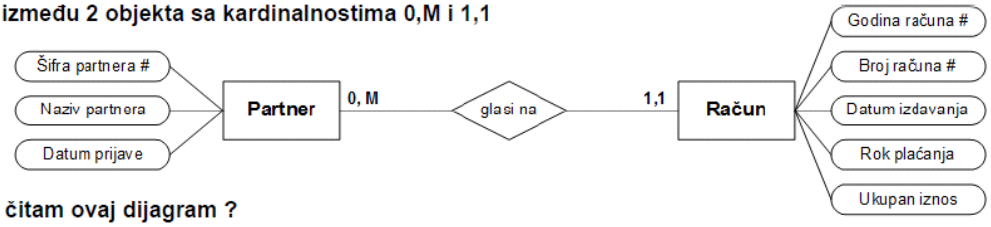
Racun je jak objekat i taj racun je identifikovan prkeo jedne stvari - IDRacuna**#**
Crta se kao pravougaonik.

Slab objekat je ono sto egzistencijalno tj. identifikaciono zavisi od jakog objekta. Ne moze da postoji bez njega.
Crta se kao pravougaonik unutar pravougaonika.

Veza između 2 objekta sa kardinalnostima 0,M i 1,1



Kako čitam ovaj dijagram ?

- Šifra partnera je jedinstveni identifikator objekta Partner, a Godina Računa i Broj računa su jedinstveni identifikatori objekta Račun
- prilikom jednog pojavljivanja jakog objekta Račun, jak objekat Partner se može pojaviti najmanje jednom i najviše jednom, dakle uvek samo jednom !
- prilikom jednog pojavljivanja jakog objekta Partner, jak objeka Račun se može pojaviti nijednom ili više puta (oznaka M)
- veza je binarna (ide u 2 smera):
    - u smeru od Računa ka Partneru veza je "Račun glasi na Partnera"
    - u smeru od Partnera ka Računu veza je "Partner plaća Račun"
    - na dijagramu nismo u obavezi da prikažemo naziv veze, ali dobro je imati naziv makar u jednom smeru

1. Racunu moze da pripada 1 i samo 1 partner.
2. Partneru moze da pripada 0 ili vise racuna.

Specijalizacija : Partner moze da bude ILI fizicko lice (koji ima JMBG) ILI pravno lice (PIB), ali ne moze oba. Oba nasledjuju partnera (romb S izmedju).

Agregacija : Nesto sto ne moze samostalno da postoji, vec je ti objekti identifikuju.
Oznaka je romb u pravougaoniku.

Novi cas - 10.03.18.

Za intervju za posao proci:
1. Design Pattern (GoF) - Mark Farragher
2. SOLID (Objasniti nesto o SOLID principima dizajniranja koda) - Uncle Bob knjiga
3. Principi : IOC - Inversion of Control i DI - Dependency Injection

CTOR <TAB> <TAB> - Pravi automatski konstruktor da ne bi pisali sve
CW <TAB><TAB> - Console.WriteLine

Static

r^2Pi je sa svaki krug isti, trebalo bi omoguciti samo izmenu poluprecnika. Pored toga, ako pozovemo 10 puta nesto trosimo memoriju. U memoriji necemo da imamo pi za onoliko objekata koliko ih imamo, pi je uvek isti za svaki krug. Ovo se radi preko **static**.

```
7   namespace ConsoleApp2
8   {
9       class Krug
10      {
11          float _PI = 3.141f;
12          int _poluprecnik;
13
14          public Krug(int _poluprecnik)
15          {
16              this._poluprecnik = _poluprecnik;
17          }
18
19          public float SracunajPovrsinuKruga()
20          {
21              return _poluprecnik * _poluprecnik * _PI;
22          }
23      }
24
25      class Program
26      {
27          static void Main(string[] args)
28          {
29              Krug k = new Krug(10);
30              // da je public float _PI, neko bi ovde mogao da promeni konstantu.
31              // tj. k._PI = 100;
32              Console.WriteLine("Povrsina je: {0}", k.SracunajPovrsinuKruga());
33          }
34      }
35  }
```

Kad je nesto static, on pripada klasi, zove se preko klase, tj. tipa. A kad nesto nije static onda ga uvek zoves preko objekta tj. reference.

Red 11 : *static float _PI;*
Red 21 : *return this._poluprecnik * this._poluprecnik * **Krug**._PI;*
Pi je static i pristupamo mu preko klase, a _poluprecnik nije statik pa se poziva preko insance (this._poluprecnik)

Kad je nesto statik on pripada klasi, a ne instanci klase. Staticki clanovi se uvek pozivaju *ime_klase.ime_statickog_clana*. Kod objekta je *ime_instance.ime_clana*.

Sa staticom pozivamo preko tipa, a kada nesto nije static ono pripada instanci tipa (klase).Staticke metode ne mogu da pristupaju nestatickim clanovima tj. clanovima instance (ne moze da se koristi ***this***).

```
namespace ConsoleApp2
{
    class Krug
    {
        static float _PI = 3.141f;
        int _poluprecnik;

        public Krug(int _poluprecnik)
        {
            this._poluprecnik = _poluprecnik;
        }

        public float SracunajPovrsinuKruga()
        {
            return _poluprecnik * _poluprecnik * Krug._PI;
        }

        public static float SracunajObimKruga(int _poluprecnik)
        {
            return 2 * _poluprecnik * Krug._PI;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Krug k = new Krug(10);
            Console.WriteLine("Povrsina je: {0}", k.SracunajPovrsinuKruga());
            Console.WriteLine("Obim je: {0}", Krug.SracunajObimKruga(10));
        }
    }
}
```

Staticki konstruktor sluzi za inicijalizaciju static polja, npr. on bi inicijalizovao PI. Staticki konstruktor se poziva pre konstruktora instance i on se uvek poziva samo jednom na pocetku bez obzira koliko instanci kreiramo.

```
7   namespace ConsoleApp2
8   {
9       class Krug
10      {
11          int _poluprecnik;
12          static float _PI;
13
14          static Krug()
15          { // Sluzi za inicijalizaciju vrednosti statickih polja.
16              Console.WriteLine("Pozivam staticku konstruktor.");
17              Krug._PI = 3.141f;
18          }
19
20          public Krug(int _poluprecnik)
21          {
22              Console.WriteLine("Pozivam konstruktor instance.");
23              // This se odnosi na instancu ove klase tj. na novi objekat.
24              this._poluprecnik = _poluprecnik;
25          }
26
27          public float SracunajPovrsinuKruga()
28          {
29              return _poluprecnik * _poluprecnik * Krug._PI;
30          }
31
32          public static float SracunajObimKruga(int _poluprecnik)
33          {
34              return 2 * _poluprecnik * Krug._PI;
35          }
36      }
```

Console.WriteLine - Staticka metoda staticke klase *Console*.

I sama klasa moze da bude staticka, i ona moze da ima samo staticke clanove u sebi. Ona sluzi kao neko skladiste koja sadrzi staticke metode koje imaju neke ulazne parametre i ona ne moraju da postavalju interne instance tih promenjivih. *Primer - klasa math. Math.Pow(3,2);*

Staticka klasa

```
static class Kvadrat
{
    static float kvadriraj(float broj)
    {
        return broj * broj;
    }
}
// Console.WriteLine("Kvadrat broja 7 je {0}", Kvadraft.kvadriraj(7));
```

Niz

Niz je referentan tip podatka koji sadrzi promenjive istog tipa.
tip[] ime; - Zagrade su rang niza (tj. da li je jednodimenzonalan, matrica...)

Deklarisanjem niza kreirana je promenjiva koja ce sadrzati referencu na instancu niza.

int[] niz;
niz = new int [5]; // Promenjiva *niz* pokazuje na niz od 5 elemenata.
int[] niz = new int[] {1, 2, 3, 4, 5};

```
7   namespace ConsoleApp3
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              int[] niz = new int[5];
14              // Pristupanje radimo preko indeksa.
15              niz[0] = 10;
16              niz[1] = 20;
17              niz[2] = 30;
18              niz[3] = 40;
19              niz[4] = 50;
20
21              foreach(int i in niz)
22              {
23                  Console.WriteLine(i);
24              }
25
26              // Da je for petlja onda bi bilo niz[i], ovako je vec uhvatio.
27          }
28      }
29  }
```

Matrice

```
int[,] matrica = new int[3, 2]; // Redovi, kolone
matrica[0, 0] = 1;
matrica[0, 1] = 2;
//...

// GetLength(dimenzija)
for (int red = 0; red < matrica.GetLength(0); red++)
{
    for (int kolona = 0; kolona < matrica.GetLength(1); kolona++)
    {
        Console.WriteLine("Element matrice[{0}][{1}] = {2})",
            red + 1, kolona + 1, matrica[red, kolona]);
        // +1 jer indeks ide od nule, pa da pise [1,1]
    }
}
// U jednom redu kupi sve kolone, pa onda prelazi na drugi red.
```

Trodimenzionalni niz

Isti princip samo sto dodajemo jos jednu for petlju i menjamo dimenziju u GetLength.
int[,,] niz3D = new int[,,] { { {1,2,3},{4,5,6}, {7,8,9}, {10,11,12}}}

Kopiranje niza

```
int[] nizz = new int[5] { 1, 2, 3, 4, 5 };
int[] kopijaNiza;
kopijaNiza = nizz;
// Ono sto je u nizz ubaci mi u kopijaNiza. Sada i kopijaNiza ima adresu
// provg niza u sebi. Ne pravi se fizicka kopija niza, vec referenca na isti.
```

String

String je niz karaktera. On je immutable type - kad se jednom kreira ta sekvenca karaktera, ona vise ne moze da se menja.

string a = "zdravo";
string b = "z";
b += "dravo" *// Ne pokazuje na isto zdravo kao i a (kao kod kopiranje niza), vec pravi novu kopiju. I ono staro z kupi garbage collector.*
c = b;

Prvo poredjenje po referenci false jer su to dva razlicita objekta. U drugom slucaju TRUE jer c pokazuje na isti objekat kao i b. Na kraju su dva razlicita objekta pa je FALSE.

```
string a = "zdravo";
string b = "z";
b += "dravo";
Console.WriteLine(a == b); // Poredjenje vrednosti. - TRUE
Console.WriteLine((object)a == (object)b); // Poredjenje po referenci. - FALSE

string c;
c = b;
Console.WriteLine(b == c); // - TRUE
Console.WriteLine((object)b == (object)c); // - TRUE

b = "zdravo svima";
Console.WriteLine(b == c); // Poredjenje vrednosti. - FALSE
Console.WriteLine((object)b == (object)c); // - FALSE
```

StringBuilder

```
StringBuilder sb = new StringBuilder("Zdravo");
sb.Append(" svima!");

Console.WriteLine(sb);

// Index, vrednost.
sb.Insert(12, " ponovo");
// Index, duzina.
sb.Remove(7, 7);
// OldChar, NewChar
sb.Replace('o', '0');
```

```
string sut = "Sutra je novi dan.";
string[] d = sut.Split(' ');
// string d ima clanove 'Sutra', 'je', 'novi' i 'dan.'.
if (sut.Contains("dan.")) { } // Sadrzi 'dan.'.
if ((sut == string.Empty) || sut.Length == 0) { } // Prazan.
// ToLower(), ToUpper().
```

Escape sekvenca je direktna naredba operativnom sistemu. C:****Windows npr. Backslash je ovde escape sekvenca da ne bismo zadali direktnu naredbu OS-u.

Imenski prostor

Koristi se za organizovanje programa i prevenciju konflikata imena.
<naziv_namespace-a>.<naziv_clana>

Izbacimo using system i onda System.Console.WriteLine("Zdravo"); //ns.klasa.metoda_klase

```
namespace TimA
{
    class Klasa
    {
        void Stampaj()
        {
            Console.WriteLine("Stampam.");
        }
    }

}

namespace TimB
{
    class Klasa
    {
    void Stampaj()
        {
            Console.WriteLine("Stampam.");
        }
    }

}

// Iako su istog imena, iz razl. su namespace-a pa moze.
// TimA.Klasa.Stampaj();
// TimB.Klasa.Stampaj();
```

Enkapsulacija
Ucaurenje je kreiranje granica oko objekta da bi se razdvojilo njegove ekstreno tj. javno (public) ponasanje od njegovih internih tj. privatnih (private) implementacionih detalja.

https://stackoverflow.com/questions/16014290/simple-way-to-understand-encapsulation-and-abstraction

https://www.tutorialspoint.com/csharp/csharp_encapsulation.htm

Napravimo mehanizam koji govori koje vrednosti mogu uci i izaci. Npr. broj indeksa ne moze biti negativan, ime i

prezime se mora uneti... (pitamo sa if unutar metode, inace se promenjivoj za broj indeksa moze pristupiti samo preko te metode (sa getterima i setterima) jer bi ovako neko mogao da promeni broj indeksa da bude negativan npr.)

```
public class Student
{
    private int _indeks;

    public void SetBrojIndeksa(int BrojIndeksa)
    {
        if (BrojIndeksa < 0)
        {
            throw new Exception("Broj indeksa ne moze biti negativan.");
        }
        else
        {
            this._indeks = BrojIndeksa;
        }
    }

    public int GetBrojIndeksa()
    {
        return this._indeks;
    }
}
```

```
class Osoba
{
    public string JMBG;
    private string p_ime;
    public string ime
    {
        get { return p_ime; }
        set { this.p_ime = value; }
    }

    public void Hodaj (int metara)
    {
        return;
    }

    public int Test (int a)
    {
        return a * 2;
    }
}
```

```
class Student : Osoba
{
    public int Indeks;
    private int p_GodinaStudija;
    public int GodinaStudija
    {
        get { return p_GodinaStudija;  }
        set
        {
            if ( value > 5 || value < 1)
            {
                throw new Exception("Pogresna godina studija");
            }
            else
            {
                p_GodinaStudija = value;
            }
        }
    }

    public int Ispit (string NazivIspita)
    {
        return 10;
    }

    public int Test(int a)
    {
        return a * 3;
    }
}
```

Nasledjivanje - 06

Izvedena klasa nasledjuje strukturu i ponasanje osnovne (bazne) klase. *(vidi sliku iznad : Osoba)*

Sakrivanje metode

Imamo identicne metode, u izvedenoj klasi imamo istu metodu. Imamo metodu u osnonovoj i izvedenoj klasi koje se zove isto, on je u konfliktu jer ne zna koju da izvrsi. Ako hoces da ti ovo bude ponsanje iz izvedene onda dodaj kljucnu rec **new**.

```
public new void PredstaviSe()
{
        base.PredstaviSe(); // Ovako pozivamo metodu iz osnovne klase.
}
```

Polimorfizam
Omogucava da pozovemo metodu izvedene klase kroz reference osnovne klase.
*OsnovnaKlasa promenjiva = **new** IzvedenaKlasa();*

Generalno se realizuje preko virtuelnih metoda.

```
/*  class Zaposleni{..}
 *  class PartTimeZaposleni : Zaposleni {...}
 *  class TrajnoZaposleni : Zaposleni{...}
    Zaposleni[] nizZaposlenih = new Zaposleni[3];
    nizZaposlenih[0] = new Zaposleni();
    nizZaposlenih[1] = new PartTimeZaposleni();
    nizZaposlenih[2] = new TrajnoZaposleni();
*/
```

Ovde gore smo preko Zaposleni pozvali izvedene PartTimeZaposleni i TrajnoZaposleni. Sve 3 klase se instanciraju prilikom kompajliranja.

Pogledati C# IS/AS , Early-Binding, Late-Binding..

Override - Ponovo definisanje kako nesto radi.

```
public class OsnovnaKlasa
{
    public virtual void Stampa()
    {
        Console.WriteLine("Osnovna Klasa - metoda Stampa()");
    }
}

public class IzvedenaKlasa : OsnovnaKlasa
{
    public override void Stampa()
    {
        Console.WriteLine("Izvedena Klasa - metoda Stampa()");
    }
}

class Program
{
    static void Main(string[] args)
    {
        OsnovnaKlasa ok = new IzvedenaKlasa();
        ok.Stampa();
    }
}
```

Novi cas - 14.03.2018.

Relacioni model primer:
Artikal(SifraArtikla, VrstaArtikla, NazivArtikla, OpisArtikla, KolUSklad)
Italic je spoljni kljuc, podvuceno primary.

MSSQL DataBase > New Database.

Imena tabela na engleskom u mnozini.

Moze i ALTER TABLE Artikal ADD CONSTRAINT PK_ART PRIMARY KEY(SifraArtikla)

SQLQuery1.sql - loc...IGSBJ\Student (51))*

```
CREATE TABLE Artikal
(
    SifraArtikla INT NOT NULL,
    VrstaArtikla NVARCHAR(30) NOT NULL,
    NazivArtikla NVARCHAR(50) NOT NULL,
    OpisArtikla NVARCHAR(50) NOT NULL,
    KolUSklad INT NOT NULL,
    CONSTRAINT PK_ART PRIMARY KEY(SifraArtikla)
)
```

Vrednosti DECIMAL(18,2) NOT NULL // Na dve decimale.

```
CREATE TABLE PoslovniPartner
(
    SifraPoslP INT PRIMARY KEY NOT NULL,
    NazivPoslP NVARCHAR(30) NOT NULL,
    AdresaPoslP NVARCHAR(60) NOT NULL,
    Delatnost NVARCHAR(40) NOT NULL
)

CREATE TABLE Katalog
(
    SifraPoslP INT NOT NULL,
    BrojKataloga INT NOT NULL,
    DatumIzdavanja DATE NOT NULL,
    CONSTRAINT PK_Kat PRIMARY KEY(SifraPoslP, BrojKataloga),
    CONSTRAINT FK_SifraPoslP FOREIGN KEY(SifraPoslP) REFERENCES PoslovniPartner(SifraPoslP),
)
```

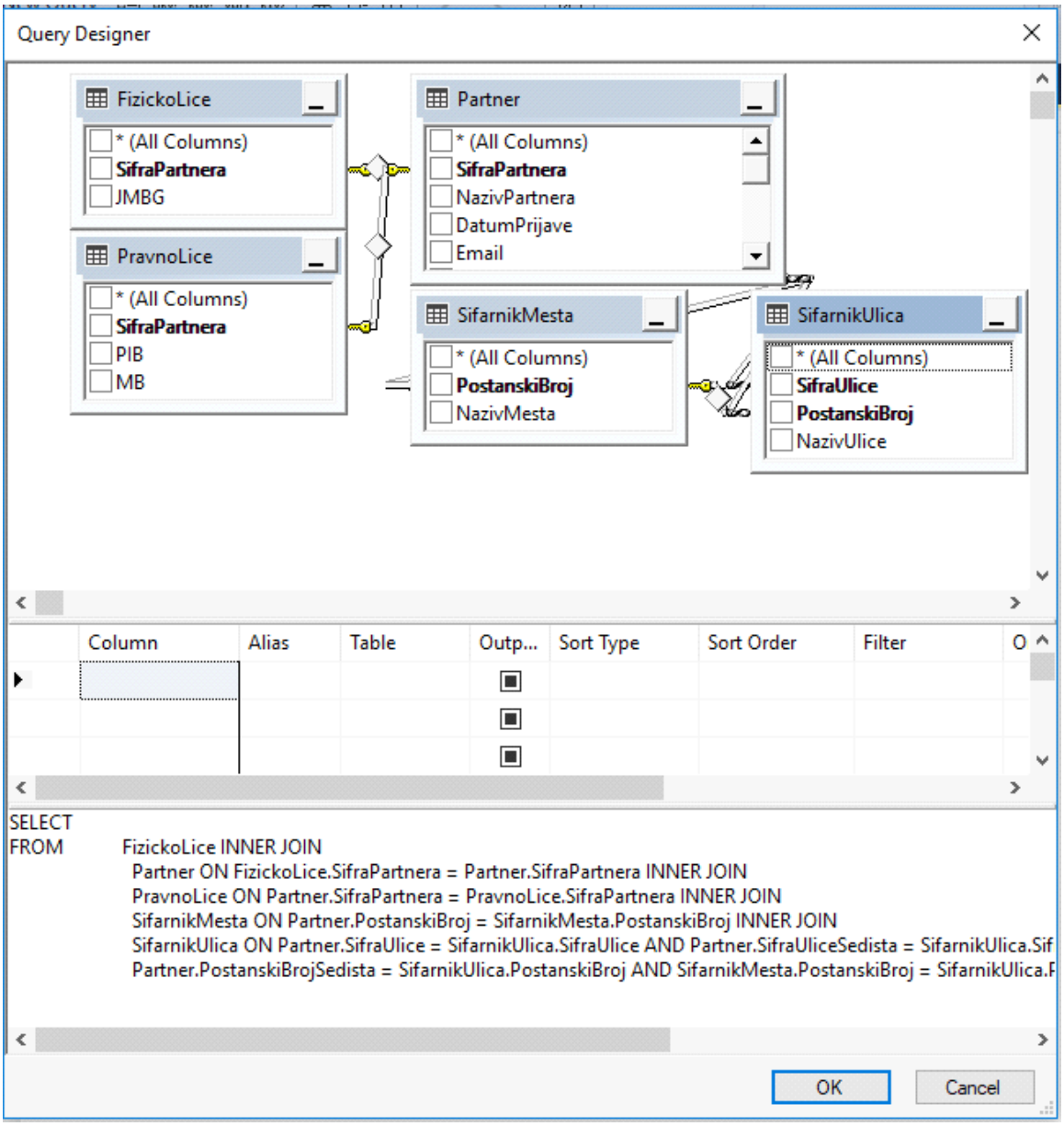
Moze i
CONSTRAINT FK_FAK_OTPR FOREIGN KEY(SifPoslPOtpr, BrojOtpr) REFERENCES Otpremnica(SifPoslPOtpr, BrojOtpr)

Privlaci i primani kljuc fakture (koji obuhvata otpremnicu) i primari kljuc otpremnice, te dva puta isto polje referenciramo constraint-ima (iako jedno polje ima drugaciji naziv da bi SQL server znao sta se desava).

