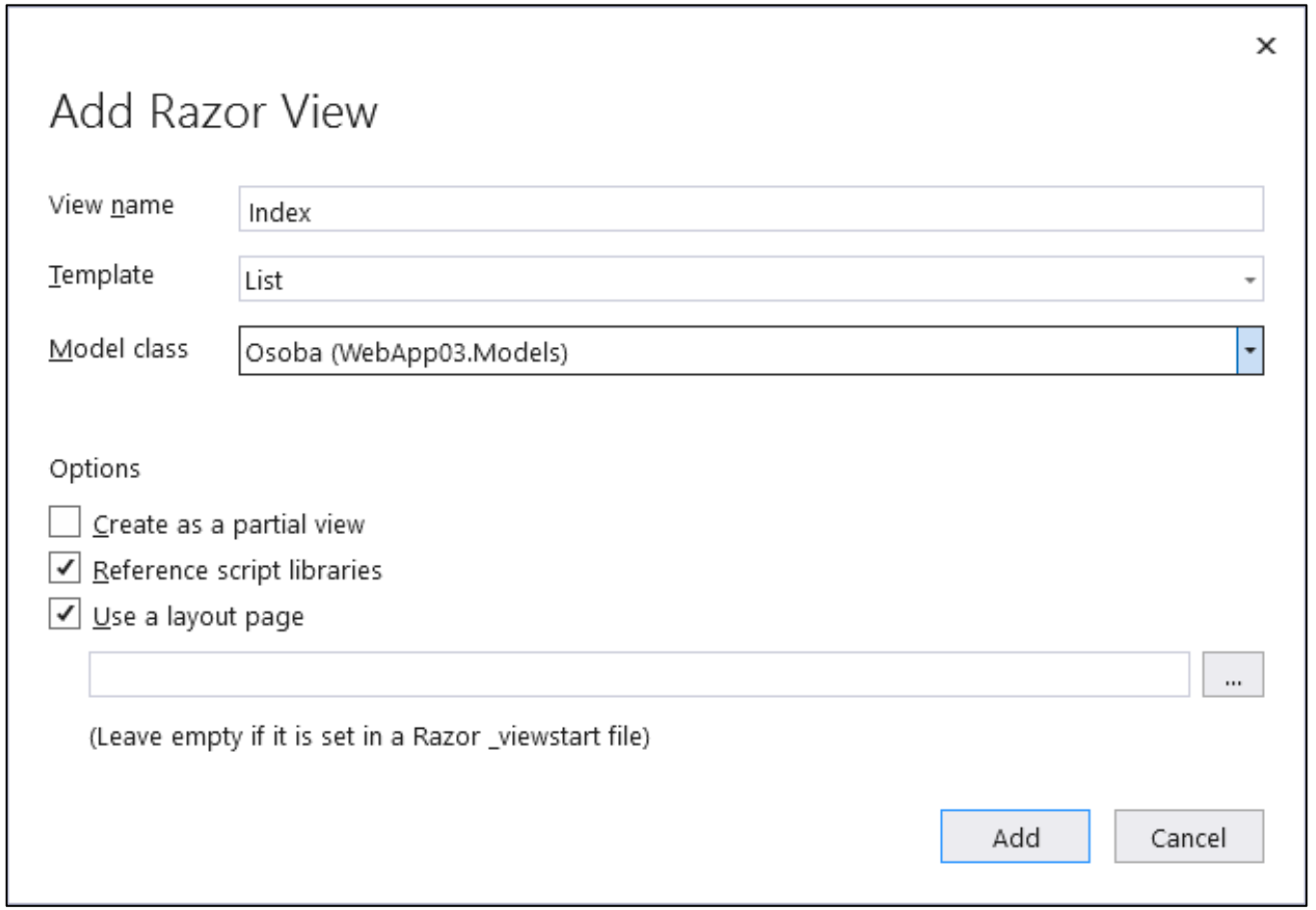
```
public class HomeController : Controller
{
    private readonly ISkladiste _skladiste;

    public HomeController(ISkladiste skladiste)
    {
        _skladiste = skladiste;
    }


    public IActionResult Index()
    {
        ViewBag.Poruka = "Lista osoba";

        return View(_skladiste.Osobe);
    }
}
```

- Kontroler ne kreira skladište podataka sam
- Implementacija interfejsa **ISkladiste** dobija se putem **Dependency Injection** kontejnera
- Metoda **Index** prosleđuje kolekciju objekata **Osoba** pogledu

Ubacivanje zavisnosti u kontroler

- MVC framework zahteva instancu kontrolera **HomeController**
- DI kontejner kreira instancu kontrolera i obezbeđuje njegove zavisnosti
- Za interfejs Iskladiste koristi se klasa **OsobaSkladiste**
- Instanca klase OsobaSkladiste prosleđuje se konstruktoru kontrolera

Kreiranje Index pogleda bazirano na modelu



Direktiva @model

```
@model IEnumerable<WebApp03.Models.Osoba>

@{
    ViewData["Title"] = "Index";
}
```

- Direktiva @**model** određuje tip modela koji koristi pogled
- Podaci koje kontroler prosleđuje pogledu dostupni su kroz svojstvo **Model**
- U ovom slučaju model je kolekcija objekata Osoba

Index pogled -1

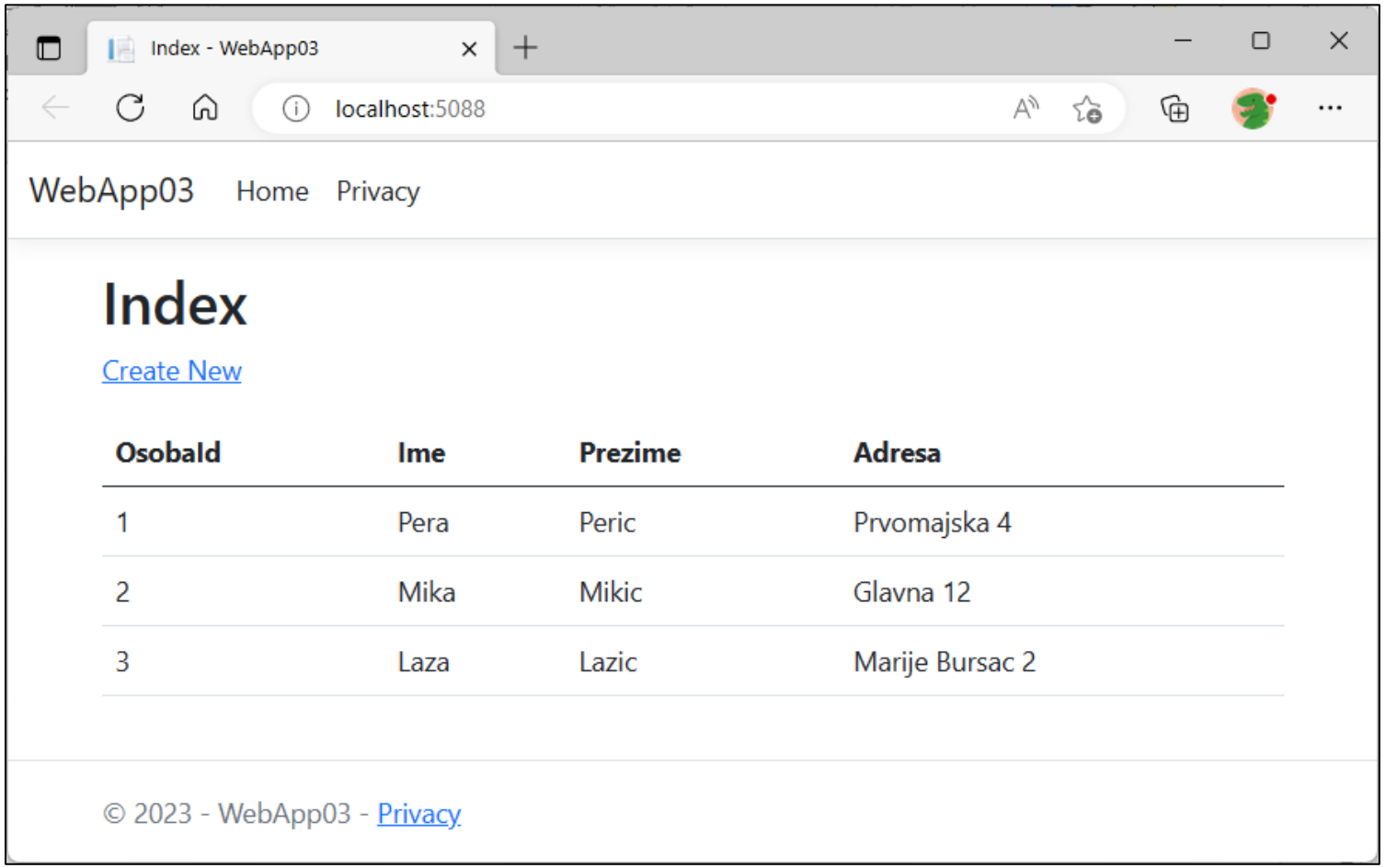
```
<h2>Index</h2>


<table class="table table-bordered table-striped">
    <thead>
        <tr>
            <th>
                Id
            </th>
            <th>
               Ime
            </th>
            <th>
                Prezime
            </th>
            <th>
                Adresa
            </th>

        </tr>
    </thead>
```

Index pogled -2

```
<tbody>
    @foreach (Osoba os in Model) {
        <tr>
            <td>
                @os.OsobaId
            </td>
            <td>
                @os.Ime
            </td>
            <td>
                @os.Prezime
            </td>
            <td>
                @os.Adresa
            </td>

        </tr>
}
    </tbody>
</table>
```

- Petlja @foreach iterira kroz kolekciju **Model**
- Svaki element kolekcije je objekat klase Osoba
- Svojstva objekta (OsobaId, Ime, Prezime, Adresa) prikazuju se u tabeli