7-8 tabela.

Novi cas - 16.03.18.

New Query > Desni klik > Design query in editor



Gore stikliramo sta zelimo i kliknemo OK.

Umesto pesaka da radimo update svaki put *(UPDATE x SET... WHERE...)*, napravicemo proceduru i triger koji ce da okine kod inserta i izvrsiti proceduru.

Results Messages

	GodinaRacuna	BrojRacuna	RedovnoStorno	DatumKreiranja	DatumDPO	Valuta	IznosRacuna	SifraPartnera	SifraRadnika
1	2017	1	R	2017-01-14	2017-01-16	2017-01-30	0.00	1	2
2	2017	2	R	2017-01-20	2017-01-20	2017-02-03	0.00	2	3
3	2017	3	R	2017-02-01	2017-02-01	2017-02-15	0.00	3	4

	GodinaRacuna	BrojRacuna	RBr	Kolicina	Osnovica	IznosPDV	BrojKataloga	RBrStKataloga
1	2017	1	1	30	1590000.00	318000.00	3	1
2	2017	1	2	50	1600000.00	320000.00	3	2
3	2017	1	3	20	900000.00	180000.00	3	8
4	2017	1	4	10	820000.00	164000.00	3	28
5	2017	1	5	1	610000.00	122000.00	3	29

Procedura proracunava kolika ce nam osnovica biti i koliki PDV.

```
CREATE PROCEDURE dbo.SP_Obracun
/* Ulazni parametri */
@var_godina_racuna INT,
@var_broj_racuna INT,
@var_rbr INT,
@var_kolicina INT,
/* Izlazni parametri */
@var_osnovica DEC(18,2) OUTPUT,
@var_PDV DEC(18,2) OUTPUT
AS
SELECT @var_osnovica = sr.Kolicina * sk.Cena,
       @var_PDV = @var_osnovica * 0.2
FROM dbo.StavkaRacuna sr
INNER JOIN dbo.StavkaKataloga sk ON sr.BrojKataloga = sk.BrojKataloga AND sr.RBrStKataloga = sk.RBr
WHERE sr.GodinaRacuna = @var_godina_racuna AND sr.BrojRacuna = @var_broj_racuna AND sr.RBr = @var_rbr
```

Triger

```
CREATE TRIGGER UnosStavkeFakture
ON dbo.StavkaRacuna
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
DECLARE
    @var_godina_racuna INT,
    @var_broj_racuna INT,
    @var_rbr INT,
    @var_kolicina INT,
    @var_osnovica DEC(18, 2),
    @var_iznos_pdv DEC(18, 2)

    /* Kupi parametre koji su upravo insertovani */
SELECT @var_godina_racuna = GodinaRacuna,
       @var_broj_racuna = BrojRacuna,
       @var_rbr = RBr,
       @var_kolicina = Kolicina
FROM INSERTED;
    /* Racuna IznosRacuna preko procedure */
EXECUTE dbo.SP_Obracun
@var_godina_racuna, @var_broj_racuna, @var_rbr, @var_kolicina, @var_osnovica OUTPUT, @var_iznos_pdv OUTPUT
    /* Update */
UPDATE dbo.StavkaRacuna SET
Osnovica = @var_osnovica,
IznosPDV = @var_iznos_pdv
WHERE GodinaRacuna = @var_godina_racuna AND BrojRacuna = @var_broj_racuna AND RBr = @var_rbr;
END
```

Agregatna funkcija

```
/* AGREGATNA F-JA */

SELECT SUM(sk.Osnovica) AS 'Suma osnovica', SUM(sk.IznosPDV) AS 'Suma PDV-a', SUM(sk.Osnovica + sk.IznosPDV)
  AS 'Celokupan iznos fakture'
FROM StavkaRacuna sk
WHERE sk.GodinaRacuna = 2017 AND sk.BrojRacuna = 2
```

Results | Messages

Suma osnovica	Suma PDV-a	Celokupan iznos fakture
11764500.00	2352900.00	14117400.00

Procedura

```
CREATE PROCEDURE SP_Sumiranje_stavki
@var_godina_racuna AS INT,
@var_broj_racuna AS INT,
@var_iznos_racuna AS DECIMAL(18, 2) OUTPUT

AS
SELECT @var_iznos_racuna = CONVERT(DECIMAL(18, 2),SUM(sr.Osnovica + sr.IznosPDV))
FROM StavkaRacuna sr
WHERE sr.GodinaRacuna = @var_godina_racuna AND sr.BrojRacuna = @var_broj_racuna
```

Novi poboljsan triger

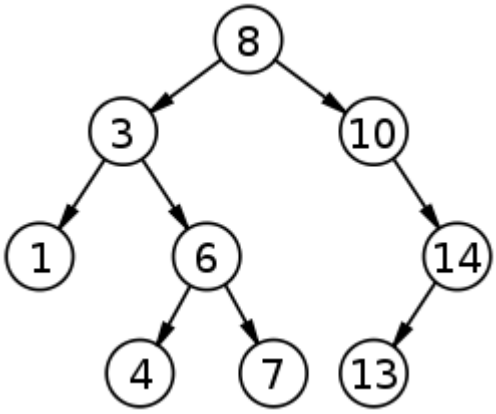
```
CREATE TRIGGER UnosStavkeFakture
ON dbo.StavkaRacuna
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
DECLARE
    @var_godina_racuna INT,
    @var_broj_racuna INT,
    @var_rbr INT,
    @var_kolicina INT,
    @var_osnovica DEC(18, 2),
    @var_iznos_pdv DEC(18, 2),
    @var_iznos_racuna DEC(18, 2)

SELECT @var_godina_racuna = GodinaRacuna,
       @var_broj_racuna = BrojRacuna,
       @var_rbr = RBr,
       @var_kolicina = Kolicina
FROM INSERTED;
EXECUTE dbo.SP_Obracun
@var_godina_racuna, @var_broj_racuna, @var_rbr, @var_kolicina, @var_osnovica OUTPUT, @var_iznos_pdv OUTPUT
UPDATE dbo.StavkaRacuna SET
Osnovica = @var_osnovica,
IznosPDV = @var_iznos_pdv
WHERE
GodinaRacuna = @var_godina_racuna AND BrojRacuna = @var_broj_racuna AND RBr = @var_rbr;


EXECUTE dbo.SP_Sumiranje_stavki
@var_godina_racuna, @var_broj_racuna, @var_iznos_racuna OUTPUT;

UPDATE dbo.Racun SET
Racun.IznosRacuna = @var_iznos_racuna
WHERE Racun.GodinaRacuna = @var_godina_racuna AND Racun.BrojRacuna = @var_broj_racuna;

END
```

View

```
CREATE VIEW dbo.FizickaLicaDetalj
AS
SELECT pa.SifraPartnera, pa.NazivPartnera, fl.JMBG
FROM dbo.Partner pa INNER JOIN dbo.FizickoLice fl ON pa.SifraPartnera = fl.SifraPartnera

SELECT * FROM
dbo.FizickaLicaDetalj
```

ntorka - tuple

Indeks je struktura podataka slicna hash tabeli (ili dictionary), implementirano preko binarnog-stabla.



Novi cas - 17.03.18.

Interfejs moze da sadrzi, svojstva, metode, delegate ili dogadjaje, ali samo deklaracije bez implementacije. Clanovi interfejsa su public po defaultu. Ukoliko klasa ili zapis nasledjuje interfejs, mora da obezbedi implementaciju za sve clanove interfejsa. Klasa ili zapis moze da nasledi vise od jednog interfejsa u isto vreme, ali **klasa ne moze da nasledi vise od jedne klase u isto vreme**. Interfejsi mogu da naslede druge interfejse i klasa koja nasledi interfejs koji je nasledio neki prethodni ona mora da implementira sve.

Razlika izmedju klase i interfejsa? - Cesto pitanje na intervjuima.
Razlika izmedju apstraktne klase i interfejsa?
Apstraktne klase mogu da imaju implementaciju nekih njenih clanova (npr. metoda), ali interfejsi ne mogu da imaju implementaciju za bilo koji od svojih clanova. Pored toga, klasa moze da nasledi vise interfejsa istovremeno, dok ne moze da nasledi vise klasa istovremeno.

Apstratkna klasa je klasa koja je na neki nacin nezavrsena i koja **ne moze da se instancira** (da ima objekat) ali moze da sluzi kao osnovna (base) klasa **koju neka druga moze da nasledi** tj. da bude izvedena. Ima dve vrste clanova, oni koji su apstraktni i oni koji nisu. Znaci, mi mozemo da abstract metode ne diramo, ali ako ih diramo moramo da ih overrideujemo. U apstraktnoj klasi mozemo da imamo i nepastraktne clanove i njih ne moramo da implementiramo u izvedenim.

1. Obezbedi implementaciju zu izvedenoj klasi **za sve apstratkne clanove.**
2. Ukoliko klasa ne obezbedjuje implementaciju za sve apstratkne clanove nasledjenje od apstratkne klase, tada i ta klasa mora da bude apstraktna.

```
public abstract class Kupac
{
        public abstract void Stampa();
}

public class Program : Kupac
{

        public overide void Stampa()
        {
```

```
            Console.WriteLine("Stampa na ekran.");
        }

        static void Main(string[] args)
        {
            Program p = new Program();
            p.Stampa();
        }
}
```

Interfejs, apstraktne klase, nasledjivanje, polimorfizam.

```
7   namespace Kombinacija
8   {
9       // Interfejs.
10      public interface IRacun
11      {
12          void IsplatiSaRacuna(double iznos);
13          void UplatiNaRacun(double iznos);
14          string VratiPodatkeORacunu();
15          double VratiStanje();
16      }
17
18      // * * * PRVA KLASA * * *
19      // Apstraktna klasa koja implementira interfejs.
20      public abstract class ApstraktniRacun : IRacun
21      {
22          private double stanje;
23          private string brojRacuna;
24
25          // Prvi konstruktor.
26          public ApstraktniRacun(string brojRacuna, double pocetnoStanje)
27          {
28              this.brojRacuna = brojRacuna;
29              this.stanje = pocetnoStanje;
30          }
31
32          // Konstruktor koji se poziva pomocu this.
33          public ApstraktniRacun(string brojRacuna) : this(brojRacuna, 0)
34          {
35
36          }
```

```
38          // Metode.
39          public  double VratiStanje()
40          {
41              return stanje;
42          }
43
44          public void IsplatiSaRacuna(double iznos)
45          {
46              stanje -= iznos + ProvizijaNaIsplatu(iznos);
47          }
48
49          public void UplatiNaRacun(double iznos)
50          {
51              stanje += iznos + ProvizijaNaUplatu(iznos);
52          }
53
54          // Dve apstraktne metode koje koristimo iznad.
55          // One ce morati biti implementirane u izvedenim klasama.
56          protected abstract double ProvizijaNaUplatu(double iznos);
57          protected abstract double ProvizijaNaIsplatu(double iznos);
58
59          public virtual string VratiPodatkeORacunu()
60          {
61              string podaci = "Racun broj: " + brojRacuna + "\nIznos na racunu : " + stanje;
62              return podaci;
63          }
64      }
65
```

```
66      // * * * DRUGA KLASA * * *
67      public class TekuciRacun : ApstraktniRacun
68      {
69          // Konsturktor klase TekuciRacun koji prima parametear broj racuna, zove mi
70          // konstruktor bazne klase i prosledjuje mi broj racuna.
71          public TekuciRacun(string brojRacuna) : base(brojRacuna)
72          {
73
74          }
75
76          public TekuciRacun(string brojRacuna, double pocetnoStanje) : base(brojRacuna, pocetnoStanje)
77          {
78
79          }
80
81          // Apstraktne metode iz klase ApstraktniRacun koje moramo da implementiramo.
82          protected override double ProvizijaNaIsplatu(double iznos)
83          {
84              return 0;
85          }
86
87          protected override double ProvizijaNaUplatu(double iznos)
88          {
89              double obracunataProvizija = 100 + (iznos * 3 / 100);
90              return obracunataProvizija;
91          }
92
93          // Override virtuelne metode VratiPodatkeORacunu i kazemo joj da pozove
94          // osnovnu klasu i njenu metodu VratiPodatkeORacunu i nadoda string.
95          public override string VratiPodatkeORacunu()
96          {
97              return base.VratiPodatkeORacunu() + "\nRip racuna : TEKUCI";
98          }
99      }
```

```
101     // * * * TRECA KLASA * * *
102     public class DevizniRacun : ApstraktniRacun
103     {
104         public DevizniRacun(string brojRacuna) : base(brojRacuna) { }
105         public DevizniRacun(string brojRacuna, double pocetnoStanje) : base(brojRacuna, pocetnoStanje) { }
106
107         // Moramo da implementiramo ove metode jer su one apstraktne.
108         protected override double ProvizijaNaUplatu(double iznos)
109         {
110             double obracunataProvizija = 100 + (iznos * 5 / 100);
111             return obracunataProvizija;
112         }
113         protected override double ProvizijaNaIsplatu(double iznos)
114         {
115             double obracunataProvizija = 100 + (iznos * 5 / 100);
116             return obracunataProvizija;
117         }
118
119         public override string VratiPodatkeORacunu()
120         {
121             return base.VratiPodatkeORacunu() + "\nTip racuna : DEVIZNI";
122         }
123     }
```

```
125     class Program
126     {
127         // * * * MAIN * * *
128         static void Main(string[] args)
129         {
130             // Napravimo niz racuna i polimorfno ubacujemo izvedene klase sa razl. konstruktorima.
131             // TekuciRacun i DevizniRacun nasledjuju ApstraktniRacun.
132             ApstraktniRacun[] racuni = new ApstraktniRacun[4];
133
134             racuni[0] = new TekuciRacun("123-456", 0);
135             racuni[1] = new DevizniRacun("287-142", 0);
136             racuni[2] = new DevizniRacun("842-654", 1000);
137             racuni[3] = new TekuciRacun("842-654", 25000);
138
139             // Testiranje.
140             for (int i = 0; i < 4; i++)
141             {
142                 Console.WriteLine("Uplata na " + i + ". racun u iznosu od 1000");
143                 racuni[i].UplatiNaRacun(1000);
144                 Console.WriteLine("Isplata na " + i + ". racun u iznosu od 500");
145                 racuni[i].IsplatiSaRacuna(500);
146                 Console.WriteLine("Uplata na " + i + ". racun u iznosu od 3000");
147                 racuni[i].UplatiNaRacun(3000);
148                 Console.WriteLine("Isplata na " + i + ". racun u iznosu od 1500");
149                 racuni[i].IsplatiSaRacuna(1500);
150                 Console.WriteLine("Podaci o racunu posle uplate: \n" + racuni[i].VratiPodatkeORacunu());
151                 Console.WriteLine();
152             }
153             Console.ReadKey();
154         }
155     }
```

Delegat je obejkat koji sadrzi referencu na metodu,. Pokazuje na metodu tj. funkciju i kada vi pozovete ovog delegata to znaci da ce funkcija biti pozvana. Zasto indirektno pozivamo metodu preko delegata, a ne direktno? Zbog fleksibilnosti.

Delegat se uvek definise na nivou namespace-a (odmah ispod namespace)

```
7   namespace ConsoleApp6
8   {
9       // Deklaracija delegata.
10      public delegate void DelegatZaIspisivanje(string poruka);
11
12      class Program
13      {
14          static void Main(string[] args)
15          {
16
17              // Preko delegata mu prosledjujem parametar "poruka".
18              // Delegat moze da pozove nekoliko metoda. Taj niz metoda koji on poziva metod invocation lista.
19              // U tu listu smo stavili metodu ispisivanje.
20              // Kada napravimo njegovu instancu on poziva tu MIL.
21              // Ispisivanju prosledi poruku iz delegata.
22              DelegatZaIspisivanje del = new DelegatZaIspisivanje(Ispisivanje);
23              del("Poruka iz delegata");
24              // Delegate je type-safe pointer na metodu.
25              // Type-safe - siguran u tip koji ce se proslediti / koji ce se vratiti.
26              // U sutini delegate je metoda koja samo ima kljucnu rec delegate.
27          }
28
29          public static void Ispisivanje(string poruka)
30          {
31              Console.WriteLine(poruka);
32          }
33      }
34  }
```

Get; set;
public string Name{ get; set; } je isto sto i :

It's a so-called auto property, and is essentially a shorthand for the following (similar code will be generated by the compiler):

```
private string name;
public string Name
{
    get
    {
        return this.name;
    }
    set
    {
        this.name = value;
    }
}
```

Sta je lose couple - lako promenjivo, lako povezano.
Delegati i dogadjaji nece biti presudni za ispit, ali trebalo bi ih znati.

Klasa <naziv> = new <Konstruktor>;
Apstrakcija - zanemirivanje detalja. Na jednom sloju resimo jedan problem a ignorisemo ostale, na nekom drugom sloju resimo neki od drugih problema.

02. OSI OSI i TCP IP referentni modeli.

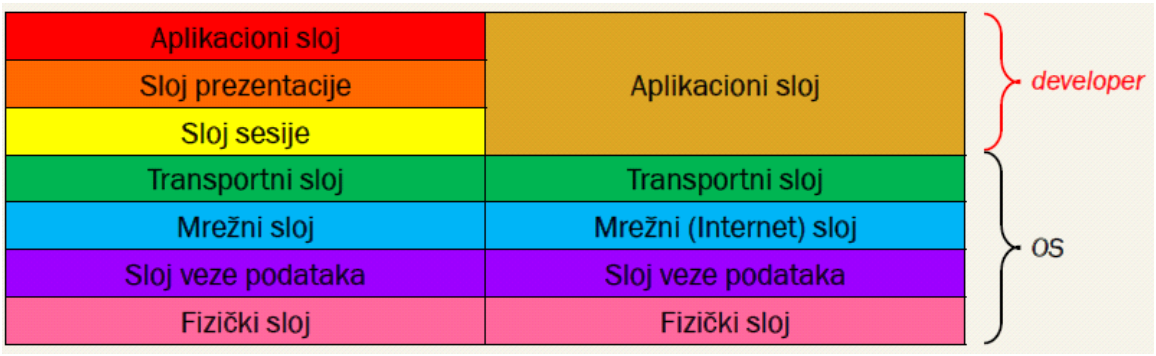
Protokol - skup pravila koja definisu komunikaciju.

OSI model deli komunikacioni sistem na 7 slojeva. Svaki sloj nudi uslugu/servis sloju iznad, a koristi uslugu sloja ispod sebe tj. oslanja se na sloj ispod.