Prikaz Index pogleda



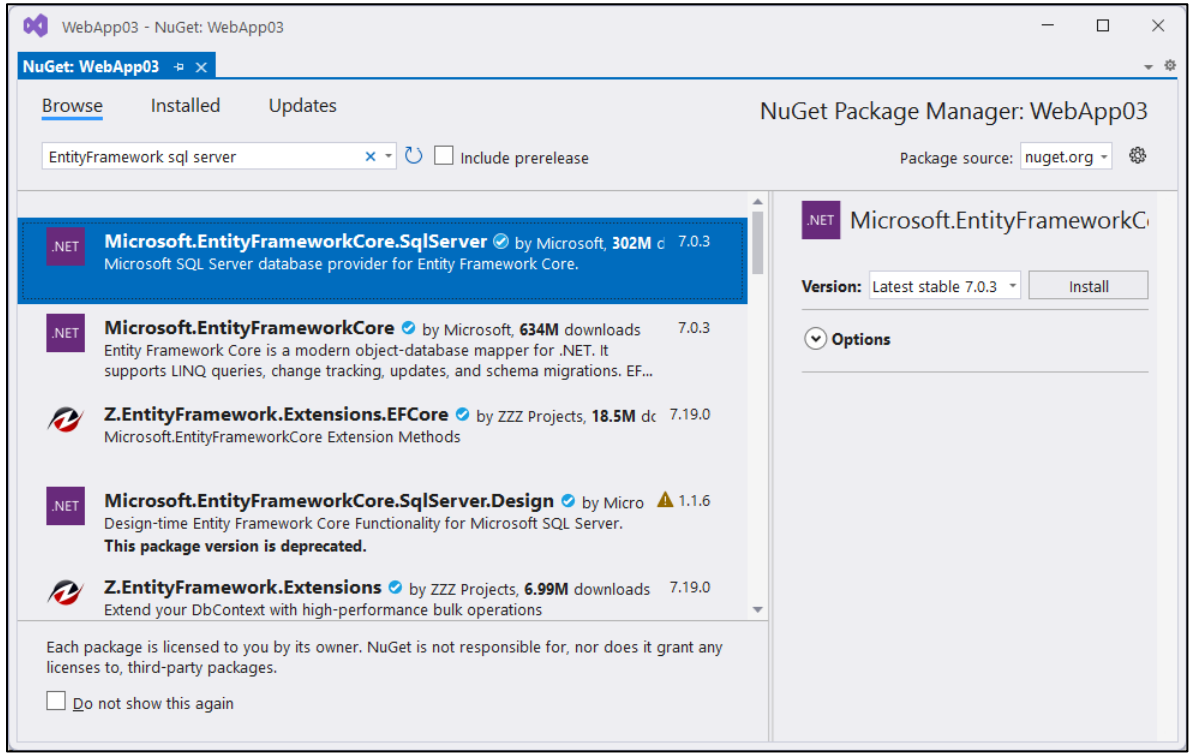
Kreiranje modela - code first pristup

Fajl appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=GORAN-HP;Initial Catalog=OsobaDb0903;
    Integrated Security=True;Encrypt=False"
  }
}
```

- appsettings.json sadrži konfiguraciju ASP.NET Core aplikacije
- Sekcija **ConnectionStrings** definiše parametre za povezivanje sa bazom podataka
- DefaultConnection sadrži string za povezivanje sa bazom **OsobaDb0903**

Microsoft.EntityFrameworkCore.SqlServer



- **Microsoft.EntityFrameworkCore.SqlServer** je EF Core provider za SQL Server
- Omogućava aplikaciji da pristupa bazi podataka preko **Entity Framework Core** biblioteke
- Paket se instalira **pomoću NuGet Package Manager**-a

Klasa OsobaContext

```
using Microsoft.EntityFrameworkCore;
```

```
public class OsobaContext : DbContext
{
    public DbSet<Osoba> Osobe { get; set; }
}
```

- Klasa **OsobaContext** predstavlja kontekst baze podataka
- Nasleđuje klasu **DbContext** iz biblioteke Entity Framework Core
- Svojstvo **DbSet<Osoba>** omogućava pristup podacima iz tabele Osobe

Registracija servisa DbContext

```
using Microsoft.EntityFrameworkCore;
using WebApp03.Models;
```

```
// Add services to the container.
builder.Services.AddControllersWithViews();
var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");

builder.Services.AddDbContext<OsobaContext>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddTransient<ISkladiste, OsobaSkladiste>();
```

- Iz konfiguracije aplikacije čita se connection string **DefaultConnection**
- Metodom **AddDbContext<OsobaContext>** registruje se DbContext i konfiguriše SQL Server provider

Registracija DbContext klase u DI kontejneru

- Metoda **AddDbContext**<**TContext**> registruje klasu OsobaContext u DI kontejneru
- Registruje se i objekat **DbContextOptions**<**OsobaContext**> koji sadrži konfiguraciju konteksta
- Objekat DbContextOptions<OsobaContext> definiše način povezivanja sa bazom podataka (provider i connection string).
- Registracijom se omogućava da DI kontejner kreira objekte klase **OsobaContext** kada su potrebni u drugim klasama

Konstruktor klase OsobaContext

```
using Microsoft.EntityFrameworkCore;
```

```
public class OsobaContext : DbContext
{
    public OsobaContext(DbContextOptions<OsobaContext> opcije) : base(opcije)
    {
    }

    public DbSet<Osoba> Osobe { get; set; }

}
```

- Konstruktor klase **OsobaContext** prima objekat **DbContextOptions<OsobaContext>**
- Ovaj objekat prosleđuje DI kontejner prilikom kreiranja konteksta
- DbContextOptions<OsobaContext> sadrži konfiguraciju za povezivanje sa bazom podataka

Implementacija skladišta korišćenjem DbContext klase

```
public class OsobaSkladisteDb : ISkladiste
{

    private readonly OsobaContext _db;
    public OsobaSkladisteDb(OsobaContext db)
    {
        _db = db;
    }

    public IEnumerable<Osoba> Osobe => _db.Osobe;

}
```

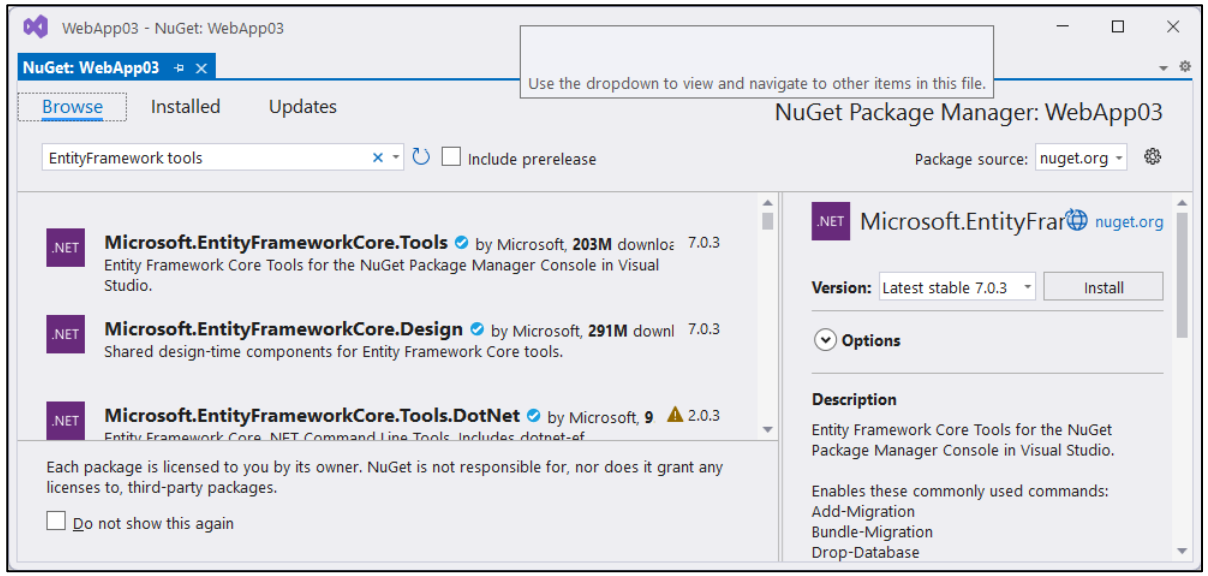
- Implementacija interfejsa **ISkladiste** koja koristi bazu podataka
- Klasa dobija objekat **OsobaContext** kroz konstruktor
- Podaci o osobama čitaju se iz **DbSet<Osoba> Osobe**

Promena implementacione klase servisa

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
//builder.Services.AddTransient<ISkladiste, OsobaSkladiste>();
builder.Services.AddTransient<ISkladiste, OsobaSkladisteDb>();
```

- Menja se implementacija interfejsa **ISkladiste**
- Umesto memorijskog skladišta koristi se skladište koje pristupa bazi podataka
- Kod kontrolera nije potrebno menjati

Microsoft.EntityFrameworkCore.Tools

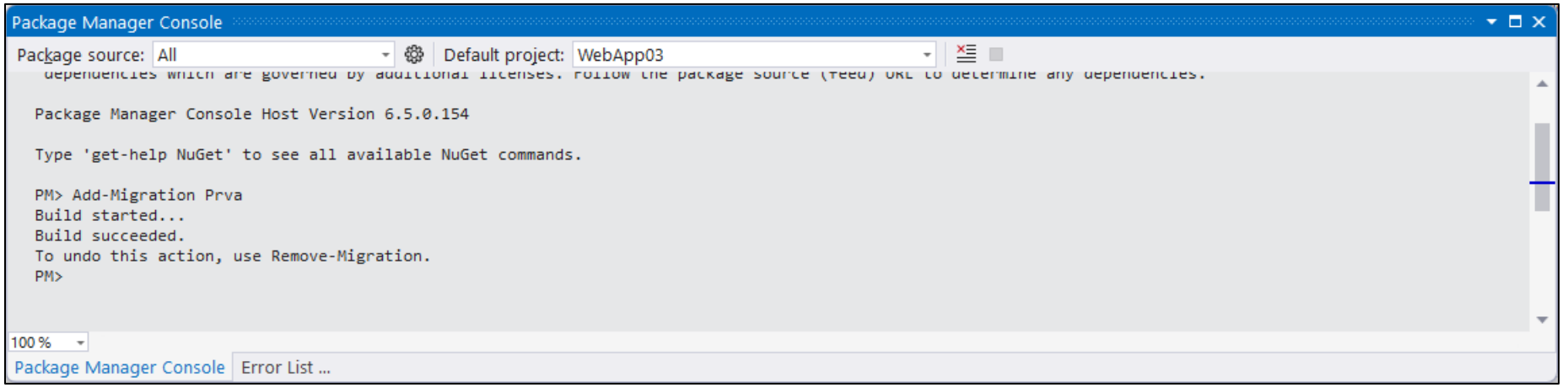


- Paket **Microsoft.EntityFrameworkCore.Tools** instalira alate za rad sa EF Core
- Alati se koriste u **Package Manager Console** u Visual Studio-u.
- Omogućavaju kreiranje baze podataka na osnovu definisanih klasa modela

Code first migracije

- Migracija predstavlja skup instrukcija koje opisuju kako treba izmeniti strukturu baze podataka
- Migracije omogućavaju kreiranje baze podataka na osnovu entitetskih klasa i DbContext klase
- Omogućavaju izmenu strukture baze podataka kada se promene entitetske ili DbContext klase
- Komanda **Add-Migration Prva** u alatu **Package Manager Console** kreira novu migraciju pod nazivom Prva.
- Nakon kreiranja migracije formira se folder **Migrations** u projektu ASP.NET Core MVC aplikacije

Dodavanje migracije



- Komandom Add-Migration Prva kreira se nova migracija
- Migracija sadrži instrukcije za kreiranje ili izmenu strukture baze podataka
- Nakon izvršavanja komande generišu se fajlovi migracije u folderu Migrations