OSI model		
Sloj (eng. layer)	PDU – Protocol Data Unit	Funkcija
Aplikacioni sloj	Podaci (eng. data)	API visokog nivoa, deljenje resursa, udaljeni pristup fajlovima
Sloj prezentacije		prevođenje podataka između mrežnih servisa i aplikacije, uključujući enkodiranje karaktera, kompresiju podataka, kriprovanje / dekriptovanje
Sloj sesije		upravljanje komunikacionim sesijama, npr. kontinualna razmena informacija u formi višestrukih transmisija između dva čvora
Transportni sloj	Segment (TCP) ili datagram (UDP)	pouzdana transmisija segmenata između tačaka na mreži uključujući segmentaciju, potvrđivanje i multipleksiranje
Mrežni sloj	Paket (eng. packet)	strukturiranje i upravljanje mrežom od više čvorova, uključujući adresiranje, rutiranje i kontrolu saobraćaja
Sloj veze podataka	Okvir (eng. frame)	pouzdan prenos okvira između 2 čvora povezanih fizičkim slojem
Fizički sloj	Bit (eng. bit)	transmisija i prihvatanje stream-ova bitova preko fizičkog medijuma

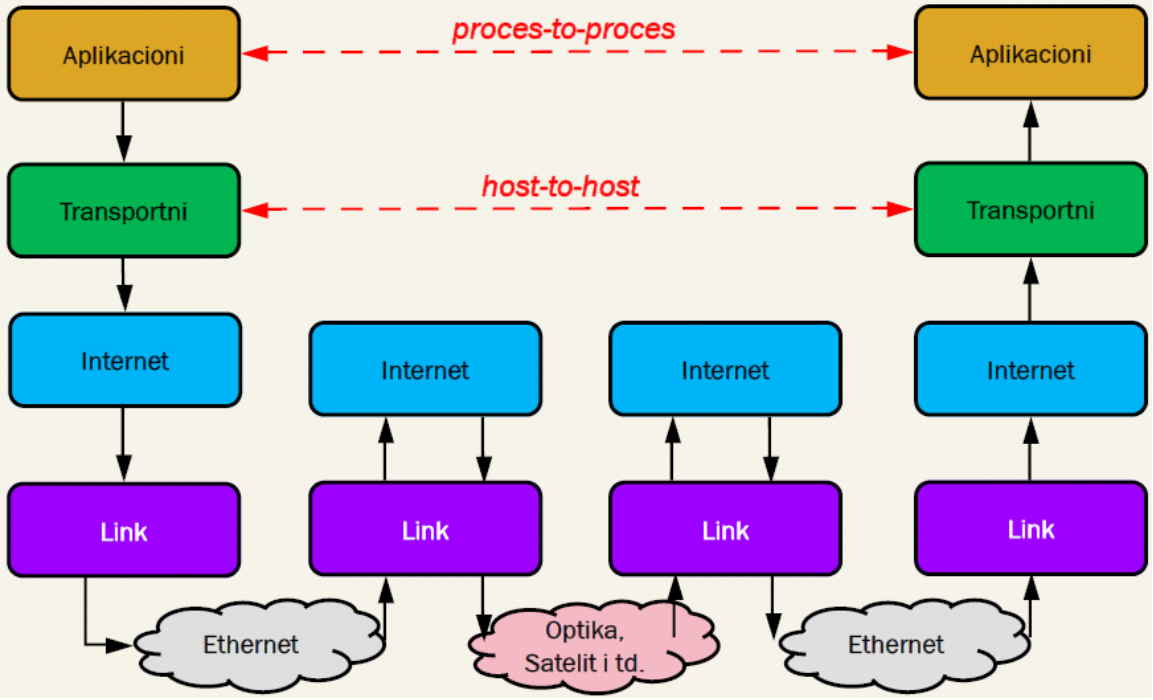
Odogvor HTML stranice, na njoj uoci 3 img src-a i posalje jos 3 requesta iza haube.

TCP/IP je protokol stek (skup protokola) ili internet model koji je takodje konceptualni model i skup k. protokola koji se koriste na Internetu. 2 najznacajnija protokola:
TCP (Transimitition Control Protocol) - odgovara transportnom sloju OSI-a
IP (Internet Protocol) - odgovara mreznom sloju OSI-a



Slojevi	Protokoli	PDU
Aplikacioni	HTTP, SMTP, FTP, DNS, IMAP, POP, LDAP, TELNET, SSH, SNMP	Poruka
Transportni	TCP, UDP, ...	Segment
Mrežni	IP (IPv4, IPv6), ICMP, ...	Datagram
Sloj veze podataka	ETHERNET, PPP	Okvir
Fizički sloj	/	Bitovi

Web server i browser - 2 procesa na 2 odvojene masine.
Aplikacioni sloj - Razmena podatka proces-ka-proces na udaljenim masinama.
Transportni sloj komunikaciju sa racunar-na-racunar (host to host)



Link je sloj veze

Port je krajnja tacka komunikacije u operstivnom sistemu. On je uvek povezan sa IP adresom host-a. On kompletira mreznu adresu destinacije ili polazista poruke.

Interprocesna komunikacija - komunikacija izmedju 2 procesa na istom sistemu. **Socket** je softverski interfejs, posrednik izmedju aplikativnog i transportnog sloja racunara.

Novi cas - Sve obrisano zbog debilnog kompa 23.03.18.

SQLConnection, Command i DataReader - 3 klase bitne za rad.

Sto mora da se zatvori konekcija? Zbog preopterecenja servera. Zatvorili bi konekciju sa using ili try catch finally blockom.

Razlika izmedju rada sa konekcijom i bez konekcijom - Sa konekcijom otvorimo konekciju, odradimo naredbe, pa zatvorimo. A ovde otvorimo konekciju, izvrsi se upit, konekcija se zatvori, obrade se podaci pa se konekcija ponovo otvori.

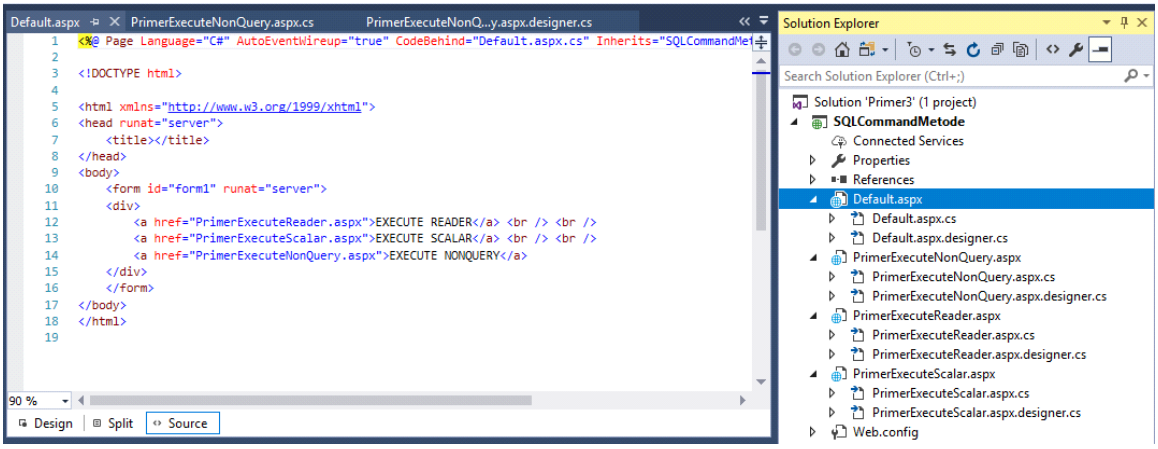
Kako da ne pravimo 100 connection stringa? Napravimo u konfiguracionom fajlu preko XML-a konekcioni string i onda ga preko Configurationmanager klase pozivamo i referenciramo na njegov name.

```
string konekcioniString =
ConfigurationManager.ConnectionStrings["WinFormKonekcija"].ConnectionString;

using (SqlConnection konekcija = new SqlConnection(konekcioniString))
{
        SqlCommanda komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
        konekcija.Open();
        SqlDataReader rider = komanda.ExecuteReader();
        BindingSource sors = new BindingSource();
        sors.DataSource = rider;
        dataGridView1.DataSource = sors;
        // Ne mora konekcija.Close() jer je unutar using bloka.
}
```

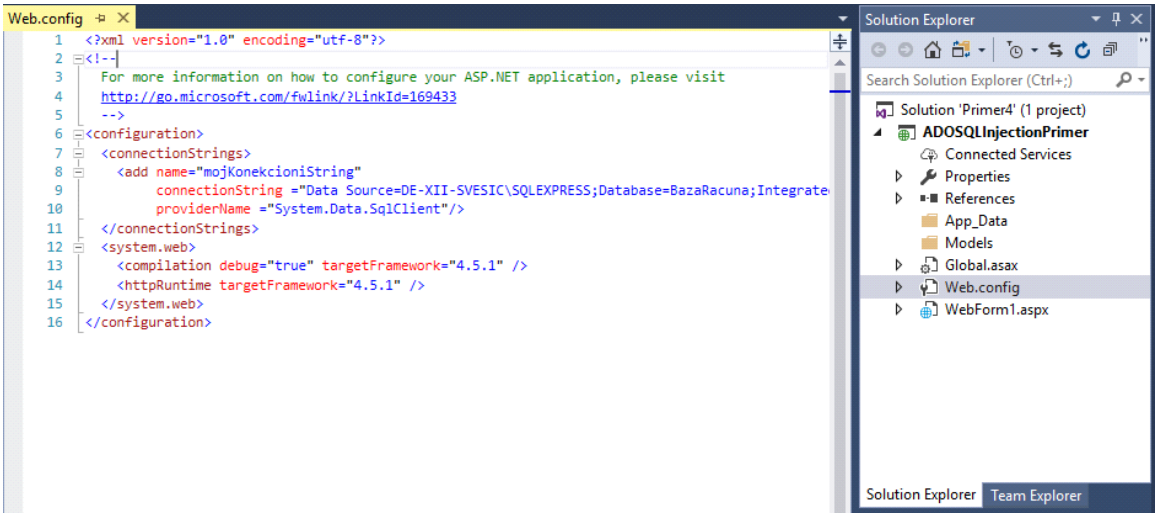
Umesto using moze i try catch **finally** da osiguramo da se zatvori vezu.

Using moze da se koristi za namespace i za osiguravanje da ce da se konekcija zatvori na kraju.

Novi cas - 28.03.18.



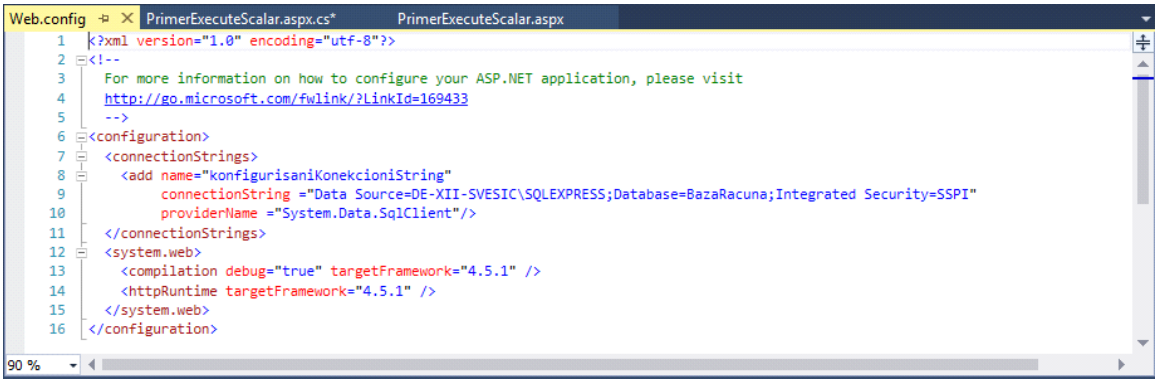
U konfiguracionom fajlu napravimo connectionString u XML-u kako ne bismo pravili 10 razlicitih konkecionih stringova.



SqlCommand priprema SQL naredbu koju zelimo da izvrsimo na SQL serveru.

ExecuteReader - Kada SQL naredba vraca vise vrednosti, tj. ukoliko upit vrati nekoliko redova podataka.
ExecuteNonQuery - Koristi se kada zelimo da uradimo Insert, Upadte ili Delete naredbe.
ExecuteScalar - Koristi se kada upit vraca jednu (skalarnu) vrednost, npr. ukoliko upit vraca proscean iznos plate zaposlenih.

KonekcioniString



1. Konekcioni string.
2. Pravljenje konekcije.
3. SqlCommand
4. Otvori konekciju.
5. Izvrsavanje naredbe.
6. Zatvaranje konekcije. (osim ako ne koristimo finally ili using)

ExecuteScalar - kada se vraca samo jedna vrednost. Vraca tip object koji menjamo.

PrimerExecuteScalar.aspx.cs* | PrimerExecuteScalar.aspx

SQLCommandMetode | SQLCommandMetode.PrimerExecuteScalar | Page_Load(object sender, EventArgs e)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

namespace SQLCommandMetode
{
    public partial class PrimerExecuteScalar : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // KonekcioniString iz konfiguracionog XML fajla.
            string konekcioniString = ConfigurationManager.ConnectionStrings["konfigurisaniKonekcioniString"].ConnectionString;

            using (SqlConnection konekcija = new SqlConnection(konekcioniString))
            {
                SqlCommand komanda = new SqlCommand();
                komanda.CommandText = "SELECT COUNT(SifraOdeljenja) FROM SifarnikOdeljenja";
                komanda.Connection = konekcija;
                konekcija.Open();
                // ExecuteScalar vrsimo gde smo sigurno da nam se vraca JEDNA vrednost - agregatne funkcije.
                // ExecuteScalar vraca object, mi znamo da je to funkcija koji broji redove, pa ga menjamo u int.
                int ukupanBrojRedova = (int)komanda.ExecuteScalar();
                // Ovo je kao MessageBox.Show.
                Response.Write("Ukupan broj redova je " + ukupanBrojRedova.ToString());
            }
        }
    }
}
```

90 %

ExecuteNonQuery - Uvek vraca *how many rows affected.* Zato ga ubacujemo u int.

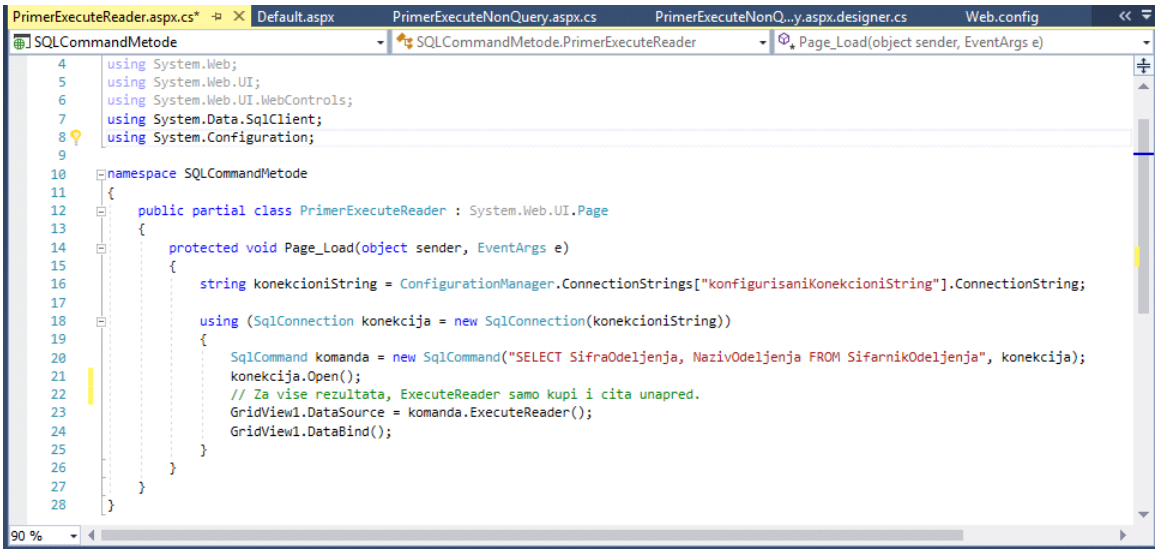
PrimerExecuteNonQuery.aspx.cs | PrimerExecuteNonQ...y.aspx.designer.cs | Web.config | PrimerExecuteScalar.aspx.cs* | PrimerExecuteScalar.aspx

SQLCommandMetode | SQLCommandMetode.ExecuteNonQuery | Page_Load(object sender, EventArgs e)

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;

namespace SQLCommandMetode
{
    public partial class ExecuteNonQuery : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string konekcioniString = ConfigurationManager.ConnectionStrings["konfigurisaniKonekcioniString"].ConnectionString;

            using (SqlConnection konekcija = new SqlConnection(konekcioniString))
            {
                SqlCommand komanda = new SqlCommand();
                komanda.CommandText = "INSERT INTO SifarnikOdeljenja VALUES (14000, 'Operativa')";
                komanda.Connection = konekcija;
                konekcija.Open();
                int brojUbacenihRedova = komanda.ExecuteNonQuery();
                Response.Write("Ukupan broj ubacenih redova " + brojUbacenihRedova.ToString() + "<br />" + "<br />");

                //SqlCommand komanda = new SqlCommand();
                //komanda.CommandText = "UPDATE SifarnikOdeljenja SET NazivOdeljenja = 'Odeljenje ugovoranja' WHERE SifraOdeljenja = 14000";
                //komanda.Connection = konekcija;
                //konekcija.Open();
                //int brojIzmenjenihRedova = komanda.ExecuteNonQuery();
                //Response.Write("Ukupan broj izmenjenih redova " + brojIzmenjenihRedova.ToString() + "<br />" + "<br />");

                //SqlCommand komanda = new SqlCommand();
                //komanda.CommandText = "DELETE SifarnikOdeljenja WHERE SifraOdeljenja = 14000";
                //komanda.Connection = konekcija;
                //konekcija.Open();
                //int brojObrisanihOdeljenja = komanda.ExecuteNonQuery();
                //Response.Write("Ukupan broj obrisanih redova " + brojObrisanihOdeljenja.ToString());
            }
        }
    }
}
```

90 %

ExecuteReader - Kupi rezultate i samo cita unapred.



SQL inekcija, stored procedure...

Videti primer 4.

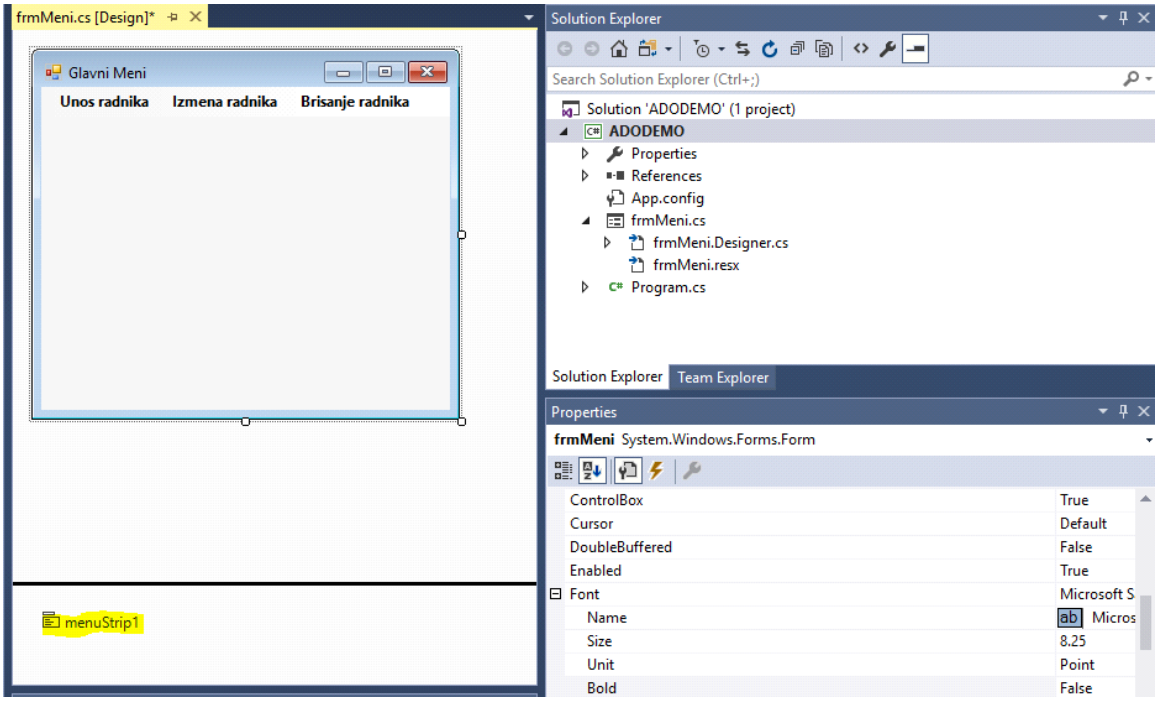
Stored procedure
komanda.CommandType = System.Data.CommandType.StoredProcedure
komanda.Parameters.AddWithValue("@naziv", TextBox1.Text);

Uglavnom koristimo AddWithValue. Za SQL inekciju ili parametrizovan upit ili stored procedure za domaci.

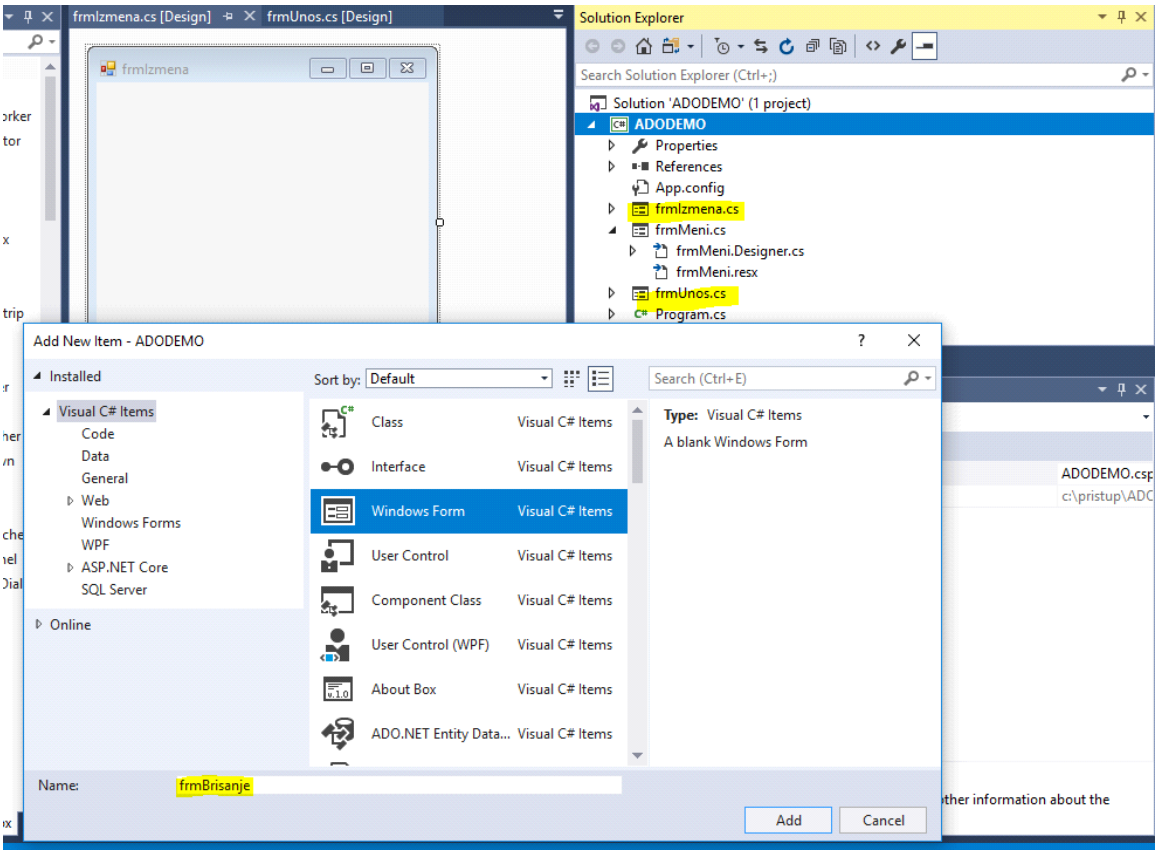
Novi cas - 30.03.18.

1. New > Project > WinForm App.

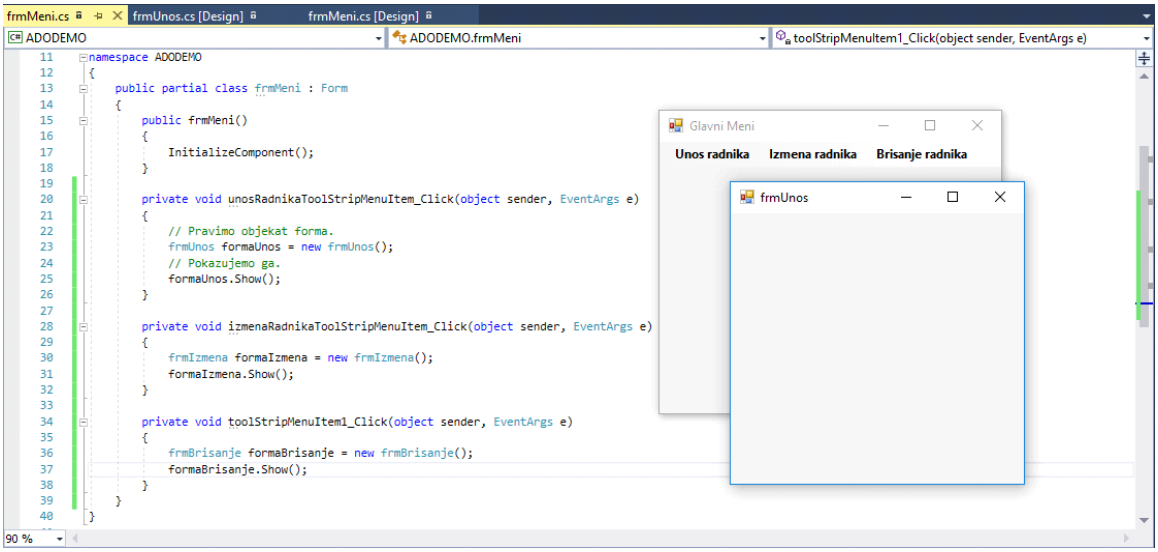
2. Dodamo menustrip i 3 stavke na njemu.



3. Desni klik na project > Add > New item > Windows Form. Dodamo 3 forme.



4. Desni klik na frmMeni > View code. Dupli klik na svaki od meni itema kako bi dobili eventove u frmMeni.

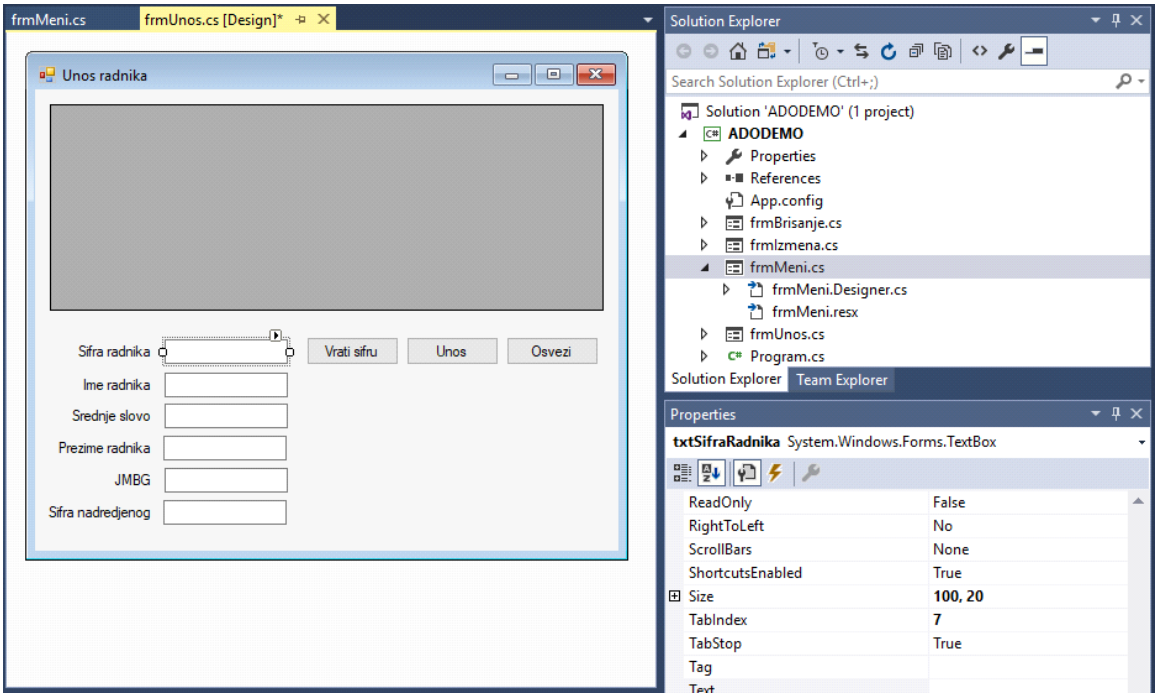


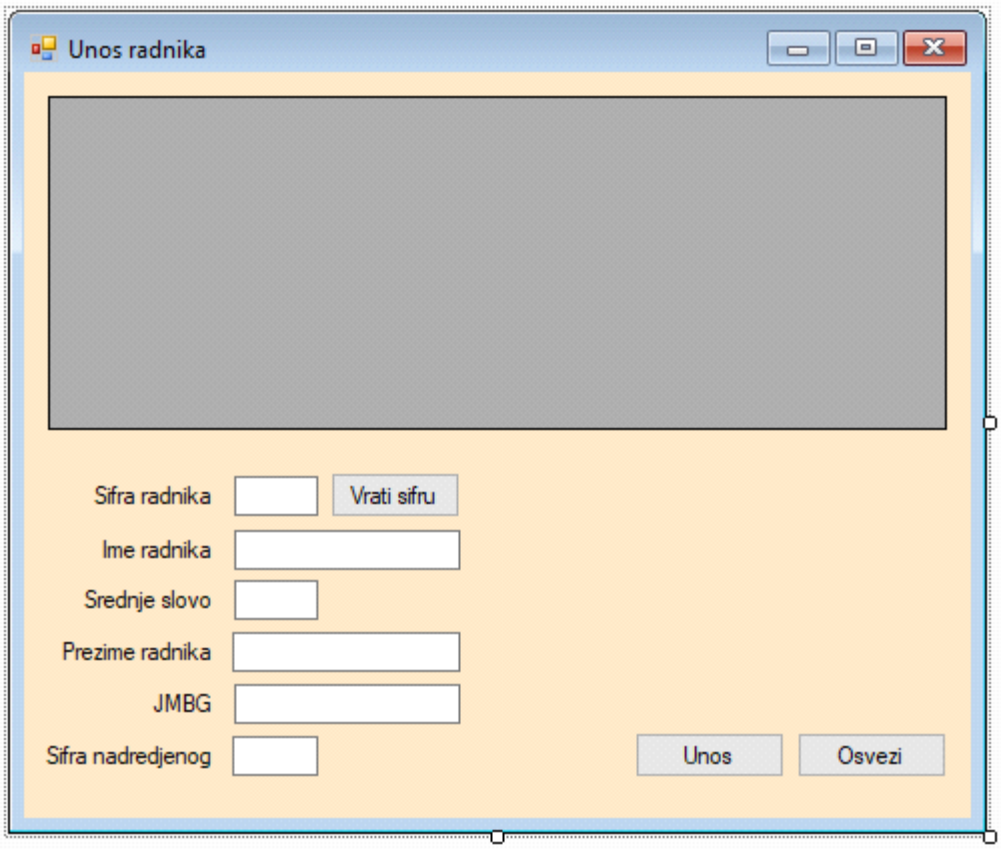

```
namespace ADODEMO
{
    public partial class frmMeni : Form
    {
        public frmMeni()
        {
            InitializeComponent();
        }

        private void unosRadnikaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Pravimo objekat forma.
            frmUnos formaUnos = new frmUnos();
            // Pokazujemo ga.
            formaUnos.Show();
        }

        private void izmenaRadnikaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            frmIzmena formaIzmena = new frmIzmena();
            formaIzmena.Show();
        }

        private void toolStripMenuItem1_Click(object sender, EventArgs e)
        {
            frmBrisanje formaBrisanje = new frmBrisanje();
            formaBrisanje.Show();
        }
    }
}
```

Moze i forma.ShowDialog();

Forma unos

1. Ubacujemo prazan DataGridView.
2. Dodajemo button-e, labele i textbox-ove. Preimenujemo Textbox1 u txtSifraRadnika npr., button1 u btnUnos i sl.






```
public partial class frmUnos : Form
{
    // Definisemo par promenjivih.
    string konekcioniString = "";
    SqlConnection konekcija;
    SqlCommand komanda;

    public frmUnos()
    {
        InitializeComponent();
        konekcioniString = "server=.;integrated security=true;database=BazaRacuna";
    }

    private void frmUnos_Load(object sender, EventArgs e)
    {
        // Kad nam se ucita forma da se izvrsi funkcija.
        OsveziEkran();
    }

    private void OsveziEkran()
    {
        using (konekcija = new SqlConnection(konekcioniString))
        {
            komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
            konekcija.Open();
            SqlDataReader reader = komanda.ExecuteReader();
```

```
frmUnos.cs*  frmMeni.cs  frmUnos.cs [Design]*
ADODEMO  ADODEMO.frmUnos  frmUnos_Load(object sender, EventArgs e)
 9  using System.Windows.Forms;
10  // Dodajemo SqlClient.
11  using System.Data.SqlClient;
12  namespace ADODEMO
13  {
14      public partial class frmUnos : Form
15      {
16          // Definisemo par promenjivih.
17          string konekcioniString = "";
18          SqlConnection konekcija;
19          SqlCommand komanda;
20
21          public frmUnos()
22          {
23              InitializeComponent();
24              // Data Source nadjemo desnim klikom > properties u MSSQL.
25              konekcioniString = "Data Source=DESKTOP-EMIGSBJ\\Student;Database=BazaRacuna;Integrated Security=SSPI";
26          }
27
28          private void frmUnos_Load(object sender, EventArgs e)
29          {
30              // Kad nam se ucita forma da se izvrsi funkcija.
31              OsveziEkran();
32          }
33
34          private void OsveziEkran()
35          {
36              using (konekcija = new SqlConnection(konekcioniString))
37              {
38                  komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
39                  konekcija.Open();
40                  SqlDataReader reader = komanda.ExecuteReader();
41
42                  // Da li ovo sto smo procitali ima redova.
43                  if (reader.HasRows)
44                  {
45                      // Ako ima, pravi DataTable i u nju load-uj reader tj. ono sto si procitao.
46                      DataTable tabela = new DataTable();
47                      tabela.Load(reader);
48                      // Ubacujemo tabelu iz memorije u grid.
49                      dataGridView1.DataSource = tabela;
50                  }
51              }
52          }
82 %
```

Dugme vrati sifru - Vraca prvu slobodnu sifru.

```
frmMeni.cs  frmUnos.cs  frmUnos.cs [Design]
ADODEMO  ADODEMO.frmUnos  OsveziEkran()
31              OsveziEkran();
32          }
33
34          private void OsveziEkran()
35          {
36              using (konekcija = new SqlConnection(konekcioniString))
37              {
38                  komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
39                  konekcija.Open();
40                  SqlDataReader reader = komanda.ExecuteReader();
41
42                  // Da li ovo sto smo procitali ima redova.
43                  if (reader.HasRows)
44                  {
45                      // Ako ima, pravi DataTable i u nju load-uj reader tj. ono sto si procitao.
46                      DataTable tabela = new DataTable();
47                      tabela.Load(reader);
48                      // Ubacujemo tabelu iz memorije u grid.
49                      dataGridView1.DataSource = tabela;
50                  }
51              }
52          }
53
54          private void btnVratiSifru_Click(object sender, EventArgs e)
55          {
56              using (konekcija = new SqlConnection(konekcioniString))
57              {
58                  komanda = new SqlCommand("SELECT COUNT(*) FROM Radnik", konekcija);
59                  konekcija.Open();
60                  int sifra = Convert.ToInt32(komanda.ExecuteScalar());
61                  sifra++;
62                  txtSifraRadnika.Text = sifra.ToString();
63              }
64          }
65      }
66  }
67
90 %
```

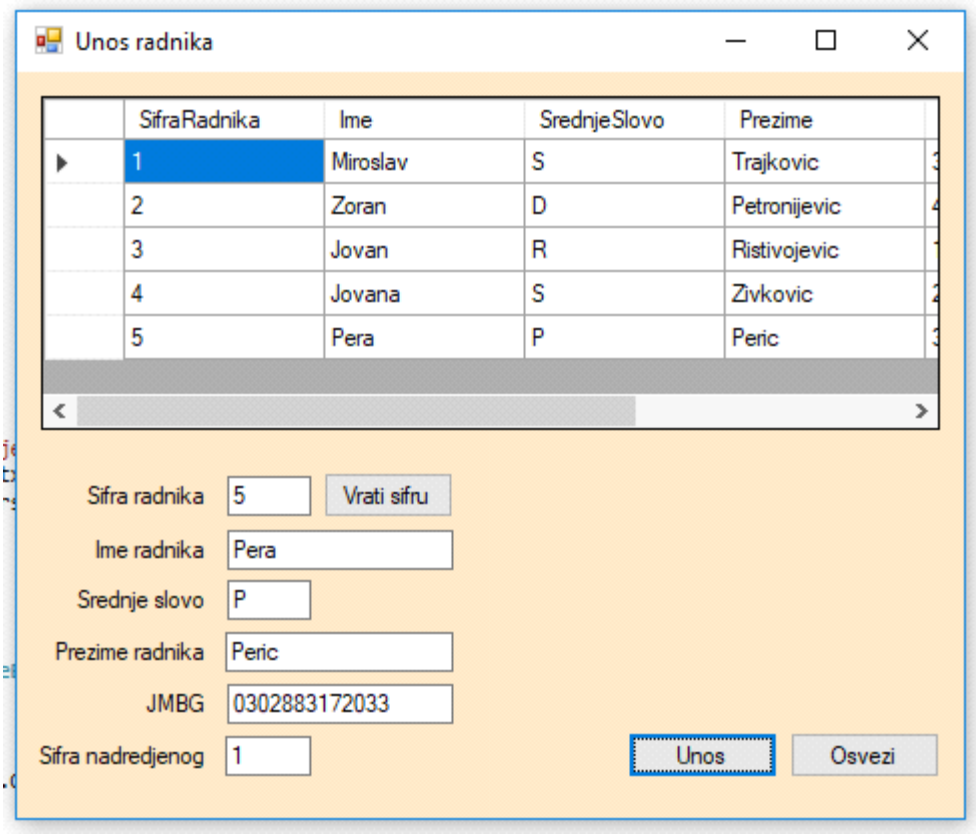
Unos radnika

	SifraRadnika	Ime	SrednjeSlovo	Prezime
▶	1	Miroslav	S	Trajkovic
	2	Zoran	D	Petronijevic
	3	Jovan	R	Ristivojevic
	4	Jovana	S	Zivkovic

Sifra radnika 5 Vrati sifru

Unos radnika

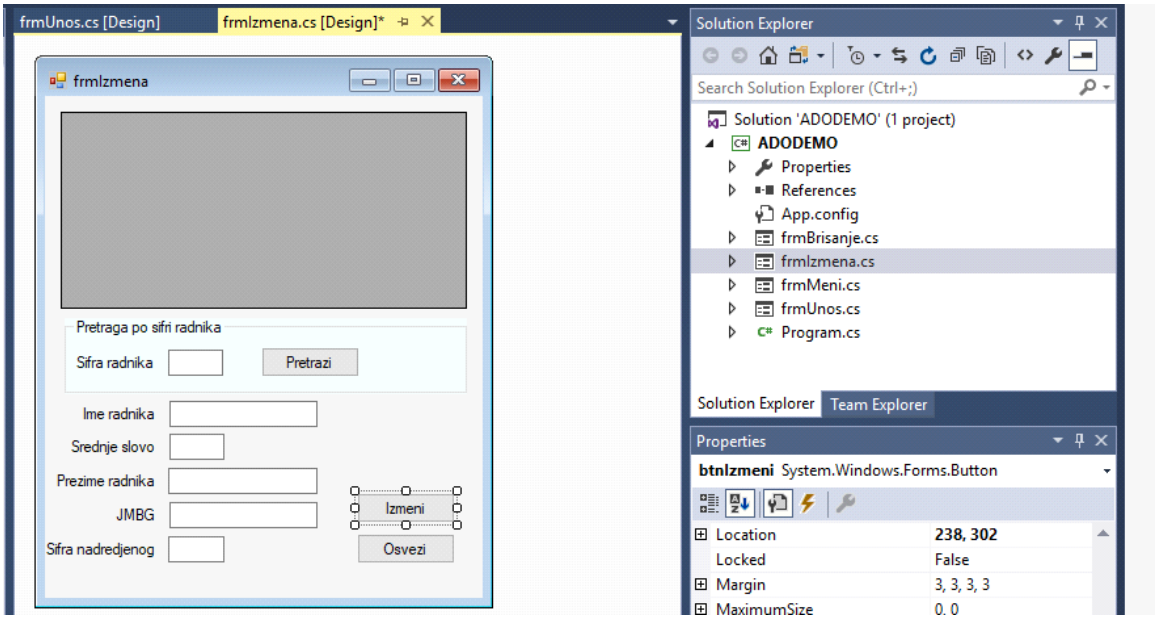
```
private void btnUnos_Click(object sender, EventArgs e)
{
    using (konekcija = new SqlConnection(konekcioniString))
    {
        // ' kod stringova.
        komanda = new SqlCommand("INSERT INTO Radnik(SifraRadnika, Ime, SrednjeSlovo, Prezime, JMBG, SifraNadredjenog) VALUES("
                                 + int.Parse(txtSifraRadnika.Text) + ",'" +txtImeRadnika.Text + "','" + txtSrednjeSlovo.Text +"','"
                                 + txtPrezimeRadnika.Text + "'," + Int64.Parse(txtJMBG.Text) +","+ int.Parse(txtSifraNadredjenog.Text) + ")"
                                 ,konekcija);
        konekcija.Open();
        try
        {
            komanda.ExecuteNonQuery();
            MessageBox.Show("Radnik je uspesno dodat.", "Informacija", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch(Exception)
        {
            MessageBox.Show("Nastala je greska.", "Greska", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        finally
        {
            OsveziEkran(); // Funkcija iznad koja radi SELECT *.
            konekcija.Close();
        }
    }
}
```



Osvezi ekran - cisti sve textbox-ove.

```
private void btnOsvezi_Click(object sender, EventArgs e)
{
    // Ciscenje textbox-eva.
    txtImeRadnika.Clear();
    txtImeRadnika.Clear();
    txtSrednjeSlovo.Clear();
    txtPrezimeRadnika.Clear();
    txtJMBG.Clear();
    txtSifraNadredjenog.Clear();

    // Fokus na sifra radnika textbox.
    txtSifraRadnika.Focus();

    // Da osvezi gridview sa najnovijim SELECT *.
    OsveziEkran();
}
```

Forma izmeni



Isto kao i na prvoj formi, na pocetku using System.Data.SqlClient i konekcioni string, SqlConnection i SqlCommand. Koristimo istu OsveziEkran funkciju koja ce se izvrsiti nakon load-a forme.

```
frmIzmena.cs*  frmUnos.cs  frmIzmena.cs [Design]*
ADODEMO  ADODEMO.frmIzmena  OsveziEkran()
```