Klasa Prva.cs

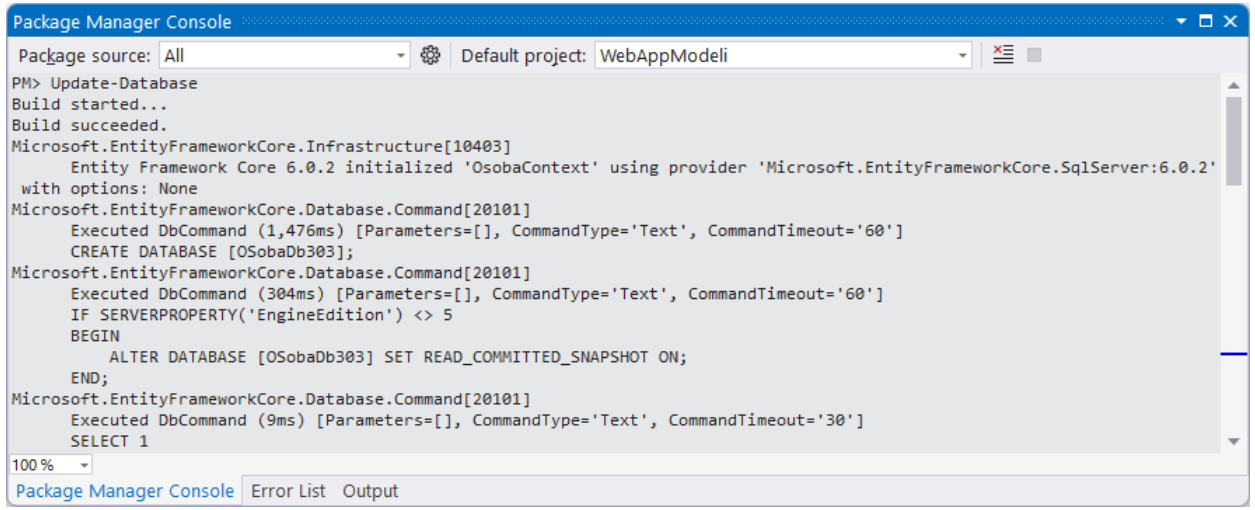
```
public partial class Prva : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Osoba",
            columns: table => new
            {
                OsobaId = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Ime = table.Column<string>(type: "nvarchar(max)", nullable: false),
                Prezime = table.Column<string>(type: "nvarchar(max)", nullable: false),
                Adresa = table.Column<string>(type: "nvarchar(max)", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Osoba", x => x.OsobaId);
            });
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Osoba");
    }
}
```

- Klasa **Prva** predstavlja generisanu migraciju
- Metoda **Up** definiše promene koje se primenjuju na bazi podataka
- Metoda **Down** definiše kako se te promene mogu poništiti

MigrationBuilder

- **MigrationBuilder** predstavlja objekat koji omogućava definisanje promena nad strukturom baze podataka
- Koristi se u metodama Up i Down klase migracije
- Pomoću njegovih metoda kreiraju se, menjaju ili brišu objekti baze (tabele, kolone, indeksi)
- **CreateTable**("Osoba") – definiše kreiranje tabele Osoba u bazi podataka
- **DropTable**("Osoba") – uklanja tabelu prilikom vraćanja migracije

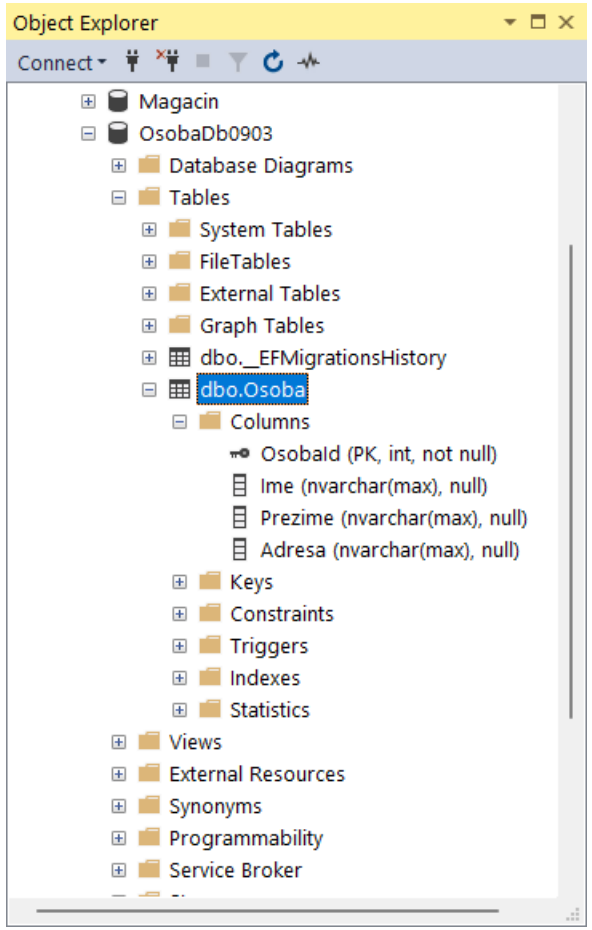
Primena migracije na bazu podataka




```
PM> Update-Database
```

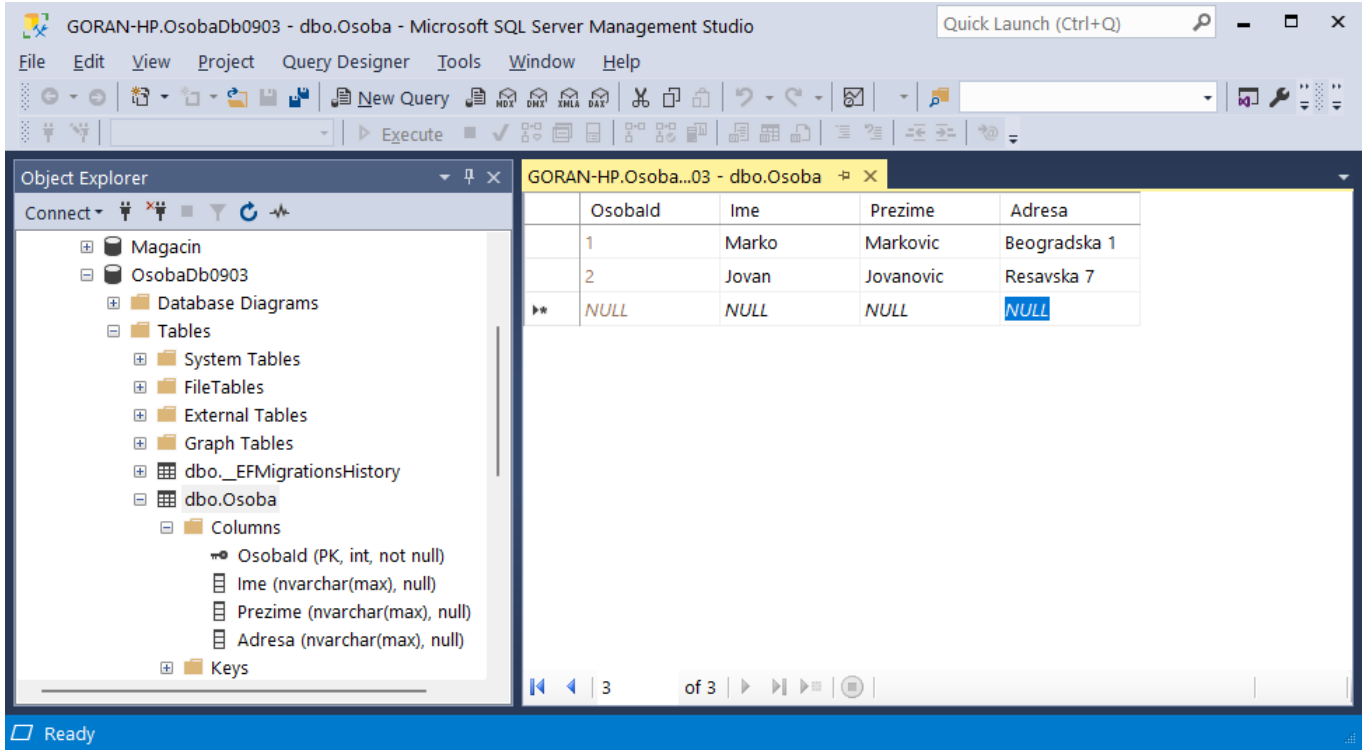
- Komanda **Update-Database** primenjuje migracije na bazu podataka
- Na osnovu definisanih migracija EF Core kreira ili menja strukturu baze
- U ovom slučaju kreira se tabela Osoba

Generisana baza podataka



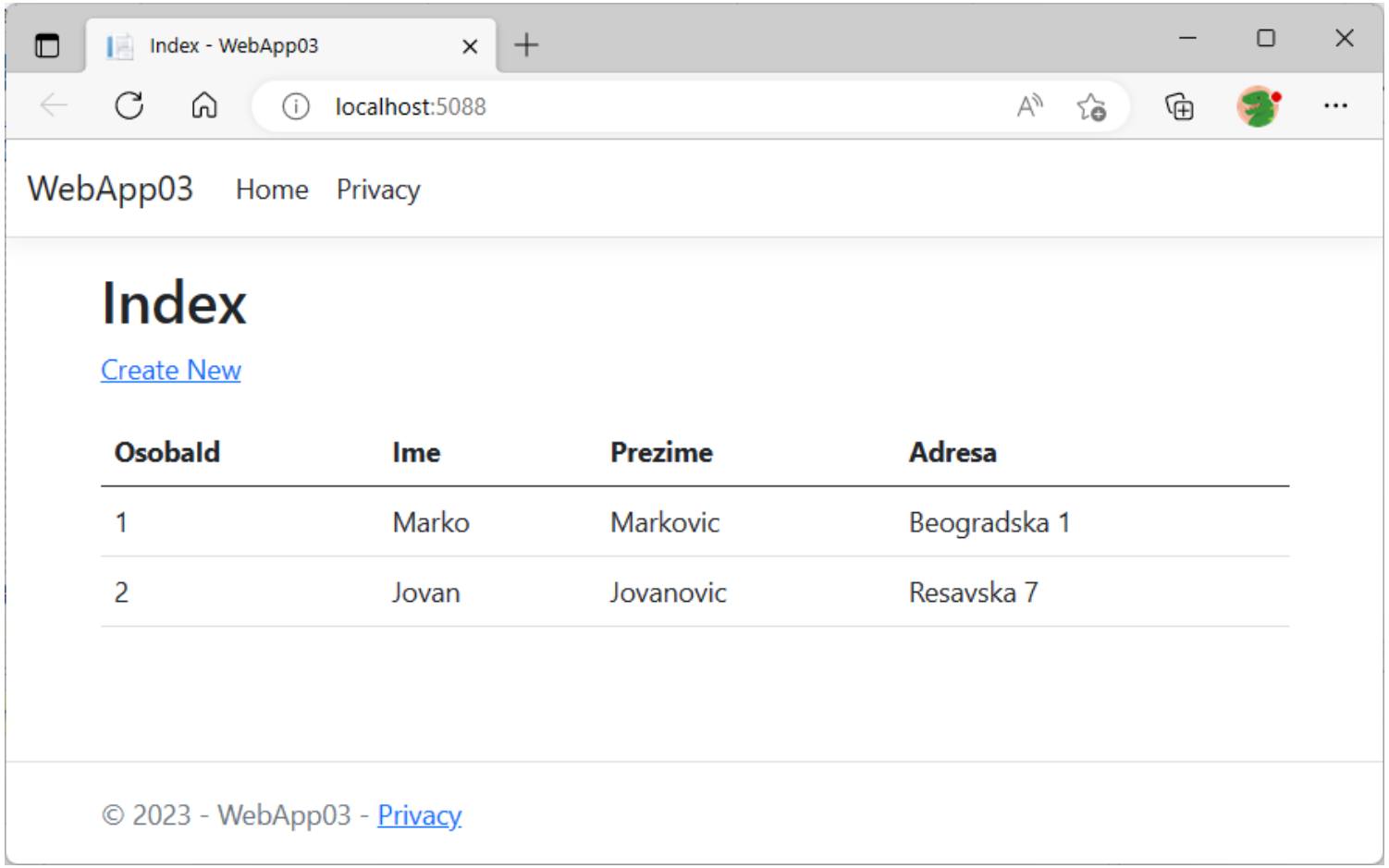
- Migracija je kreirala tabelu Osoba u bazi podataka
- Kolone su generisane na osnovu svojstava klase Osoba
- Pošto u klasi nisu korišćene anotacije za ograničenje dužine stringova, tip kolona je nvarchar(max)