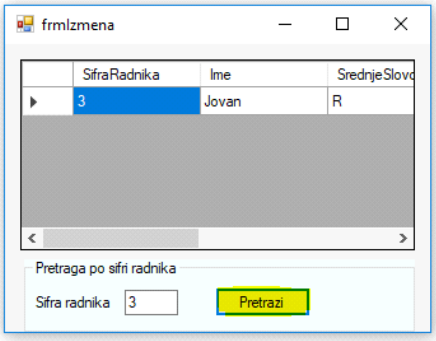
```
9   using System.Windows.Forms;
10  using System.Data.SqlClient;
11
12  namespace ADODEMO
13  {
14      public partial class frmIzmena : Form
15      {
16          string konekcioniString = "";
17          SqlConnection konekcija;
18          SqlCommand komanda;
19
20          public frmIzmena()
21          {
22              InitializeComponent();
23              konekcioniString = "server=.;integrated security=true;database=BazaRacuna";
24          }
25
26          private void OsveziEkran()
27          {
28              using (konekcija = new SqlConnection(konekcioniString))
29              { // 1. Komanda; 2. Otvaranje; 3. Reader; 4. DataTable; 5. .Load i Grid.
30                  komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
31                  konekcija.Open();
32                  SqlDataReader reader = komanda.ExecuteReader();
33
34                  if (reader.HasRows)
35                  {
36                      DataTable tabela = new DataTable();
37                      tabela.Load(reader);
38                      dataGridView1.DataSource = tabela;
39                  }
40              }
```

82 %

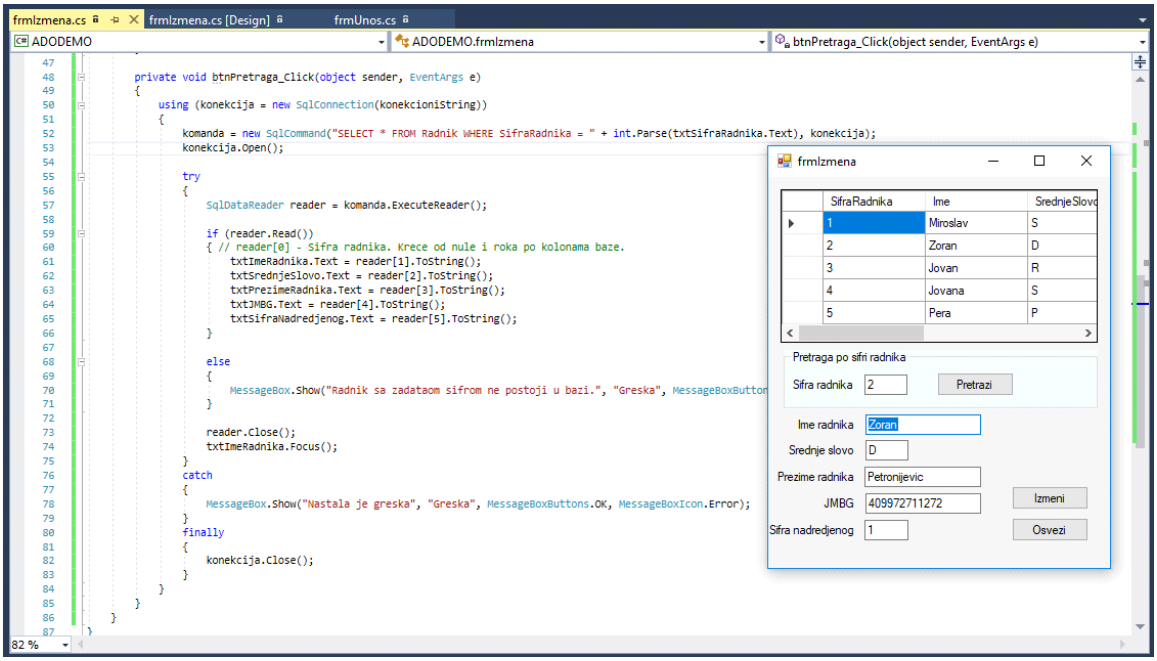
```
private void frmIzmena_Load(object sender, EventArgs e)
{
    OsveziEkran();
}
```

Pretraga radnika po sifri

```
48          private void btnPretraga_Click(object sender, EventArgs e)
49          {
50              using (konekcija = new SqlConnection(konekcioniString))
51              {
52                  komanda = new SqlCommand("SELECT * FROM Radnik WHERE SifraRadnika = " + int.Parse(txtSifraRadnika.Text), konekcija);
53                  konekcija.Open();
54                  SqlDataReader reader = komanda.ExecuteReader();
55
56                  if (reader.HasRows)
57                  {
58                      DataTable tabela = new DataTable();
59                      tabela.Load(reader);
60                      dataGridView1.DataSource = tabela;
61                  }
62              }
63          }
64      }
65  }
66
```



Preradjeno da sam popuni polja na formi.



Izmena podataka

Koristimo parametrizovan upit.

```
        private void btnIzmeni_Click(object sender, EventArgs e)
        {
            konekcija = new SqlConnection(konekcioniString);

            // Radimo sa parametrizovanim upitima.
            string tekstKomande = "UPDATE Radnik SET Ime = @var_ime_radnika, " +
                                                    "SrednjeSlovo = @var_srednje_slovo, " +
                                                    "Prezime = @var_prezime_radnika, " +
                                                    "JMBG = @var_jmbg, " +
                                                    "SifraNadredjenog = @var_sifra_nadredjenog" +
                                                    "WHERE SifraRadnika = @Var_sifra_radnika";
            komanda = new SqlCommand(tekstKomande, konekcija);

            // Prvo Parameters.Add, pa onda tom dodamo dodelimo .Value.
            komanda.Parameters.Add("@var_sifra_radnika", SqlDbType.Int);
            komanda.Parameters["@var_sifra_radnika"].Value = Convert.ToInt32(txtSifraRadnika.Text);

            komanda.Parameters.Add("@var_ime_radnika", SqlDbType.NVarChar);
            komanda.Parameters["@var_ime_radnika"].Value = txtImeRadnika.Text;

            komanda.Parameters.Add("@var_srednje_slovo", SqlDbType.NVarChar);
            komanda.Parameters["@var_srednje_slovo"].Value = txtSrednjeSlovo.Text;

            komanda.Parameters.Add("@var_prezime_radnika", SqlDbType.NVarChar);
            komanda.Parameters["@var_prezime_radnika"].Value = txtPrezimeRadnika.Text;

            komanda.Parameters.Add("@var_jmbg", SqlDbType.BigInt);
            komanda.Parameters["@var_jmbg"].Value = Convert.ToInt64(txtJMBG.Text);

            komanda.Parameters.Add("@var_sifra_nadredjenog", SqlDbType.Int);
            komanda.Parameters["@var_sifra_nadredjenog"].Value = Convert.ToInt32(txtSifraNadredjenog.Text);
```

```
118
119             try
120             {
121                 konekcija.Open();
122                 int redovi = komanda.ExecuteNonQuery(); // ExecuteNonQuery vraca how many affected rows.
123                 MessageBox.Show("Uspesna izmena.", "Informacija", MessageBoxButtons.OK, MessageBoxIcon.Information);
124             }
125             catch(Exception)
126             {
127                 MessageBox.Show("Nastala je greska", "Greska", MessageBoxButtons.OK, MessageBoxIcon.Error);
128             }
129             finally
130             {
131                 konekcija.Close();
132             }
133         }
```

31.03.18. - Novi cas

Korisnik moze da nesto ima, zna, i nesto sto jeste. 2 factor authentification koristi 2 faktora odavde (npr. mobilni imamo i znamo nas username i password).

MVC daju razdvajanje poslovne logike, pristupa podacima i korisnicki interfejs u modele, kontrolere i poglede (MVC).

- Views omogucavaju programeru da precizno kontorlise HTML koji se renderuje.
- Mozemo koristiti Routing Engine da precizno kontrolisemo URL-ove
- Poslovna logika, ulazna logika i logika korisnickog interfejsa su odvojeni u modele, kontrolere i poglede.
- Moguca je upotreba Unit testing tehnika i Test Driven Development (TDD).

Normalno
http://mojsajt/faktura.asp?id=17&god=2018

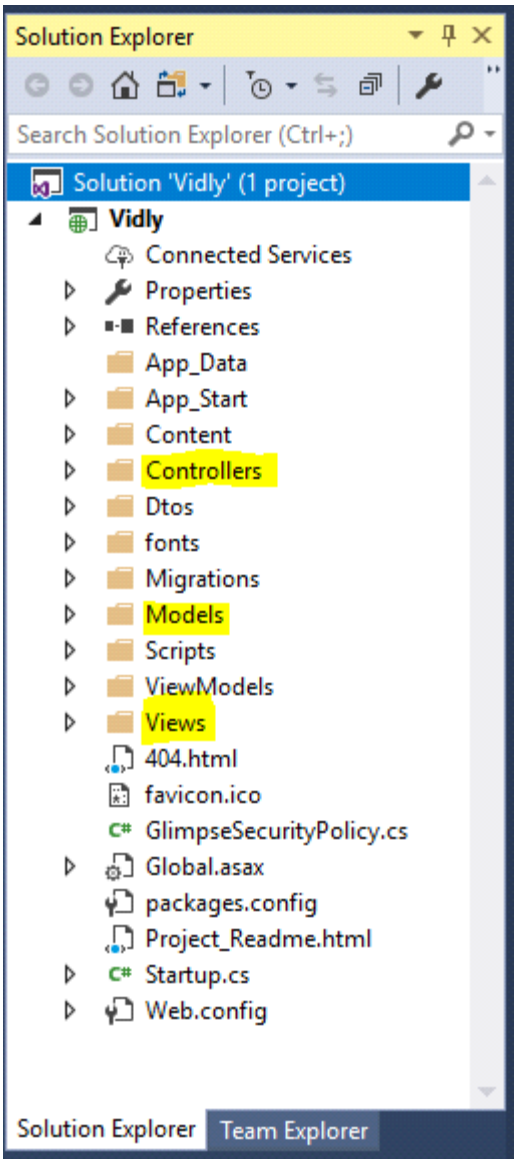
MVC
http://mojsajt/faktura/17/2018

mojsajt.com/Home/Contact
Home je kontroler, Contact je view.

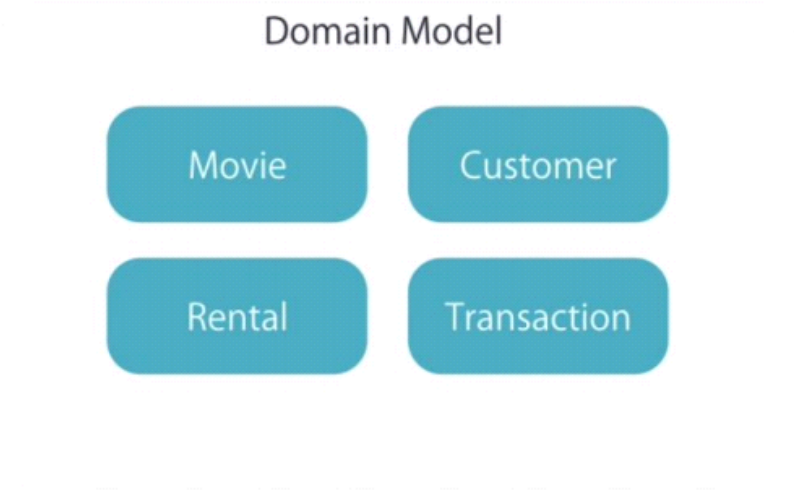
Federation za projekat - Prijava preko FB, GMail, Twitter i sl. dodatnih 10 poena. Barem 2 servisa implementirati. 1 za 10, 2 za 15. OAUTH.

Otvoriti vidly mvc file.

Mdf tj. baza se nalazi u App_Data


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Vidly' (1 project)
Vidly
Connected Services
Properties
References
App_Data
App_Start
Content
Controllers
Dtos
fonts
Migrations
Models
Scripts
ViewModels
Views
404.html
favicon.ico
GlimpseSecurityPolicy.cs
Global.asax
packages.config
Project_Readme.html
Startup.cs
Web.config
Solution Explorer
Team Explorer

MVC je skraćenica od Model – View – Controller i to je jedan od arhitekturalnih paterna za implementaciju korisničkih interfejsa.
Model čine aplikacioni podaci i ponašanje u pojmovima njegovog domena problema i nezavisnost od korisničkog interfejsa. Npr. u našem primeru, naš model će se sastojati od klasa kao što su: Movie, Customer, Rental i Transaction.



18. Domenski model

Ove klase imaju svojstva (eng. properties) i metode (eng. methods) koji predstavljaju stanje aplikacije i pravila. One zavezane za korisnički interfejs, to znači da možemo da uzmemo ove klase i da ih koristimo drugim aplikacijama kao što su desktop ili mobilne aplikacije. Ove klase su POCO *(eng. Plain Old CLR Objects)* objekti.

View – je HTML koji prikazujemo korisniku.

Controller – odgovoran za upravljanje HTTP zahtevima (*eng. requests*).

Pretpostavimo da je naša aplikacija postavljena *(eng. hosted)* na domen **vidley.com**.
Kada pošaljemo zahtev na **http://vidly.com/movies** kontroler će biti odabran da upravlja ovim zahtevom. Ovaj kontroler će dobiti sve filmove iz baze podataka, staviti ih u View i vratiti View klikentu ili pretražuvaču. U ovoj arhitekturi svaka komponenta ima različite odgovornosti i ovaj rezultat koji se ogleda u boljoj **podeli odgovornosti** (*eng. separation of concerns*) i kodu koji je lakše održavati (*eng. maintain*).

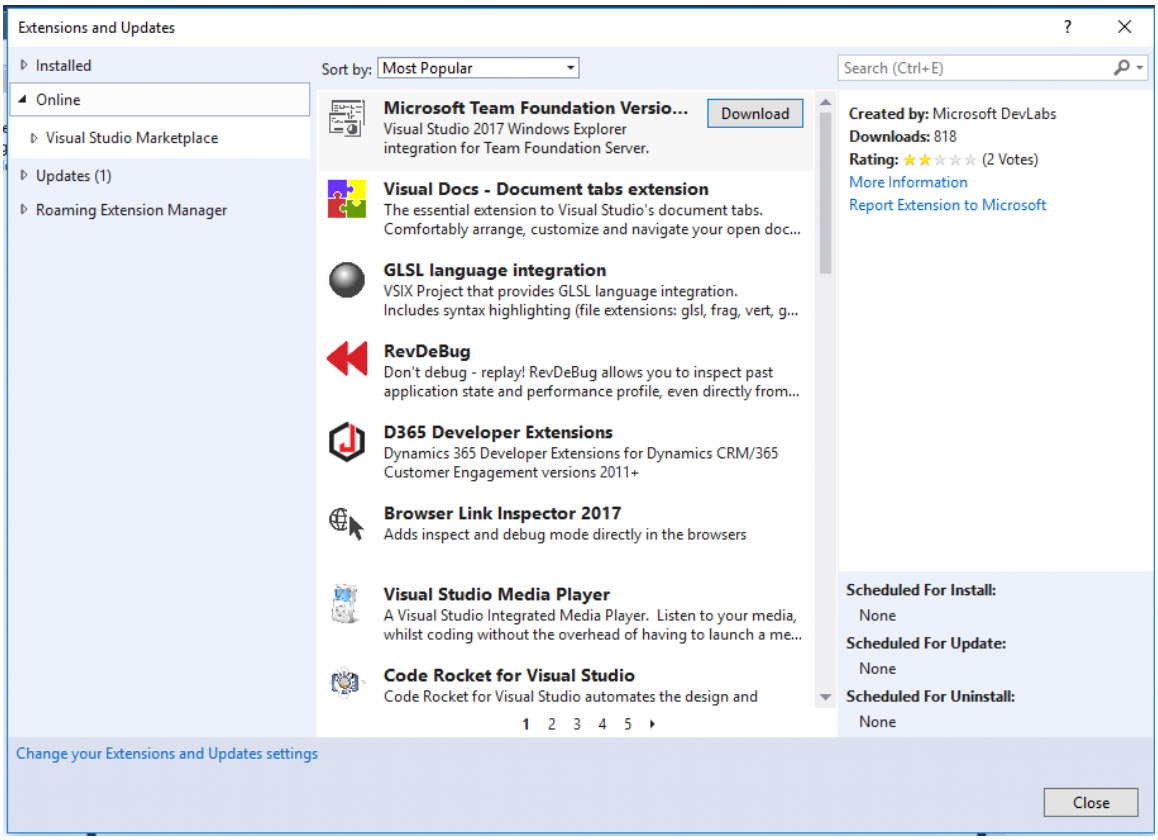
Postoji još jedan deo ove arhitekture koji se ne nalazi u MVC akronimu, ali je skoro uvek pristuan i naziva se **ruter** (*eng. router)*. Ranije smo rekli da kada zahtev dođe do aplikacije kontroler će biti odabran da upravlja i procesira tim zahtevom. Odabir odgovarajućeg kontrolera je u odgovornosti rutera. Ruter na osnovi nekih pravila zna da zahtev sa vašim URL-om **/movies** će biti procesiran preko klase koja se zove **MoviesController**. Tačnije to će biti procesirano od jedne metode klase MoviesController, jer klasa može da ima mnogo različitih metoda. U ASP.NET MVC-u metode kontrolera se nazivaju **akcije** *(eng. actions)*, tako da je prceiznije reći da je akcija u kontroleru odgovorna za upravljanje i procesiranje zahteva.

Ovo je bilo ukratko objašnjenje ASP.NET MVC-a.

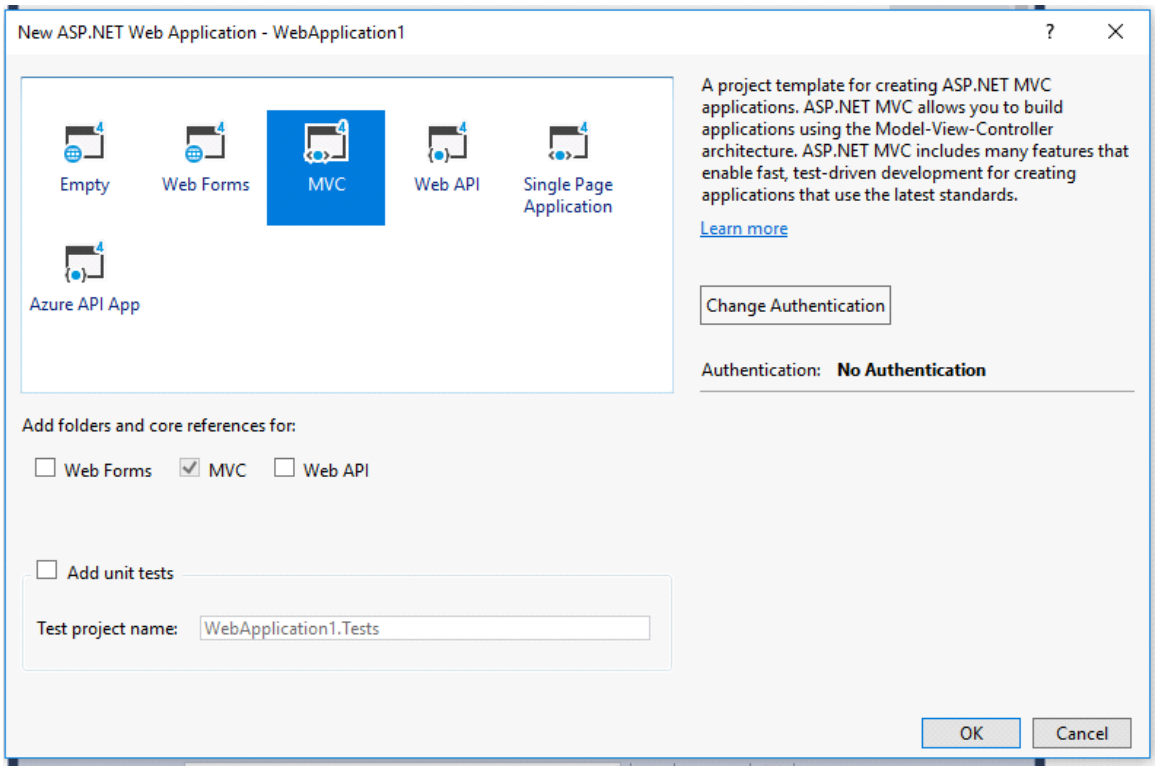
Tools > Extensions and updates > Skinemo Productivity Power Tools 2017

U C# osnovna klasa je **object**.

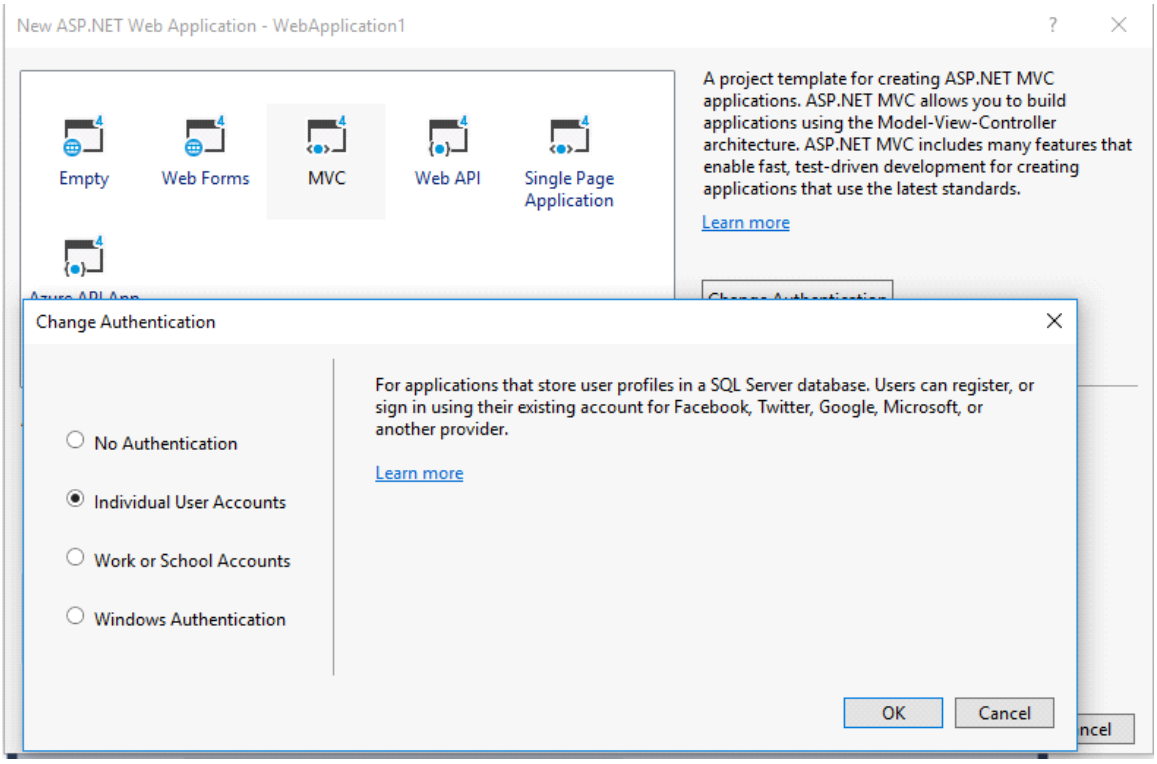
Kako neki paket promeni verziju, zavisnosti (dependencies) je bitno da se odrze.



File > New project > Web > ASP.NET Web Appplication (.NET framework)



Change Authentification - Odaberem individual user accounts

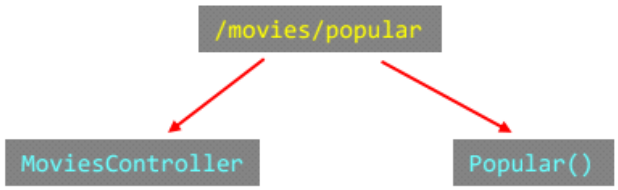


Prvi je **App_Data**, gde će naš fajl baze podataka biti skladišten.
Drugi je **App_Start**, koji uključuje nekoliko klasa koje će biti pozvane onda kada se aplikacija pokrene, tj. startuje.
Kao primer pogledajmo klasu **RouteConfig.cs**. Ovo je konfiguraciona klasa za pravila rutiranja (*eng. routing rules*).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace Vidly
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

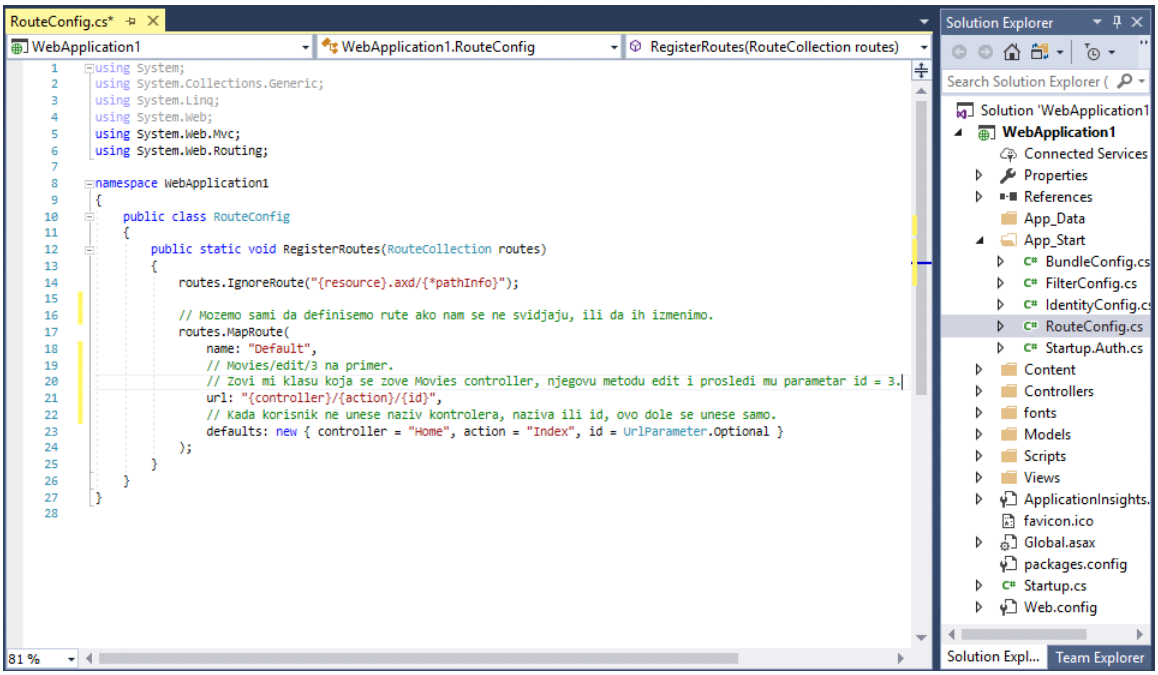
kod 1. klasa RouteConfig.cs

Vidimo da imamo rutu sa imenom Default i imamo odgovarajući URL patern. Ukoliko URL se poklopi sa ovim URL paternom, za prvi deo URL-a će se pretpostaviti da je **kontroler**, za drugi deo URL-a će se pretpostaviti da je **akcija** i treći deo je **id** koji se moće proslediti akciji (*eng. action*). Ukoliko imamo sledeće pravilo:

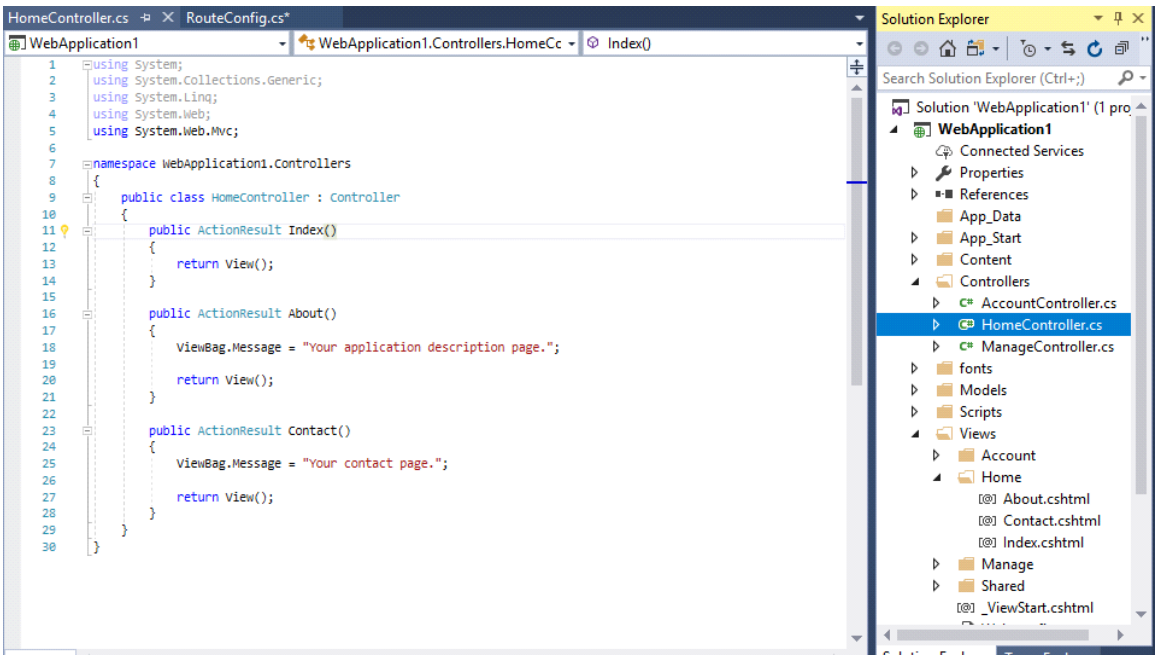


Dakle, ako nema nijedan parametar definisan, on ce po defaultu uzeti HomeController i izvrsiti metodu index (koja vraca view index) bez parametara.

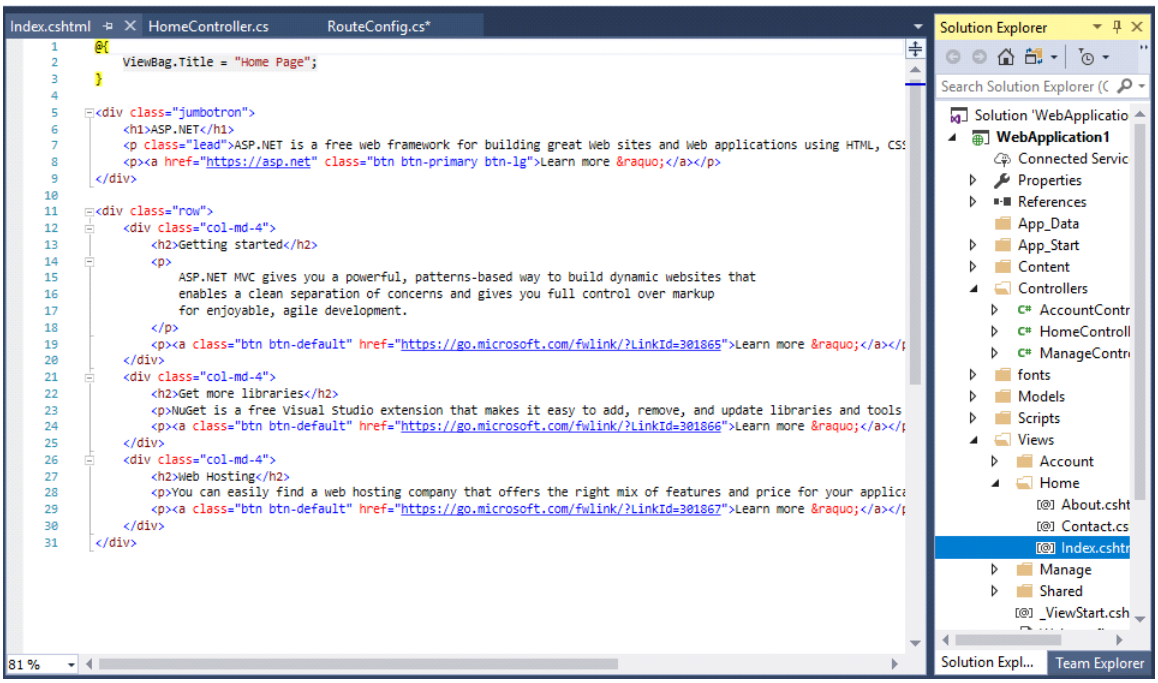
App_start > RouteConfig.cs



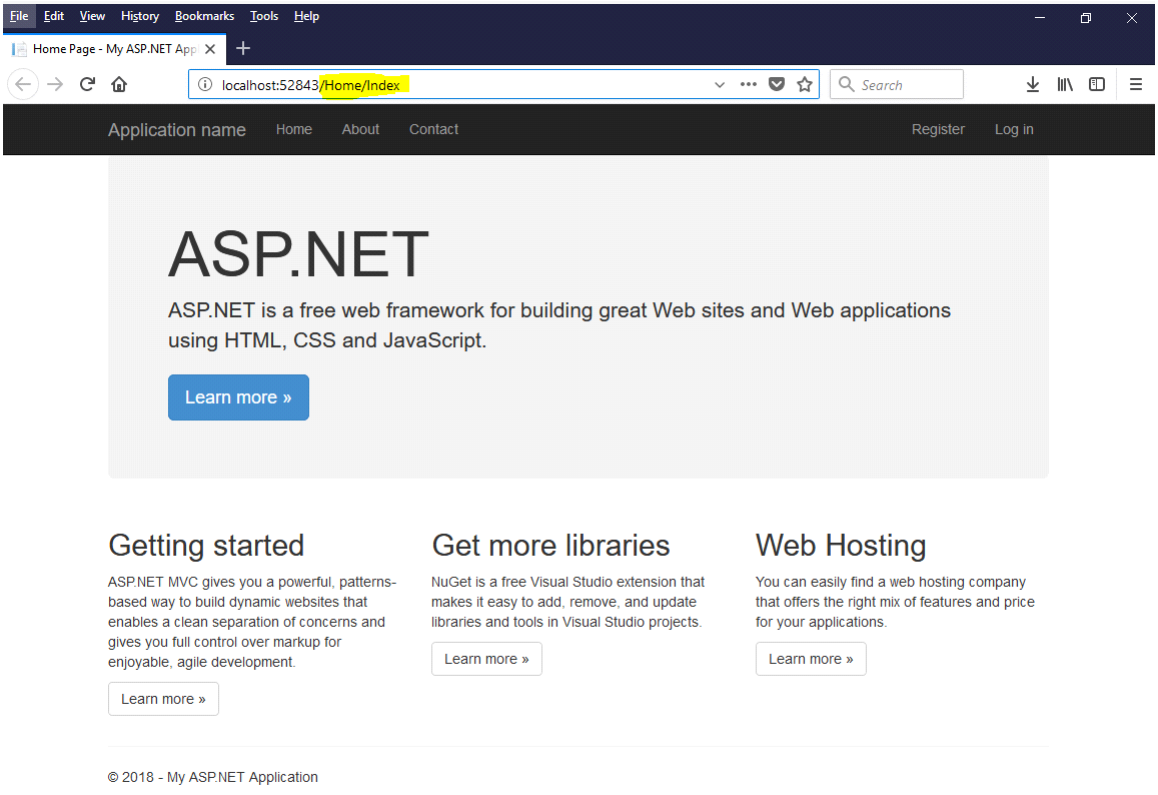
controller : Home *(Vidi sliku iznad - default, controller = "home")*
U controllers folderu naziva as <Ime>Controller, tj. u ovom slucaju HomeController.



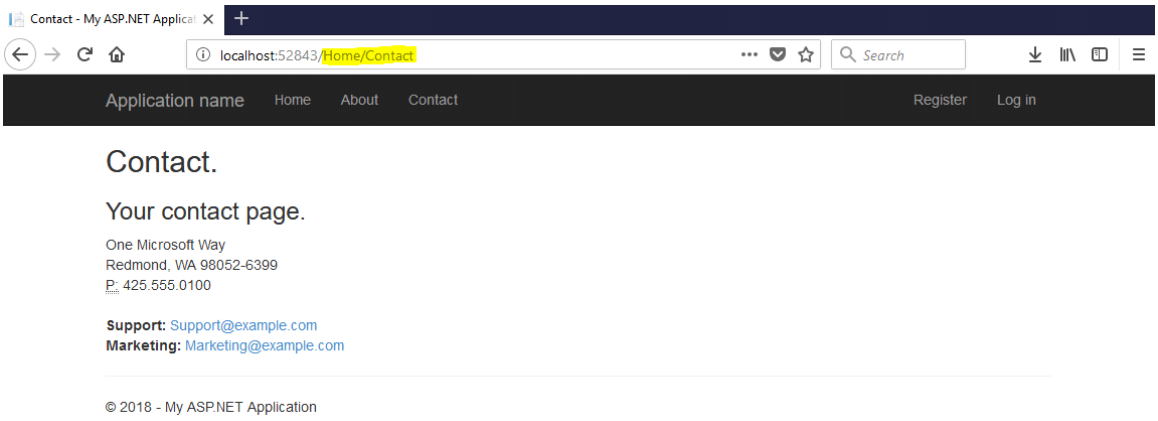
U views imamo folder Home i u njemu imamo html i cs.

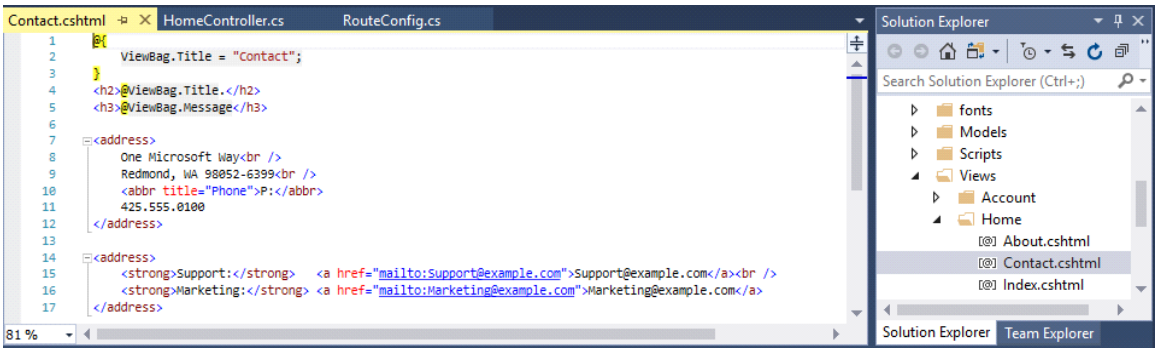


Home/Index - tj. Oba ova su default, dok je id opcioni (UrlParameter.Optional)



Home/Contact - vidi kontroler deo, vraca contact view. Poziva se HomeController i vrsi Contact metoda, koja vraca view tj contact iz folder Home u views. Folder u view ima isto ime.



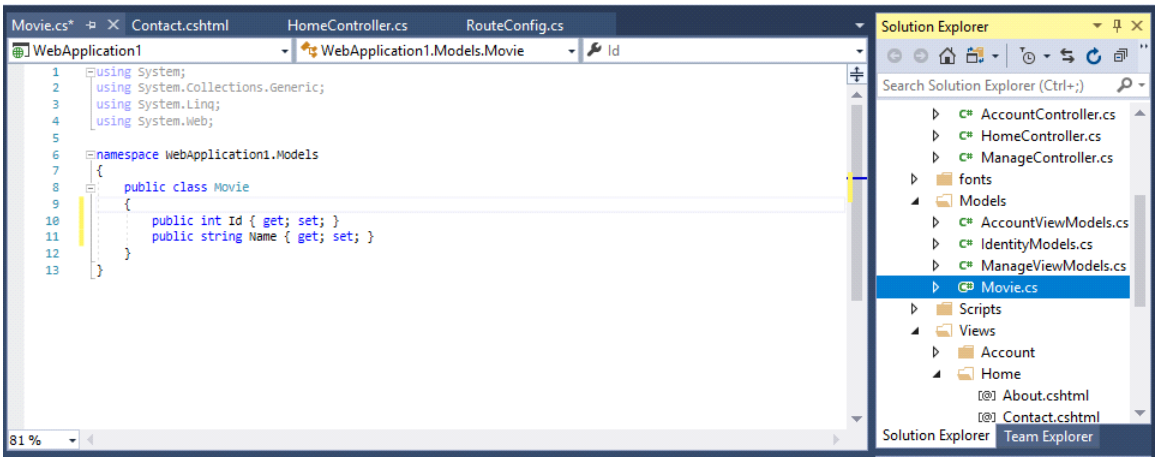


U VIEW IMAMO ONOLIKO FOLDERA KOLIKO IMAMO KONTROLERA + 1 ZAJEDNICKI ZA SVE.