Unos podataka u bazu

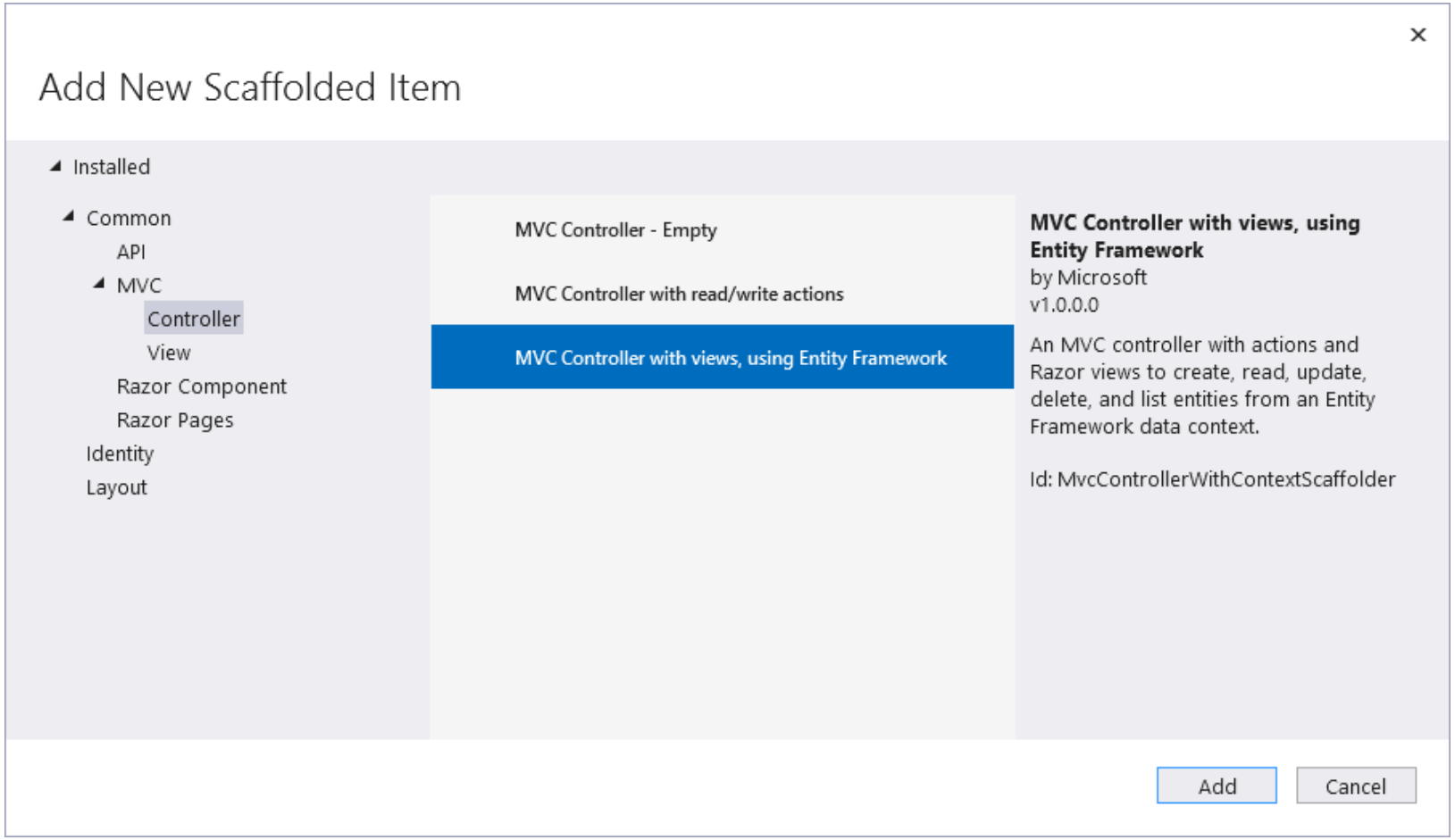


- Podaci se mogu uneti direktno u tabelu korišćenjem **SQL Server Management Studio** alata
- Nakon unosa podataka, aplikacija može da ih učita kroz Entity Framework Core i prikaže u MVC pogledu