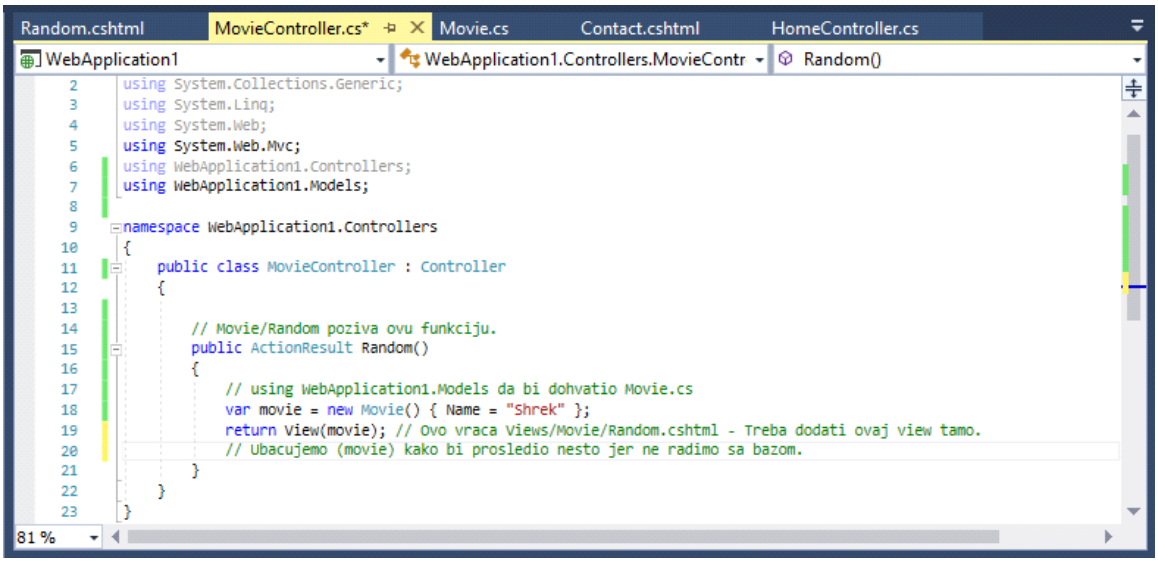
Telemetrija daje mogucnost da se prate i nadziru razni parametri web aplikacije - npr. da li mi neko pristupa nekom parametru 100000 puta.

Zadatak : Kreiranje stranice gde zelimo da odaberemo film - Movie nasumicno i da prikazemo njegove detalje. Pretpostavimo da ce stranica koju pozovemo biti na putanji /movie/random, te je potrebno da kreiramo kontroler MoviesController sa akcijom Random().

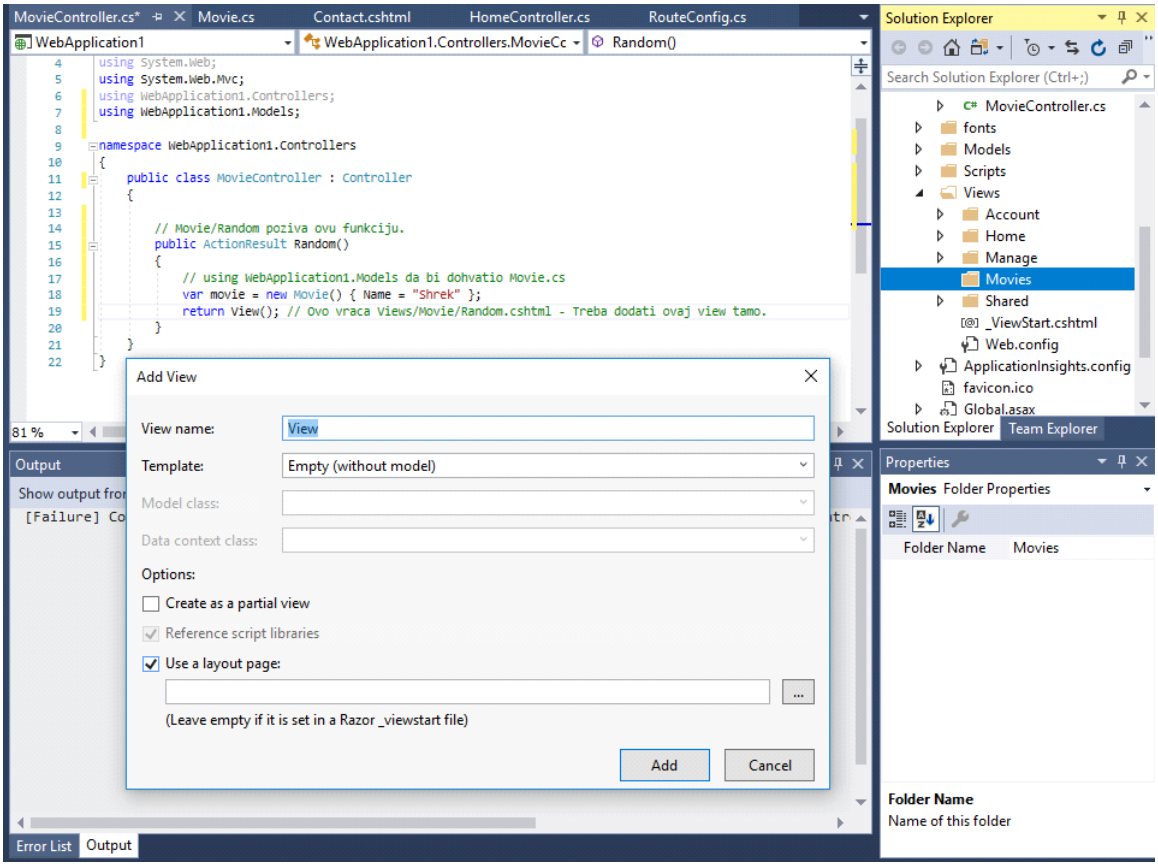
Desni klik na models > Add > New item > Class . Nazovemo ga movie i dodamo 2 property-a.



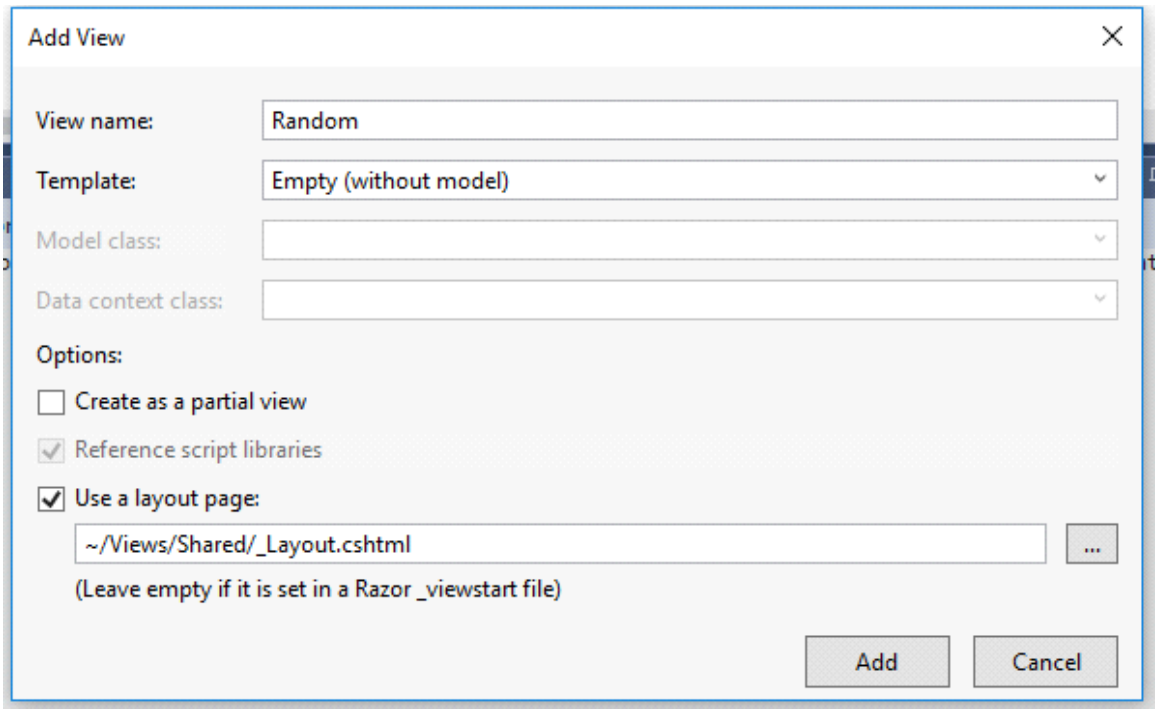
Desni klik na controllers > Add > Controller > MovieControler.cs

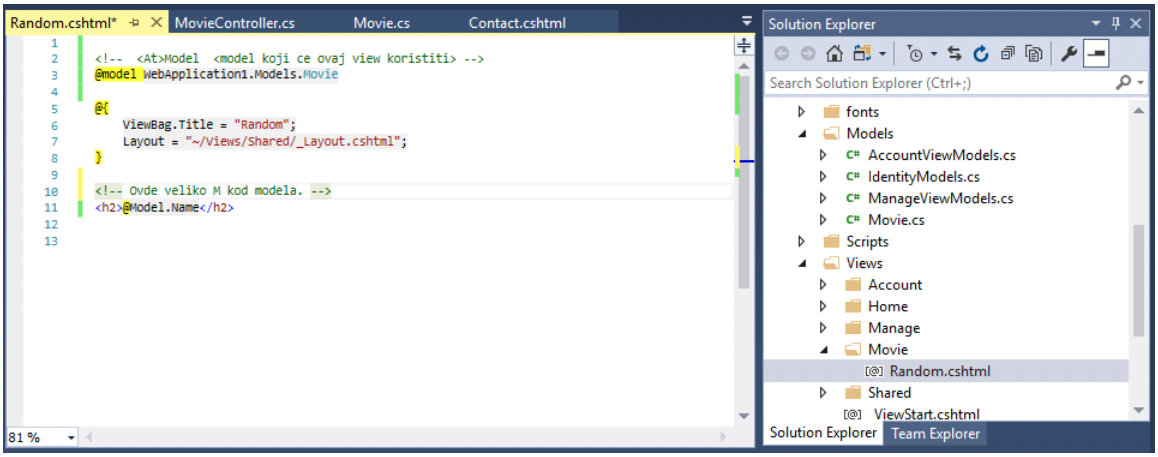


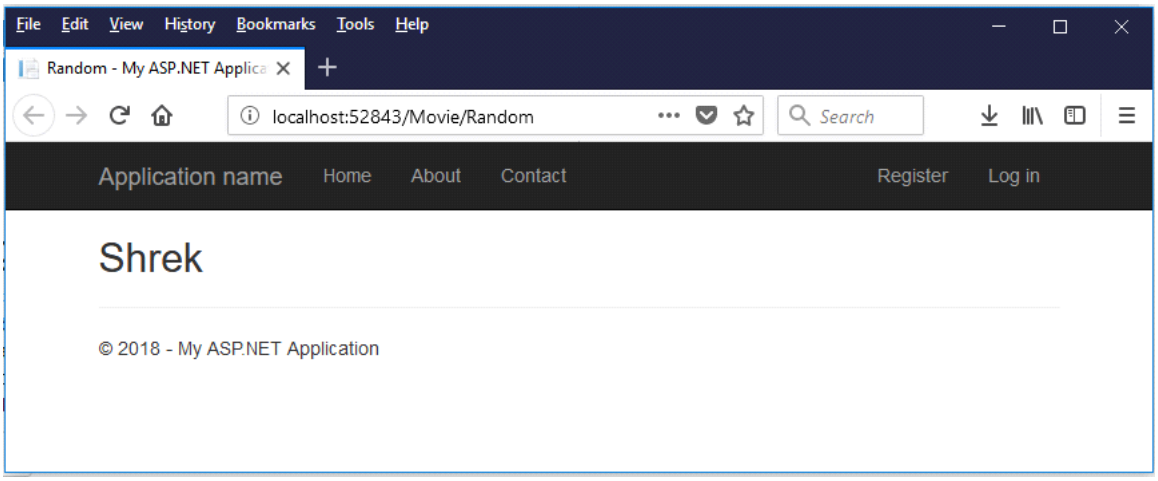
Desni klik na views folder > Add > View



Damo mu ime Random i cekiramo create as a partial view - bude vise kao widget koji moze da se koristi u vise razlicitih view-a.

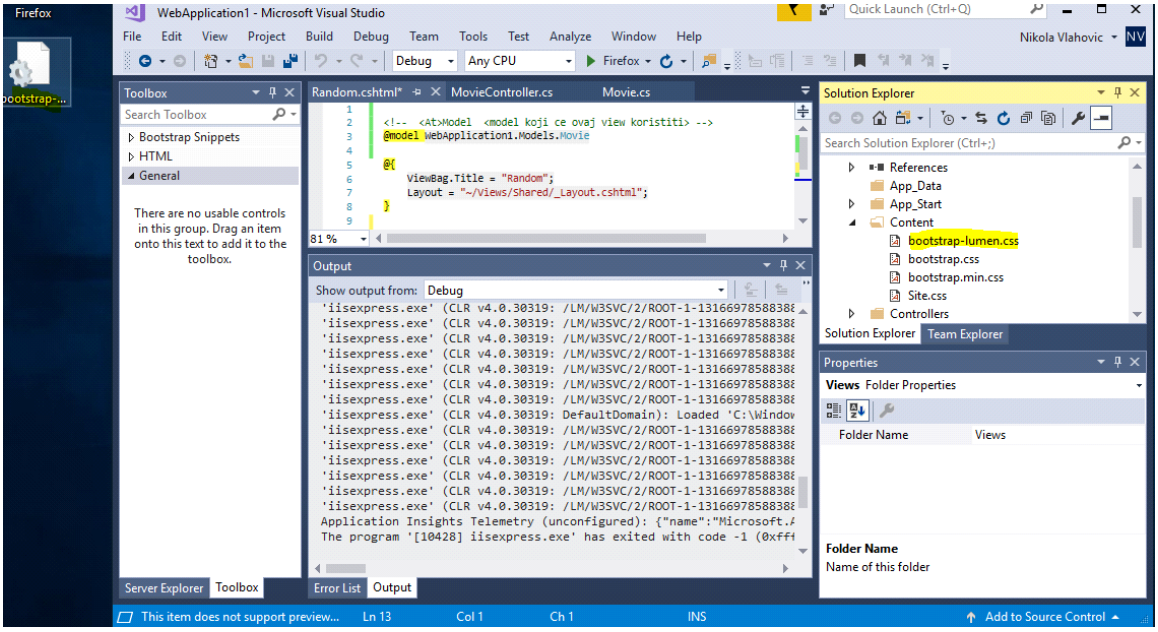

Add View
View name:
Random
Template:
Empty (without model)
Model class:
Data context class:
Options:
Create as a partial view
Reference script libraries
Use a layout page:
~/Views/Shared/_Layout.cshtml
(Leave empty if it is set in a Razor _viewstart file)
Add
Cancel


Random.cshtml*
MovieController.cs
Movie.cs
Contact.cshtml
<!-- <At>Model <model koji ce ovaj view koristiti> -->
@model WebApplication1.Models.Movie
@{
ViewBag.Title = "Random";
Layout = "~/Views/Shared/_Layout.cshtml";
}
<!-- Ovde veliko M kod modela. -->
<h2>@Model.Name</h2>
81 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
fonts
Models
AccountViewModels.cs
IdentityModels.cs
ManageViewModels.cs
Movie.cs
Scripts
Views
Account
Home
Manage
Movie
Random.cshtml
Shared
_ViewStart.cshtml
Solution Explorer
Team Explorer


File Edit View History Bookmarks Tools Help
Random - My ASP.NET Applica
localhost:52843/Movie/Random
Search
Application name
Home
About
Contact
Register
Log in
Shrek
© 2018 - My ASP.NET Application

Bootswatch.com > Lumens > Skinemo boostrap.css

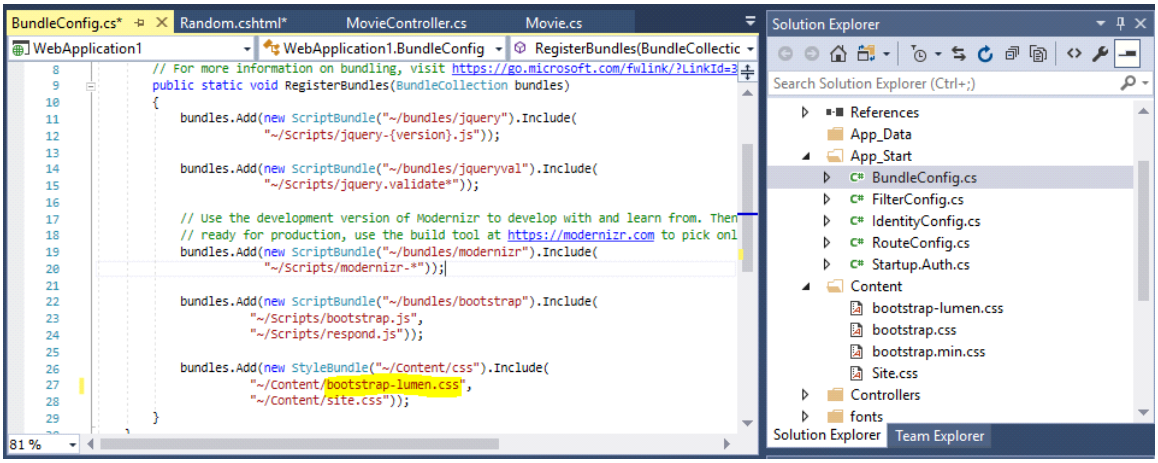
Potom drag and drop sa desktopa u Content

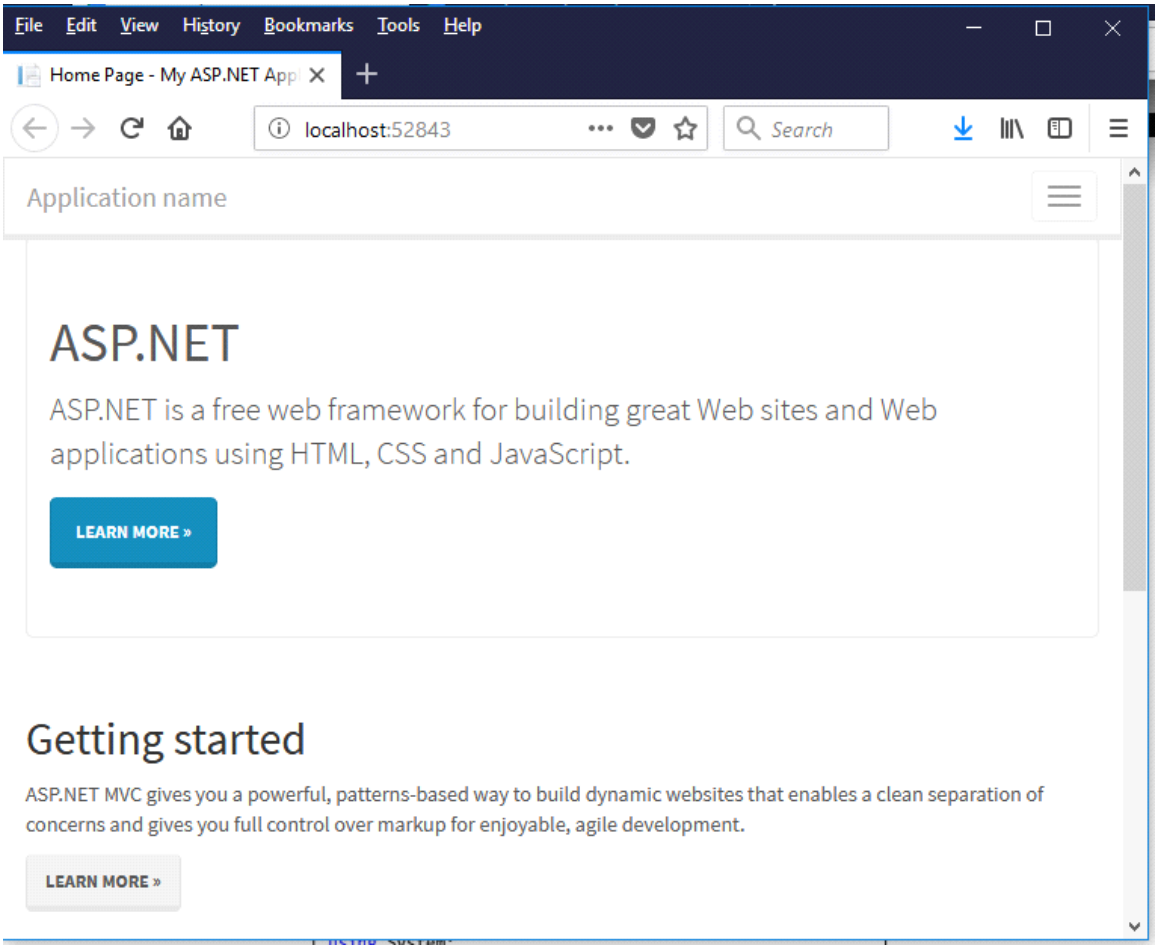


Potom App_Start>BundleConfig.cs

Ovde imamo gomilu bundle-ova : Nesto sto mi koristimo da kompresujemo nase http zahteve, tj. smanjujemo broj http zahteva koji klijent i server treba da razmene tako sto nabijemo vise .js ili .css u jednom npr.

Izmenimo poslednji bundle da bude bootstrap-lumen umesto samo bootstrap.





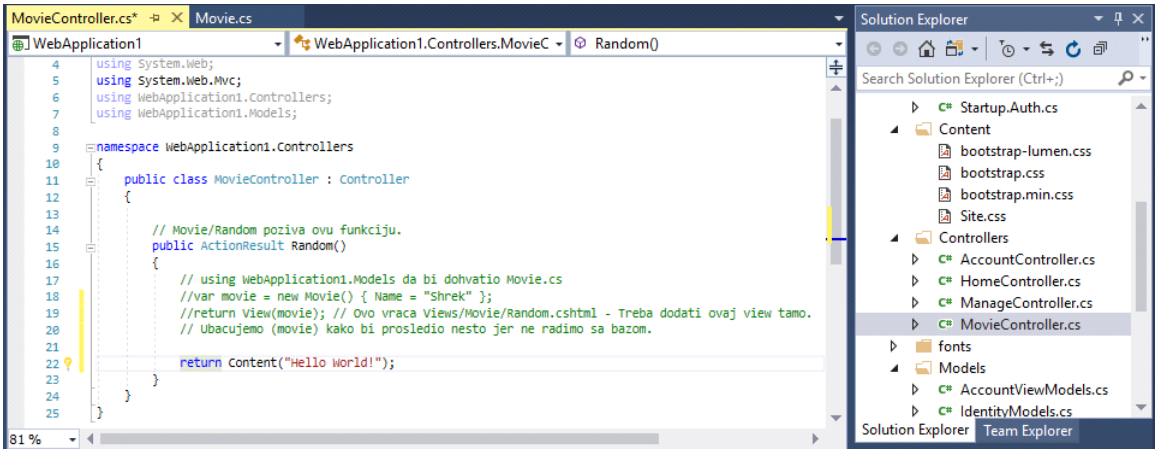
Poglavlje 2 - Strana 32

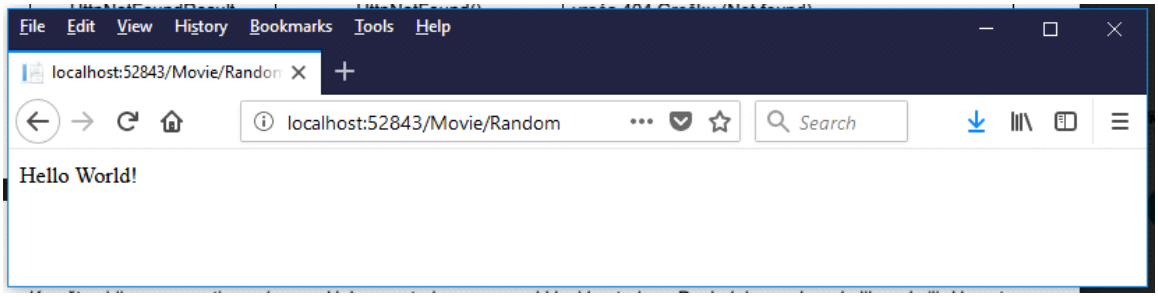
ActionResult je bazna klasa za sve akcije. U Zavisnosti sta Action radi, vratice jednu od instanci klasa koje nasledjuju - return View je helper metoda nasledjena iz osnovne klase Controller i ta metoda nam omogucava da brzo kreiramo ViewResult. *Vidi sliku MovieController iznad.*

return View ili ViewResult je isto fakticki.

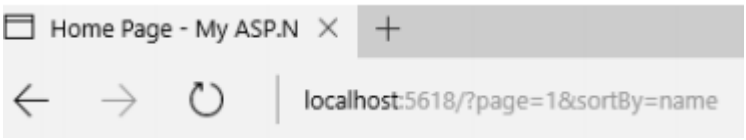
ActionResults

Type	Helper Method	Opis
ViewResult	View()	
PartialViewResult	PartialView()	vraća PartialView
ContentResult	Content()	vraća jednostavan tekst
RedirectResult	Redirect()	vrši preusmerenje (redirektovanje) korisnika na URL
RedirectToRouteResult	RedirectToAction()	vrši preusmerenje (redirektovanje) na akciju umesto na URL
JsonResult	Json()	vraća serijalizovani JSON objekat
FileResult	File()	vraća fajl
HttpNotFoundResult	HttpNotFound()	vraća 404 Grešku (Not found)
EmptyResult		koristi se kada akcija ne vrać ni jednu vrednost, kao void





return RedirectToAction("Index", "Home")
Redirektuj na metodu Index kontrolera Home-a. /Home/Index
return RedirectToAction("Index", "Home", new { page = 1, sortBy = "Name" });
Ovako prosledjujemo parametre.

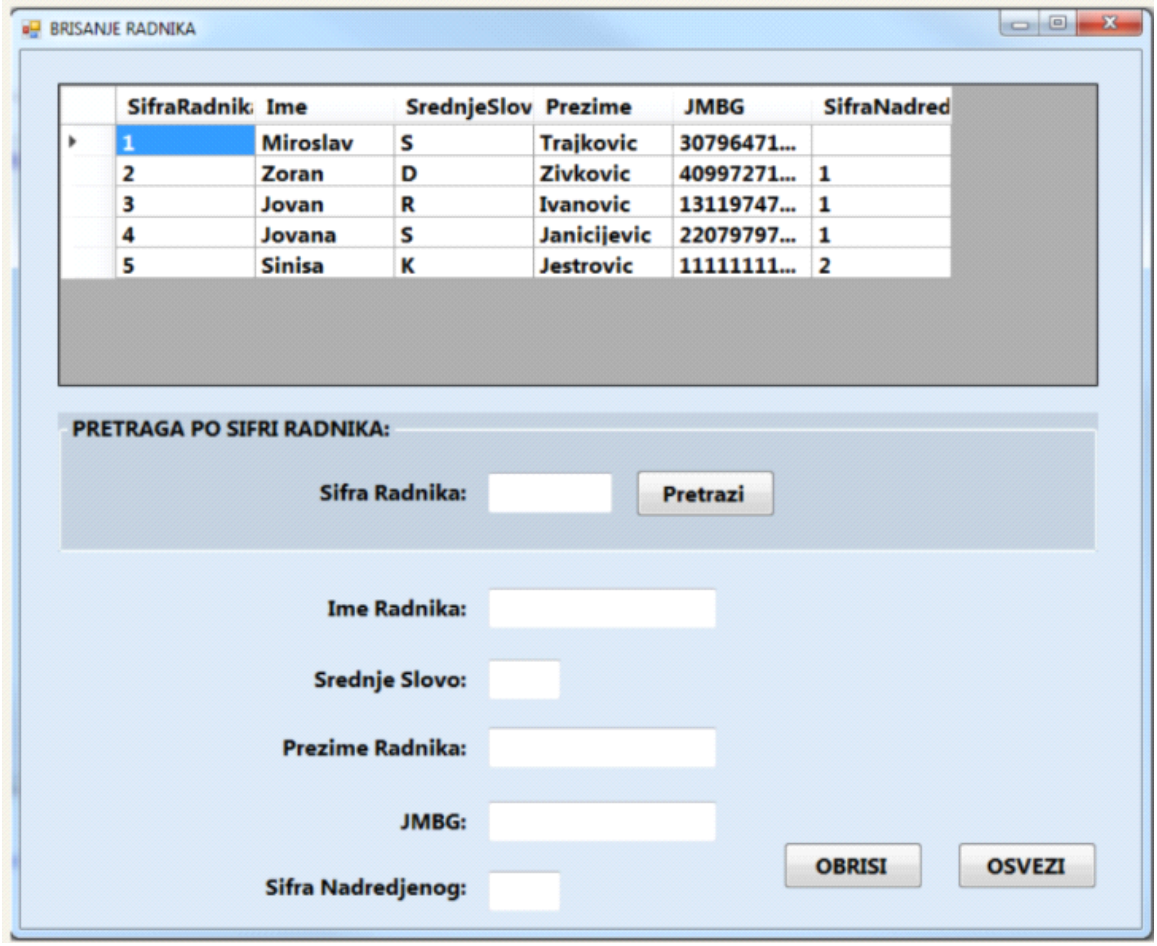


Rute definisu koji controller se poziva. **Controller** daje instrukciju **view**-u da se prikaze. View koristi **model** za podatke.

Novi cas - 04.04.18.

Troslojna aplikacija sa data access layer-om. - +15 poena
25.04.18. - Odbrana

Ne radi virtuelna masina, sada se radi DELETE. Sve je ovo konektovana arhitektura sa DataReaderom.



SqlDataReader cita podatke na najefikasniji nacin i zahteva aktivnu konekciju na izvor podataka (data source) dok cita podatke (ovo se obavlja sa .Open() metodom, a mozemo osigurati zatvaranje koristeci using ili finally). Instanca se kreira upotrebom metode ExecuteReader() objekta SqlCommand.

SQLDataReader je konektovana arhitektura/model jer zahteva otvorenu konekciju.

```
string konekcioniString =
ConfigurationManager.ConnectionStrings["mojKonekcioniString"].ConnectionString;
using (SqlConnection konekcija = new SqlConnection(konekcioniString))
{
   konekcija.Open();
   SqlCommand komanda = new SqlCommand("SELECT SifraArtikla, NazivArtikla FROM Artikal", konekcija);
   using(SqlDataReader rider = komanda.ExecuteReader())
   {
      GridView1.DataSource = rider;
      GridView1.DataBind();
   }
}
```

Ukoliko zelimo da prodjemo kroz redove obj. SqlDataReader mozemo iskoristiti metodu **Read()**.
OVO KORISTIMO ZA SELECT UPITE.

```
string konekcioniString = ConfigurationManager.ConnectionStrings["mojKonekcioniString"].ConnectionString;
using (SqlConnection konekcija = new SqlConnection(konekcioniString))
{
   konekcija.Open();
   SqlCommand komanda = new SqlCommand("SELECT SifraArtikla, NazivArtikla FROM Artikal", konekcija);
   using (SqlDataReader rider = komanda.ExecuteReader())
   {
      DataTable tabela = new DataTable();
      tabela.Columns.Add("Sifra");
      tabela.Columns.Add("Naziv");

      while(rider.Read())
      {
         DataRow red = tabela.NewRow();
         red["Sifra"] = rider["SifraArtikla"];
         red["Naziv"] = rider["NazivArtikla"];
         tabela.Rows.Add(red);
      }

      GridView1.DataSource = tabela;
      GridView1.DataBind();
   }
}
```

ADO.NET

- **ADO.NET** je skup biblioteka, tj. skup klasa koje su deo .NET framework-a, a koje se koriste za interakciju sa izvorima podataka (eng. data sources) kao što su: baze podataka, XML fajlovi, Excel fajlovi i Text fajlovi.
- ADO.NET je tehnologija za pristup podacima unutar Microsoft .NET Framework-a koja obezbeđuje komunikaciju sa relacionim i nerelacionim sistemima putem zajedničkih komponenti.
- ADO je skraćenica od Microsoft ActiveX Data Objects
- Windows aplikacije, ASP.NET Web aplikacije i ostali tipovi .NET aplikacija koriste ADO.NET da bi se uspostavila konekcija sa bazom, izvršila komandu tj. upit (eng. query) i vratili podaci iz baze.
  - *PRIMER1 – putanja za kod (**c:\Pristup**) – primer za pregled ADO.NET-a sa visokog nivoa*
    - *uspostavi se konekcija sa SQL Server-om*
    - *pripreme se I proslede se komande ka SQL Server-u*
    - *izvrše se komande na SQL Server*
    - *kada se rezultat izvršanja komande vrati – prikazati te podatke u aplikaciji*
- za izvršavanje pomenutog primera prvo je potrebno da uspostavimo konekciju (eng. connection)
- klase koje nam pomažu da se povežemo sa SQL Server-om se nalaze u **System.Data.SqlClient**;

```
using System.Data.SqlClient;
```

1. **Uspostavljanje SQL konekcije – *kreiranje SqlConnection objekta***

```
SqlConnection konekcija = new SqlConnection("Data Source=DE-XII-SVESIC\\SQLEXPRESS;Database=BazaRacuna;Integrated Security=SSPI;");
```

konekcioni string (eng. connection string)

2. **Priprema SQL komande – koristimo *SqlCommand* objekat**

```
SqlCommand komanda = new SqlCommand("SELECT * FROM Radnik", konekcija);
```

3. **Otvaranje konekcije – koristimo metodu *Open()***

```
konekcija.Open();
```

4. **Izvršavanje komande – za ovaj primer koristimo metodu *ExecuteReader()*, koji se skladišti u objekat *SqlDataReader***

```
SqlDataReader rider = komanda.ExecuteReader();
```

5. **Prikazujemo podatke preko *GridView* kontrole**

```
GridView1.DataSource = rider;
GridView1.DataBind();
```

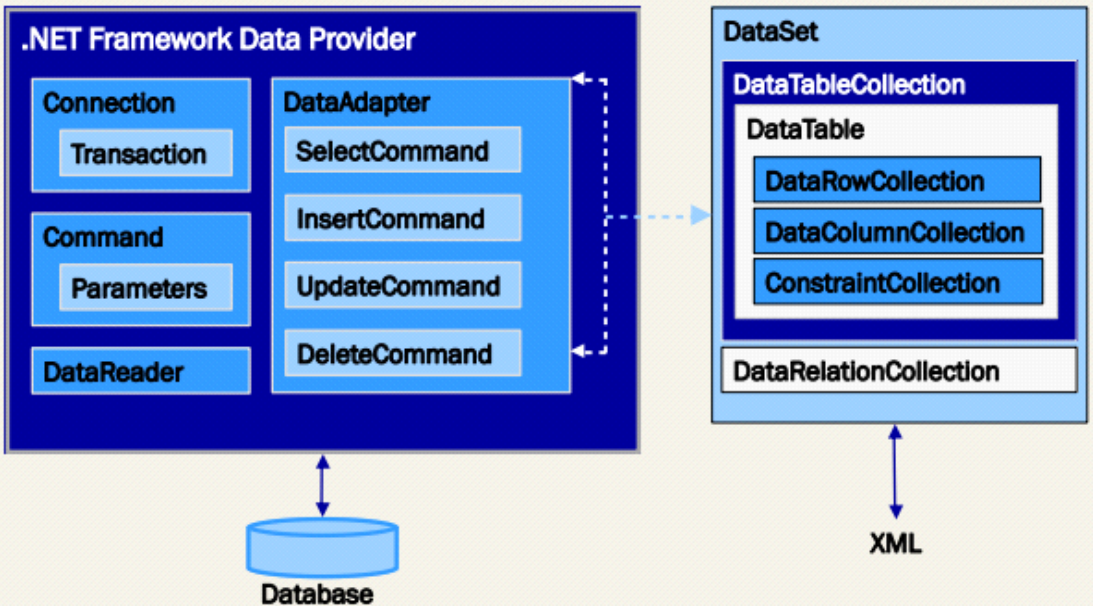
6. **Zatvaranje SQL konekcije**

```
konekcija.Close();
```

DISKONEKTOVANI MODEL : KORISTIMO DATA ADAPTER I DATA SET. Data set je u system.data, jedina klasa koja radi sa podacima koja nije vendorski specificna, univerzalna je u odnosu na sve ostale.

- ADO.NET podržava 2 načina pristupa podacima:

1. **konektovana arhitektura (eng. connected architecture)**
2. **diskontektovana arhitektura (eng. disconnected architecture)**



Kreiramo instancu DataAdaptera i kazemo koju komandu zelimo da izvrsimo i konekciju preko koje zelimo da je izvrsimo, mozemo da radimo INSERT DELETE UPDATE i SELECT kroz jedan DataAdapter. DataSet je nezavisan od Database provajdera/vendora i njega je najlakse shvatiti kao reprezentaciju tabele u memoriji, on ne samo sto moze da cuve tabele, vec i odnose izmedju njih definisanih u bazi. Imamo **FILL()** metodu DataAdaptera cime ga punimo podacima iz DataSeta. FILL metoda otvara

konekciju, izvrsava naredbu, cita podatke u dataset i zatvara konekciju. DataAdapter sam upravlja o konekciji i mi ne moramo da vodimo racuna o otvaranju i zarvaranju konekcije.
Gore prvo ide using System.Data.

```
string konekcioniString = ConfigurationManager.ConnectionStrings["mojKonekcioniString"].ConnectionString;

using(SqlConnection konekcija = new SqlConnection(konekcioniString))
{
   SqlDataAdapter adapter = new SqlDataAdapter("SELECT SifraPartnera, NazivPartnera, DatumPrijave FROM Partner",
konekcija);
   DataSet set = new DataSet();
   adapter.Fill(set);

   GridView1.DataSource = set;
   GridView1.DataBind();
}
```

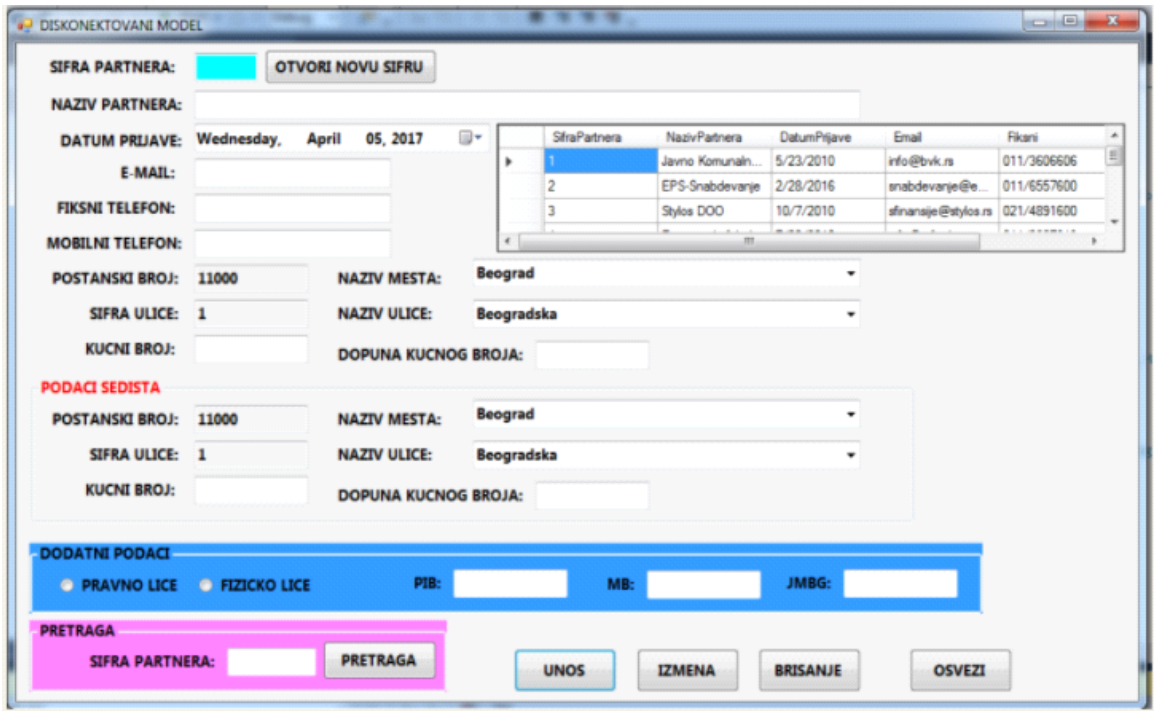
Pozivanje uskladistene procedure i prosledjivanje parametra *(iz textBox1)*

CREATE PROCEDURE spVratiArtikalPoSifri
@Sifra INT
AS
BEGIN

SELECT SifraArtikla, NazivArtikla FROM Artikal WHERE Sifra = @Sifra
END

// exec spVratiPartnera.

```
string konekcioniString = ConfigurationManager.ConnectionStrings["mojKonekcioniString"].ConnectionString;

using (SqlConnection konekcija = new SqlConnection(konekcioniString))
{
   SqlDataAdapter adapter = new SqlDataAdapter("spVratiArtikalPoSifri", konekcija);

   //SqlDataAdapter adapter = new SqlDataAdapter();
   //adapter.SelectCommand = new SqlCommand("spVratiArtikalPoSifri", konekcija);

   adapter.SelectCommand.CommandType = CommandType.StoredProcedure;
   adapter.SelectCommand.Parameters.AddWithValue("@Sifra", TextBox1.Text);
   DataSet set = new DataSet();
   adapter.Fill(set);

   GridView1.DataSource = set;
   GridView1.DataBind();
}
```



Troslojna - Klasa za svaki domenski objekat. Ako imamo stavku fakturi, da imamo listu stavki i tu se nadovezuju stvari koje se dalje kreiraju. Ako vec ima nesto u bazi on mora nesto da cita i formira listu. Prvo napravimo listu pa napravimo fakturu.Forma ima nesto sto se zove zaglavlje i stavka dokumenta. Ovo cemo da radimo sa DataRow. (deo kod while(reader.read()) sa sl. gore)

https://stackoverflow.com/questions/3109009/how-to-make-a-datatable-from-datagridview-without-any-datasource

```
using System.Data.SqlClient;
using System.Data;

        Void IzvrsiUpit(string KonekcioniString, string Naredba, DataGridView GridView)
        {
                try
                {
                using(SqlConnection Konekcija = new SqlConnection(konekcioniString))
                {
                        SqlDataAdapter Adapter = new SqlDataAdapter('"' + Naredba +'"', Konekcija);
                        DataSet Set = new DataSet();
                        Adapter.Fill(set);

                GridView.DataSource = Set;
                GridView.DataBind();
        }
}
catch(Exception EX)
{
        MessageBox.Show(" Greska - " + EX.Message, "Greska");
}
}
```