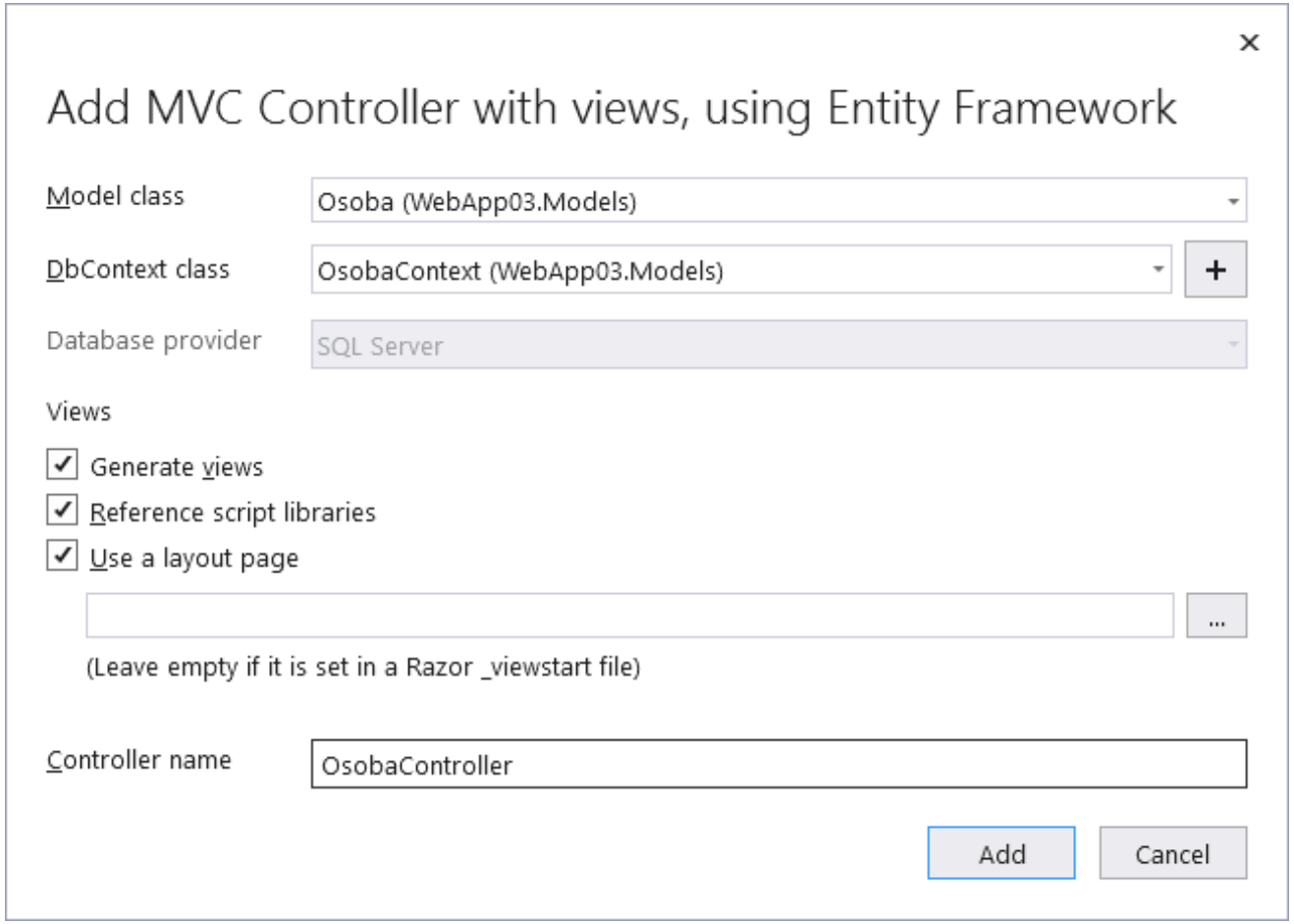
Index strana prikazuje podatake iz baze



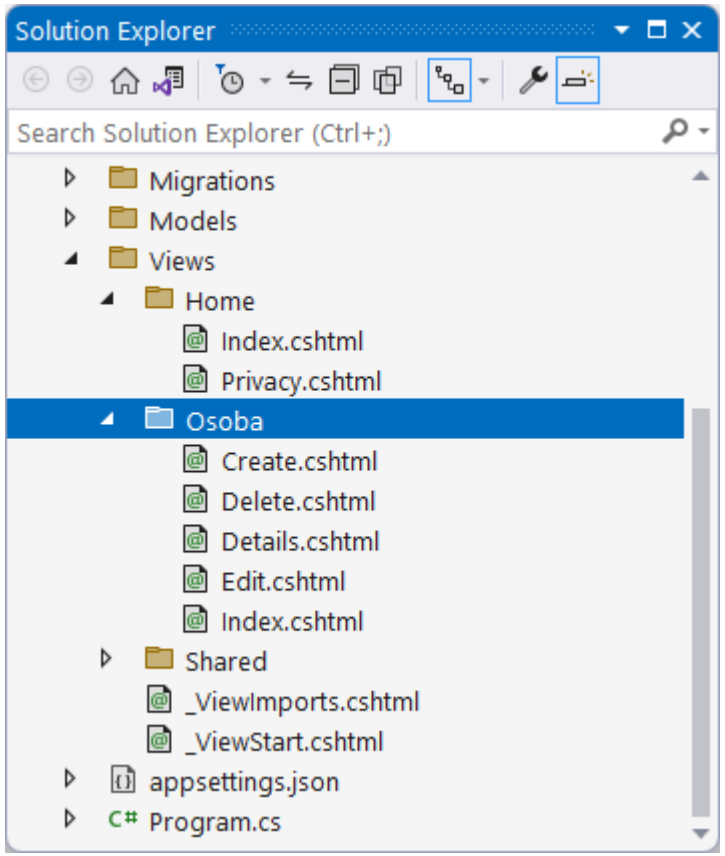
OsobaController



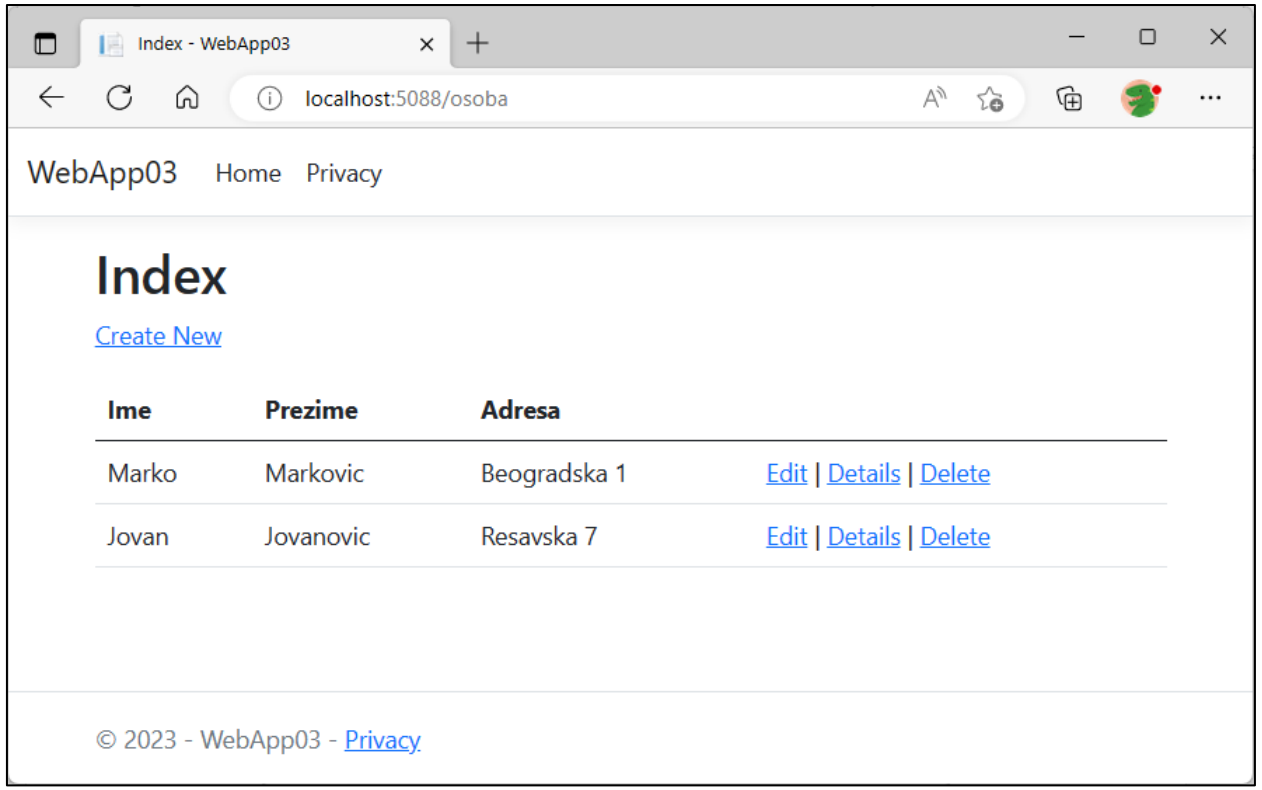
OsobaController



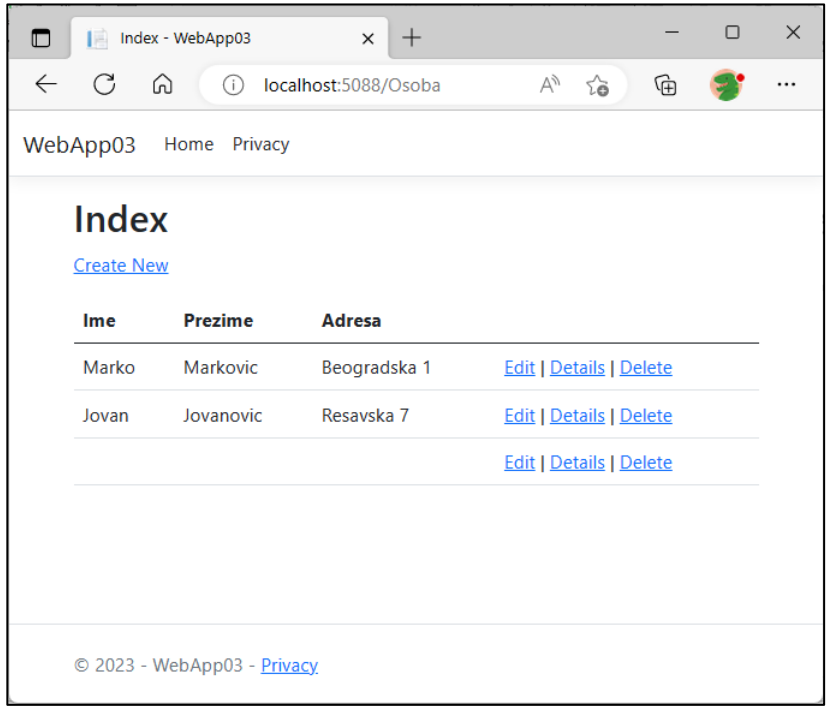
Generisani pogledi

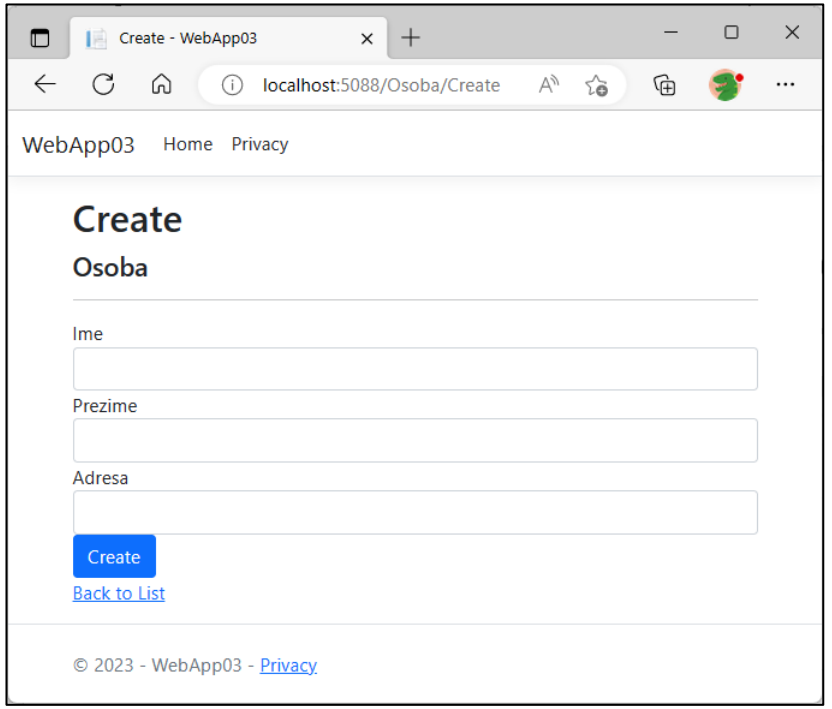


Index pogled klase OsobaController



Unos podataka kroz Create akciju





- Forma omogućava unos nove instance klase Osoba
- Pošto u modelu nisu definisane validacione anotacije, moguće je sačuvati red sa praznim vrednostima
- U sledećem koraku dodaju se anotacije za validaciju i generiše se nova migracija

Atributske klase

- **Atributske** klase omogućavaju dodavanje metapodataka elementima programa (klasama, svojstvima, metodama)
- **Data Annotations** su atributske klase koje se primenjuju na svojstvima modela
- U MVC aplikacijama koriste se za **validaciju podataka** i definisanje ograničenja modela
- Atribut **Required** definiše obavezno polje koje mora imati vrednost
- Atribut **StringLength** ograničava maksimalnu dužinu tekstualnog polja u bazi podataka
- Promena modela zahteva generisanje nove migracije kako bi se promenila struktura baze podataka

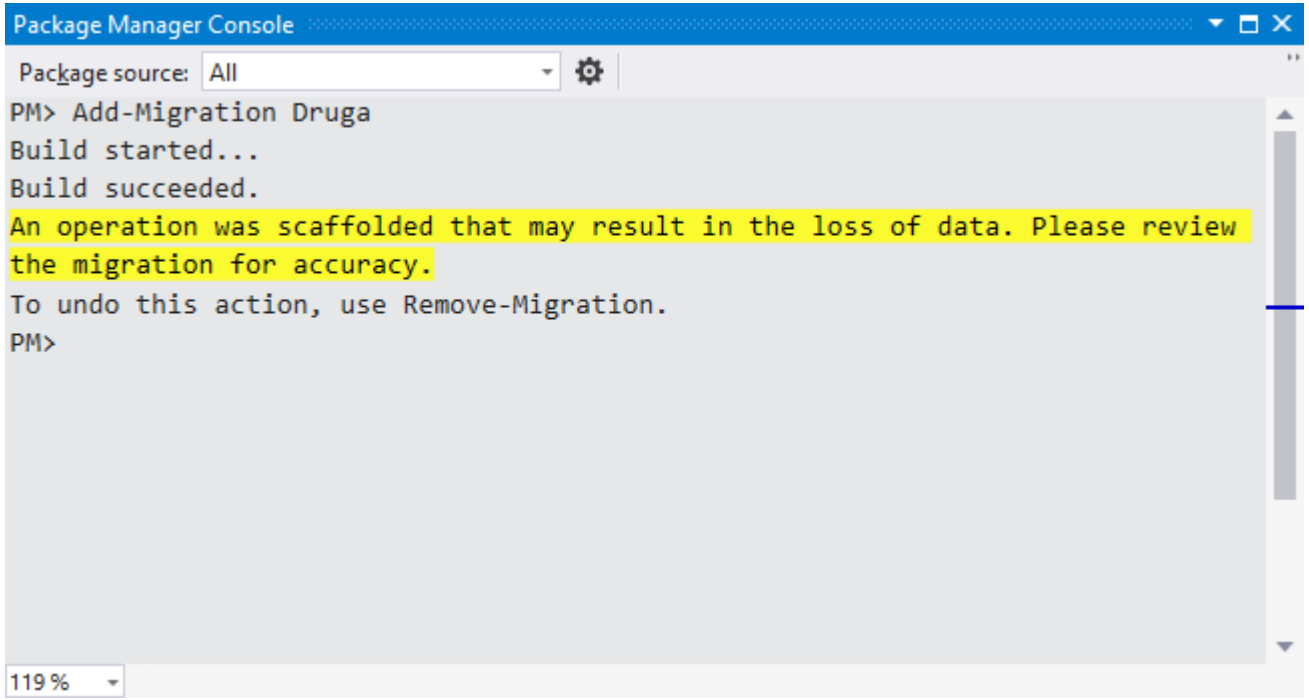
Model klasa sa anotacijama polja

```
using System.ComponentModel.DataAnnotations;
```

```
[Table("Osoba")]

    public class Osoba
    {
        public int OsobaId { get; set; }
        [Required(ErrorMessage = "Morate uneti ime")]
        [StringLength(30, ErrorMessage = "Ime je duzine do 30 karaktera")]
        public string Ime { get; set; }


        [Required(ErrorMessage = "Morate uneti prezime")]
        [StringLength(30, ErrorMessage = "Prezime je duzine do 30 karaktera")]
        public string Prezime { get; set; }

        [Required(ErrorMessage = "Morate uneti adresu")]
        [StringLength(60, ErrorMessage = "Adresa je duzine do 60 karaktera")]
        public string Adresa { get; set; }

    }
```

Dodavanje nove migracije

```
PM> Add-Migration Druga
```



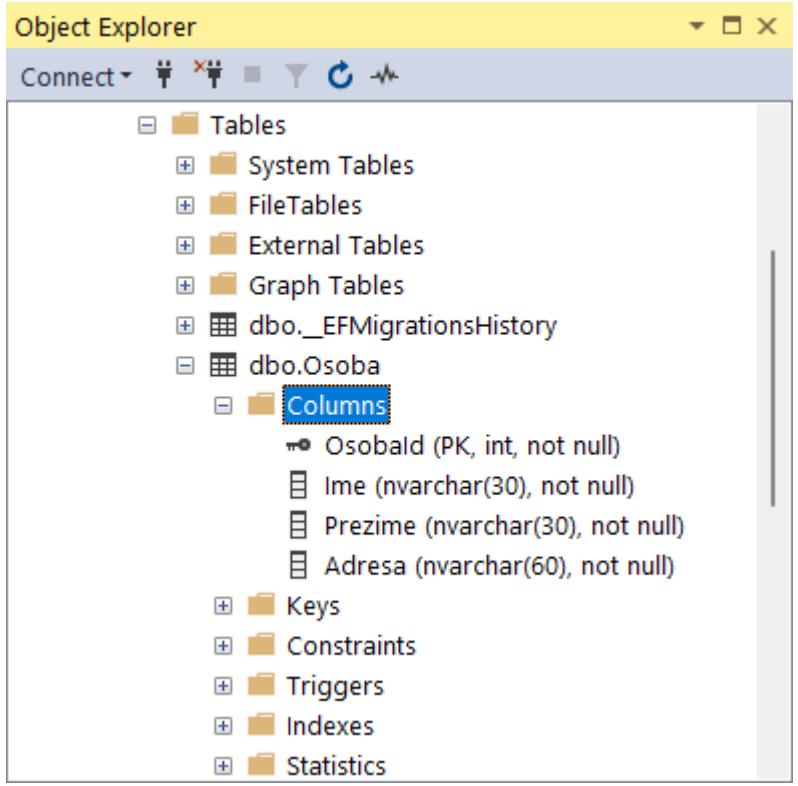
Metoda Up() migracije

```
public partial class Druga : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AlterColumn<string>(
                name: "Prezime",
                table: "Osoba",
                type: "nvarchar(30)",
                maxLength: 30,
                nullable: false,
                oldClrType: typeof(string),
                oldType: "nvarchar(max)");

            migrationBuilder.AlterColumn<string>(
                name: "Ime",
                table: "Osoba",
                type: "nvarchar(30)",
                maxLength: 30,
                nullable: false,
                oldClrType: typeof(string),
                oldType: "nvarchar(max)");

            migrationBuilder.AlterColumn<string>(
                name: "Adresa",
                table: "Osoba",
                type: "nvarchar(60)",
                maxLength: 60,
                nullable: false,
                oldClrType: typeof(string),
                oldType: "nvarchar(max)");
        }
```

- Nakon izmene model klase generisana je nova migracija **Druga**
- Metoda Up() definiše promene koje treba primeniti nad strukturom baze podataka
- U ovom slučaju menja se tip kolona Ime, Prezime i Adresa na nvarchar(30) i nvarchar(60), kao i njihova obaveznost
- Ove promene su posledica dodavanja atributa Required i StringLength u model klasi
- U praksi se model klase obično definišu sa odgovarajućim anotacijama odmah na početku, kako bi se izbegle naknadne izmene modela i generisanje dodatnih migracija

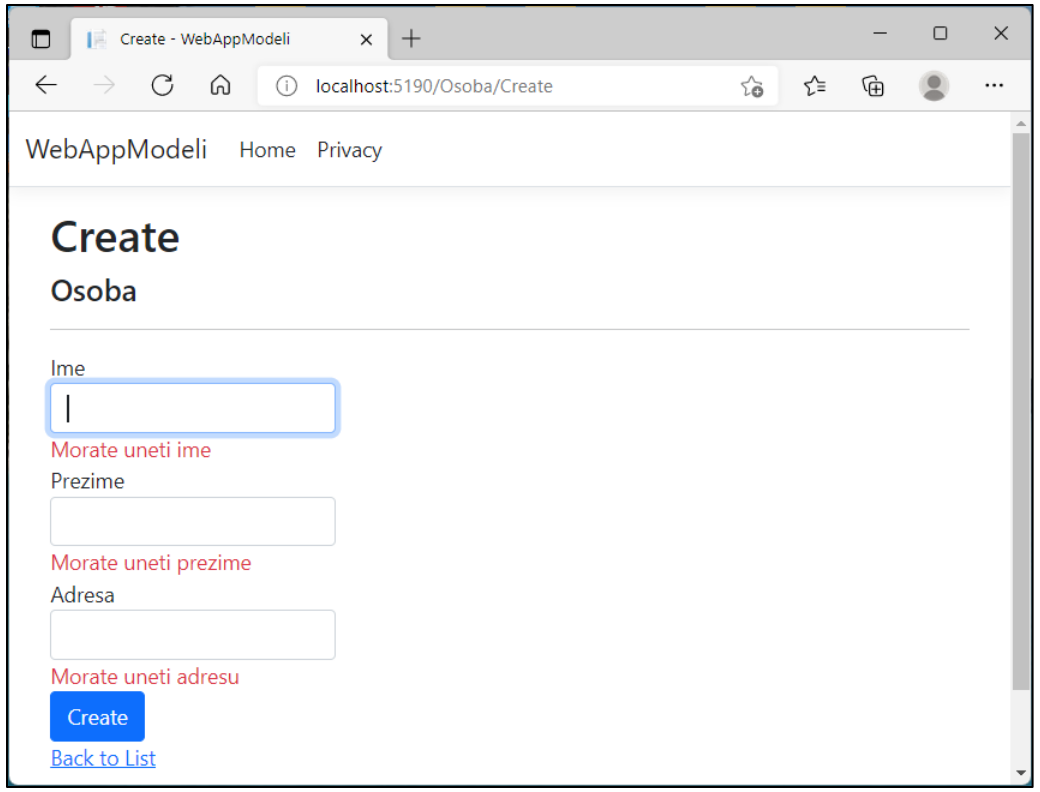
Primena druge migracije

```
PM> Update-Database
```



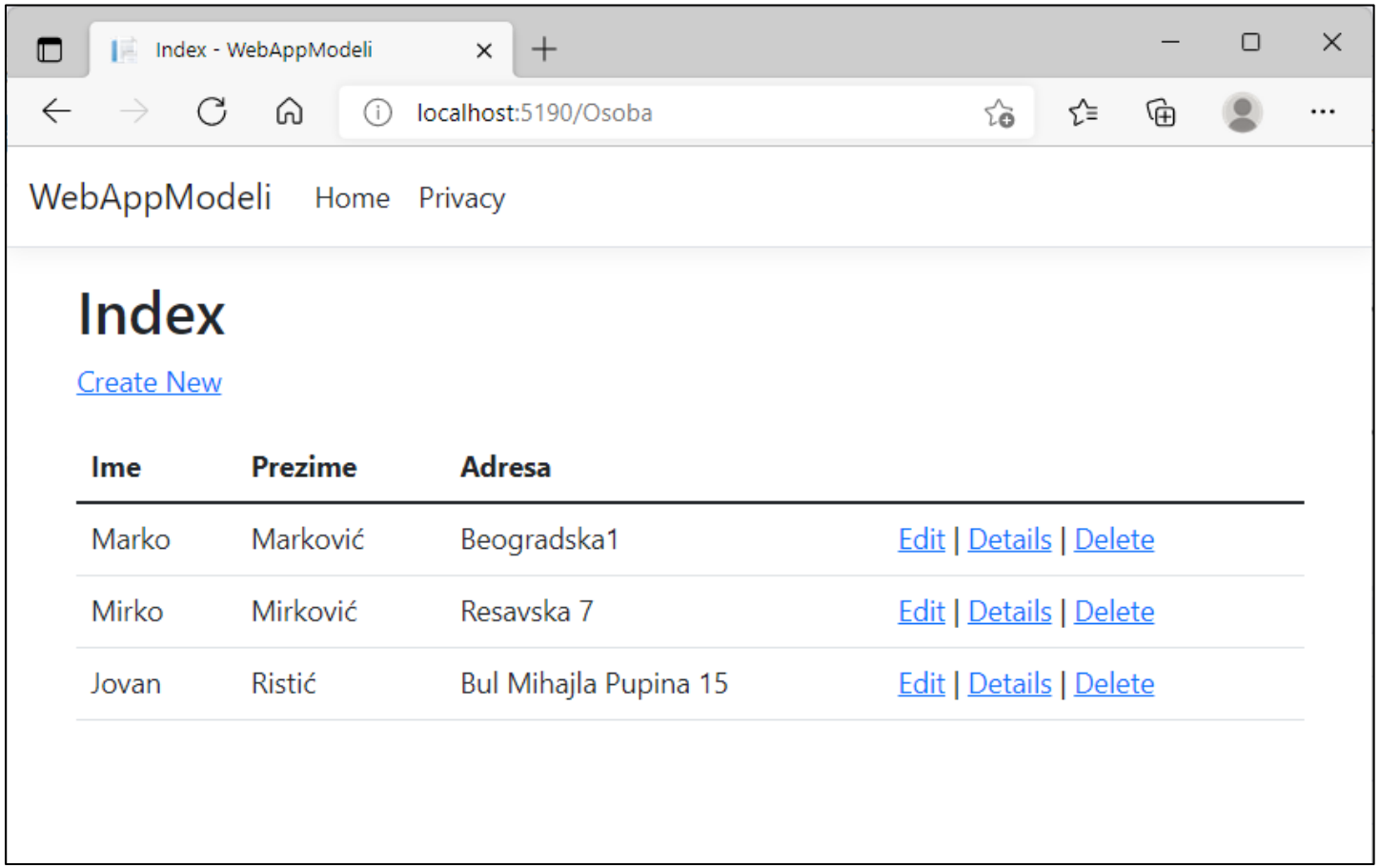
- Komandom Update-Database primenjuje se migracija Druga na bazu podataka
- Struktura tabele Osoba se menja u skladu sa izmenama model klase
- Kolone Ime i Prezime sada imaju tip nvarchar(30), dok kolona Adresa ima tip nvarchar(60) i definisana su kao obavezna polja

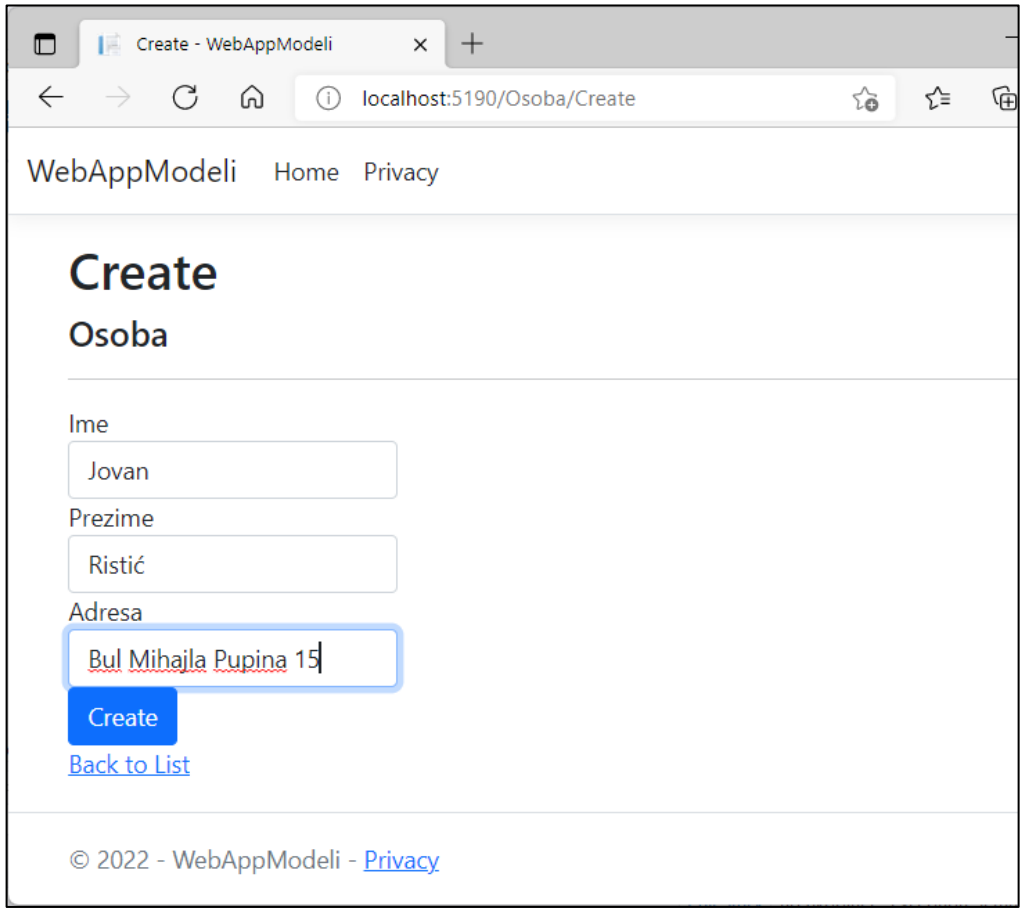
Validacija podataka



- Validacija podataka u MVC aplikaciji izvršava se na klijentskoj i serverskoj strani
- Na osnovu atributskih klasa u modelu automatski se generiše JavaScript kod koji omogućava validaciju u browseru pre slanja forme
- Nakon slanja forme vrši se i serverska validacija u okviru MVC aplikacije kako bi se proverila ispravnost podataka
- Serverska validacija je neophodna jer korisnik može isključiti JavaScript u broseru ili poslati zahtev direktno ka serveru, čime bi se zaobišla klijentska validacija
- Ukoliko validacija nije uspešna, korisniku se prikazuju odgovarajuće poruke o grešci

Unos podataka





Kreiranje modela na osnovu baze

Kreiranje baze

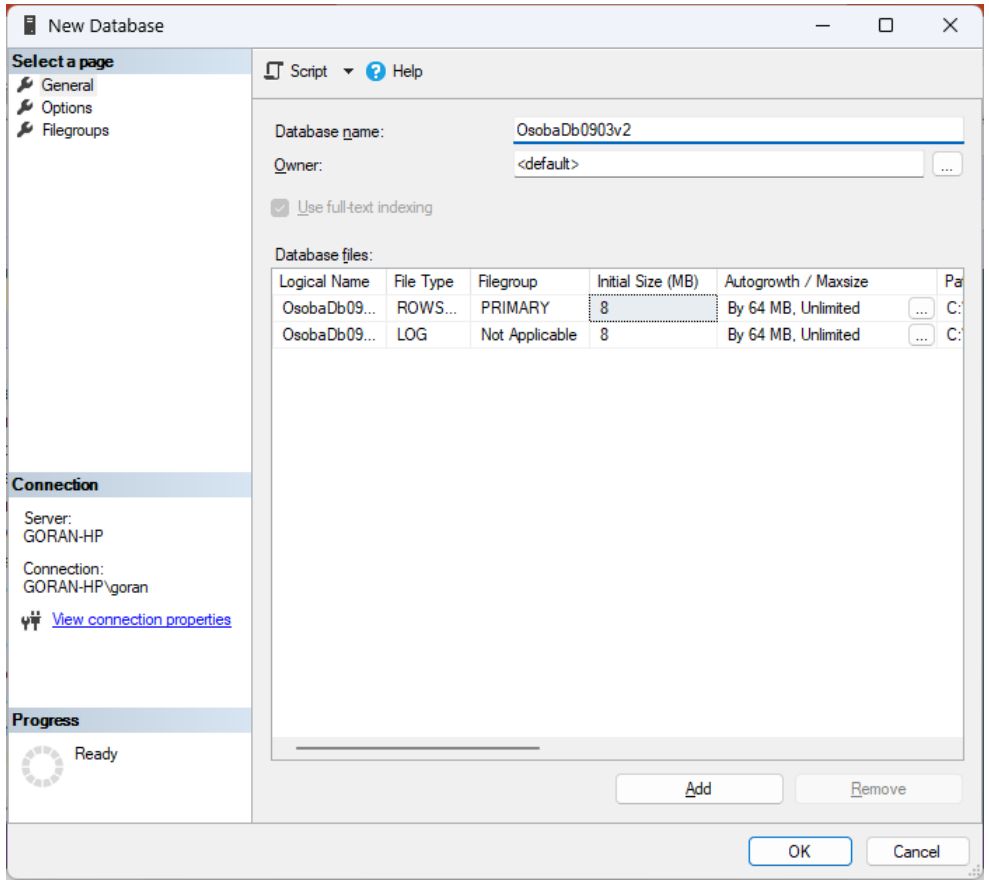
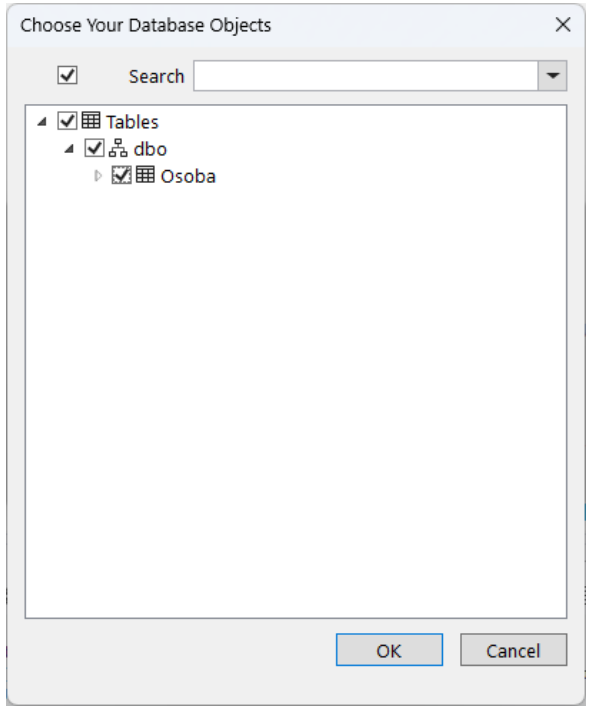


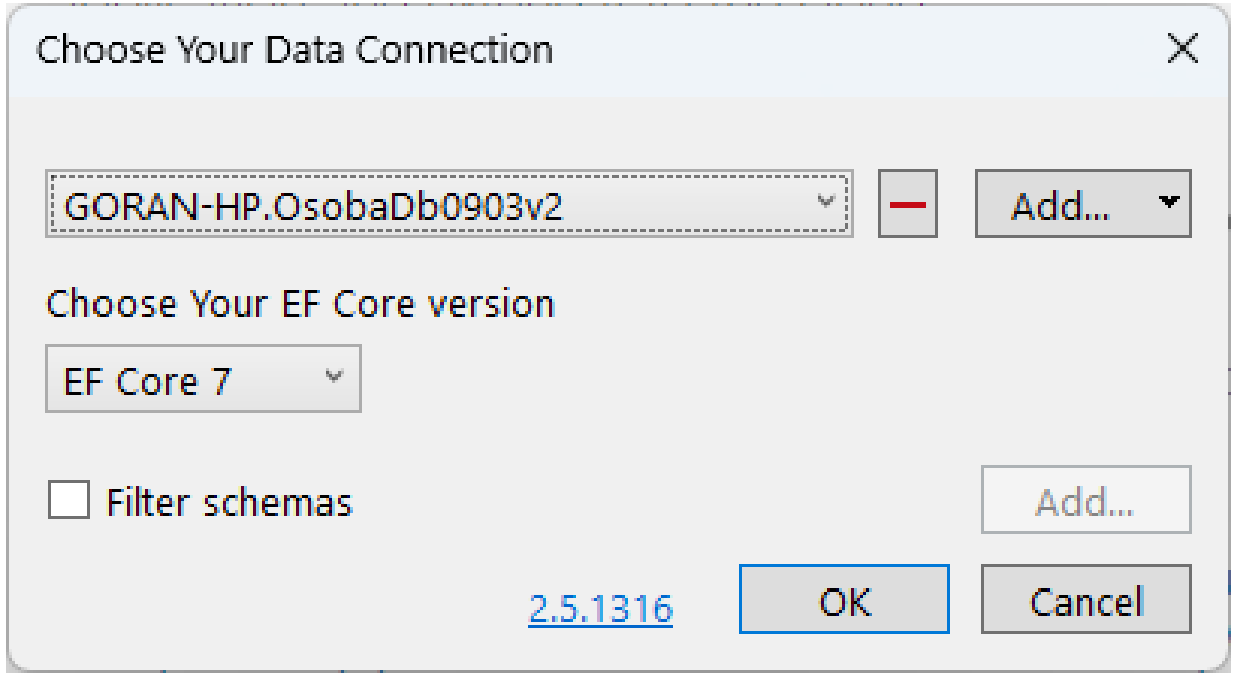
Tabela u bazi OsobaDb303ver2

```
CREATE TABLE Osoba(
OsobaId INT IDENTITY(1,1) PRIMARY KEY NOT NULL,
Ime NVARCHAR(30) NOT NULL,
Prezime NVARCHAR(30) NOT NULL,
Adresa NVARCHAR(60) NOT NULL
)
```

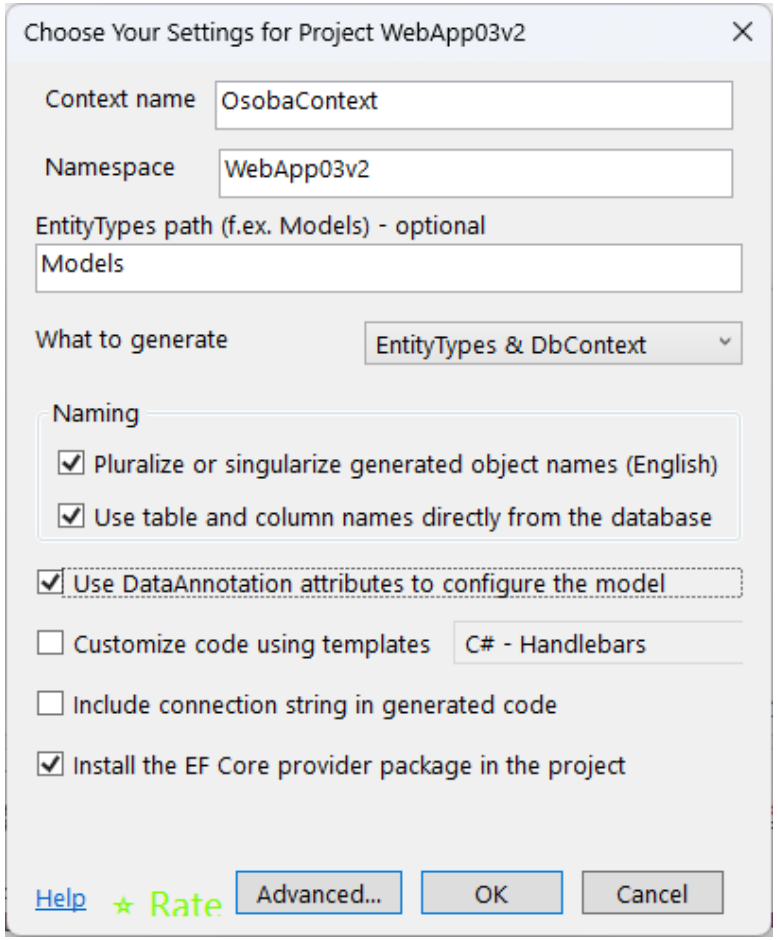
- U bazi podataka kreirana je tabela Osoba SQL komandom CREATE TABLE
- Definisane su kolone, tipovi podataka i ograničenja
- Na osnovu postojeće baze moguće je generisati model klase i DbContext
- Ovaj pristup naziva se **Database First**

EF Core Power Tools -1





EF Core Power Tools -2



Generisana entitetska klasa

```
[Table("Osoba")]
    public partial class Osoba
    {
        [Key]
        public int OsobaId { get; set; }

        [Required]
        [StringLength(50)]
        public string Ime { get; set; }

        [Required]
        [StringLength(30)]
        public string Prezime { get; set; }

        [Required]
        [StringLength(30)]
        public string Adresa { get; set; }
    }
```

- Entitetska klasa Osoba generisana je na osnovu postojeće tabele u bazi podataka
- Klasa je automatski kreirana pomoću alata **EF Core Power Tools**
- Generisani su atributi koji definišu mapiranje tabele i ograničenja kolona
- Ovaj postupak predstavlja **Database First** pristup u razvoju aplikacije

Generisana DbContext klasa

```
public partial class OsobaContext : DbContext
{
    public OsobaContext(DbContextOptions<OsobaContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Osoba> Osobe { get; set; }

}
```

- Klasa OsobaContext generisana je na osnovu postojeće baze podataka pomoću alata EF Core Power Tools
- Nasleđuje klasu DbContext i predstavlja vezu između aplikacije i baze podataka
- Svojstvo DbSet<Osoba> Osobe predstavlja kolekciju entiteta koja odgovara tabeli Osoba u bazi
- Ova klasa omogućava rad sa podacima korišćenjem Entity Framework Core biblioteke

Modifikovani appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=Goran-HP;Initial Catalog=OsobaDb0903v2;
     Integrated Security=True;Encrypt=False"
  }
}
```

- U fajlu appsettings.json definiše se konekcioni string za pristup bazi podataka
- Sekcija ConnectionStrings sadrži podatke o serveru i nazivu baze
- Vrednost **DefaultConnection** se **ručno upisuje** i prilagođava prema lokalnom SQL Server okruženju
- Ovaj konekcioni string koristi DbContext za povezivanje aplikacije sa bazom podataka

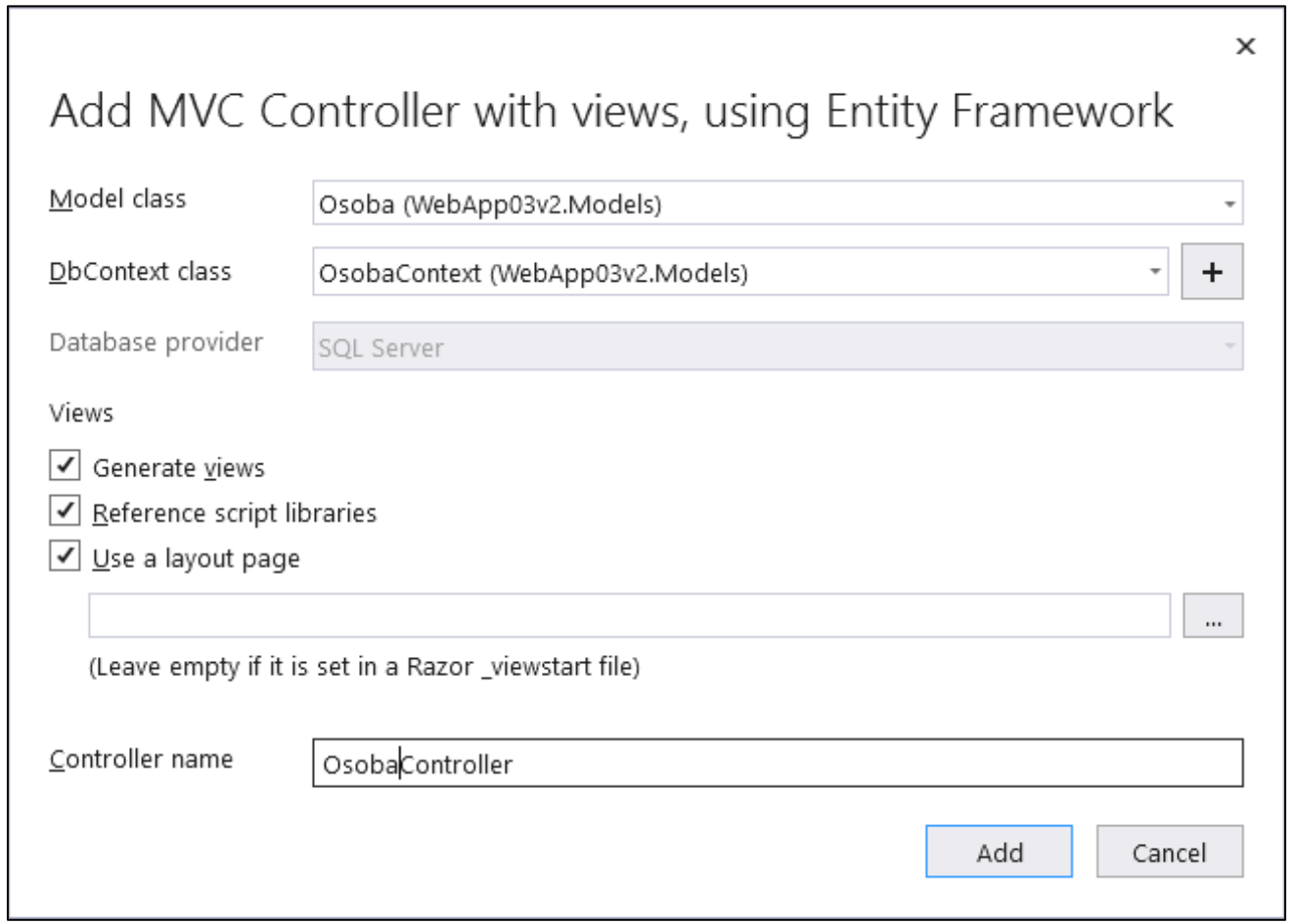
Registracija DbContext servisa

```
using Microsoft.EntityFrameworkCore;
using WebApp03v2.Models;
```

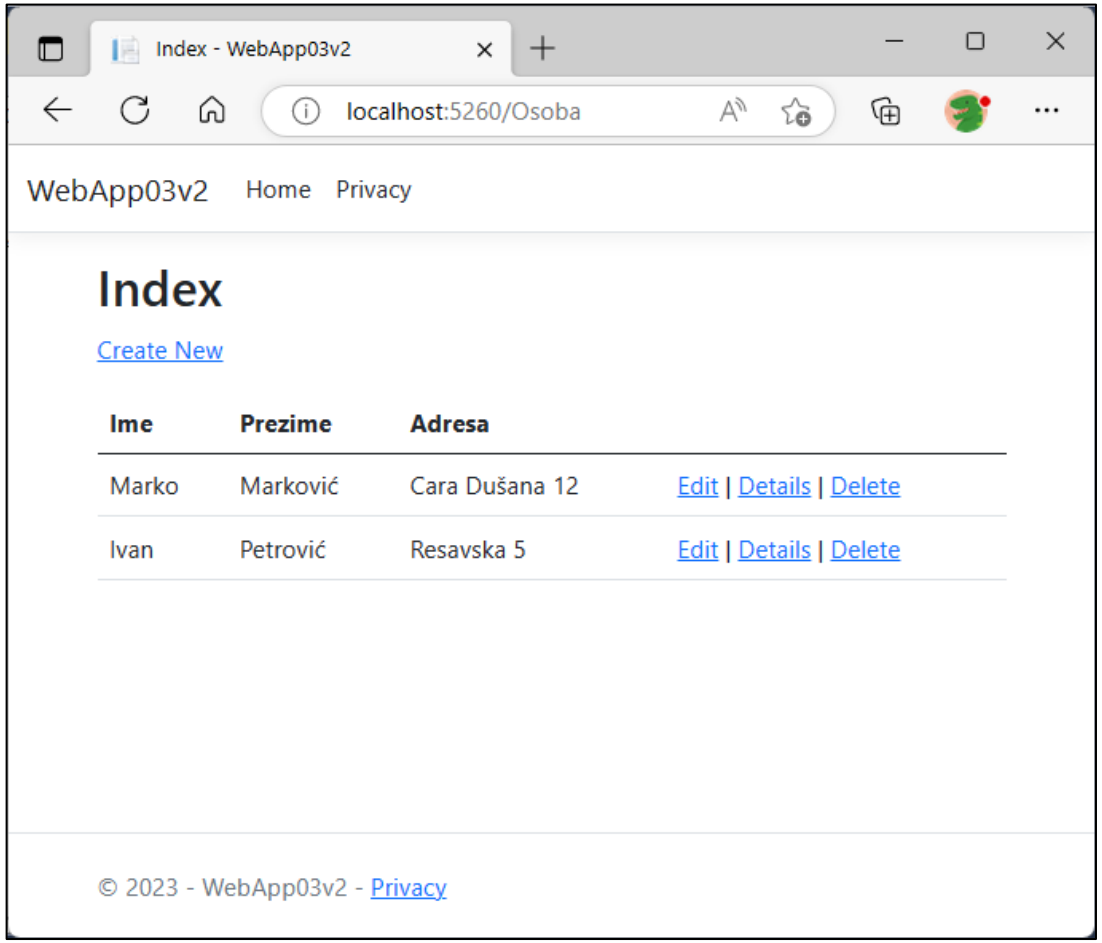
```
// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<OsobaContext>(opcije =>
    opcije.UseSqlServer(connectionString));
```

- Kod pristupa Database First potrebno je ručno registrovati DbContext u DI kontejneru
- Registracija omogućava korišćenje OsobaContext klase u kontrolerima aplikacije

Kreiranje OsobaController klase



Index pogled



Pitanje 1

Modeli podataka kod ASP.NET Core MVC web aplikacija definišu se unutar:

a. foldera Models
b. foldera wwwroot
c. foldera Views

Odgovor: a

Pitanje 2

Unutar pogleda (cshtml) fajla podacima koje prosleđuje kontroler pristupa se:

a. korišćenjem svojstva Model
b. korišćenjem ključne reči @model
c. korišćenjem klase ModelData

Odgovor: a

Pitanje 3

U Razor pogledu (.cshtml) direktiva @model koristi se za:

a. definisanje tipa modela koji pogled koristi
b. slanje podataka iz pogleda ka kontroleru
c. povezivanje aplikacije sa bazom podataka

Odgovor: a

Pitanje 4

Klasa DbContext u Entity Framework Core služi za:

a. definisanje HTML pogleda
b. komunikaciju aplikacije sa bazom podataka
c. validaciju podataka u formi

Odgovor: b

Pitanje 5

Komanda Add-Migration u Package Manager Console služi za:

a. pokretanje web aplikacije
b. generisanje promena strukture baze na osnovu modela
c. ubacivanje podataka u bazu podataka

Odgovor: b