FILOZOFSKI FAKULTET

ODSJEK INFORMACIJSKIH I KOMUNIKACIJSKIH ZNANOSTI

AKADEMSKA GODINA 2017/2018

Ivan Ćosić

# PRISTUP BAZAMA PODATAKA S WEBA

DIPLOMSKI RAD

MENTOR: doc. dr. sc. Vedran Juričić

ZAGREB, prosinac 2017.

# Sadržaj

# UVOD

U diplomskom radu govorit će se općenito o bazama podataka te njihovim dijelovima i arhitekturi, zatim o SQL jeziku za pristup podatcima unutar jedne ili više baza podataka a biti će prikazani i primjeri SQL naredbi i procedura. Govorit će se o web servisima općenito, kao i o poznatijim servisima kao što su SOAP, REST i RPC.

U diplomskom radu predstavit će se i alat za izgradnju servisno-orijentiranih aplikacija na Windowsu, WCF (eng. *Windows Communication Foundation*), te će se reći nešto više o dijelovima WCF-a kao što su *contracts, bindings, endpoints* te će se za iste prikazati i primjeri. Uz rad će biti razvijana web aplikacija pomoću MVC (eng. *Model-View-Controller*) tehnologije.

U diplomskom radu obradit će se različiti načini pristupa bazama podataka iz web aplikacija. Prikazat će se protokoli, tehnologije i biblioteke koda koji omogućuju rad s najčešće korištenim standardima u dohvaćanju podataka iz baze. U sklopu rada bit će dizajnirana baza podataka i web aplikacija, kao primjer pristupa bazi podataka putem web aplikacije.

# 1. BAZE PODATAKA

Baza podataka se može definirati kao skup međusobno povezanih podataka pohranjenih bez nepotrebne zalihosti s ciljem da na optimalni način posluže u raznim primjenama. Podaci se spremaju neovisno o programima koji ih koriste, zajedničkim pristupom dodaju se novi podaci te mijenjaju i premještaju postojeći.[1](Slika 1.)

Slika 1. Baza podataka

Prvi sustavi upravljanja bazama podataka su razvijeni u 1960-ima, a začetnik je bio *Charles Bachman*. Dva su ključna modela podataka u to vrijeme: *CODASLY* je razvio mrežni model na osnovu Bachmanovih ideja, te hijerarhijski model koji se koristio u sustavu razvijenom od strane *North American Rockwell* kojeg je kasnije prihvatio *IBM* kao temelj svojeg *SUI* proizvoda. *E. F. Codd* je 1970. godine razvio Odnosni model, koji je dugo vremena ostao samo u području akademskih interesa, dok su se *CODASYL* i *SUI* koristili kao rješenje praktičnog inženjerstva. Devedesetih godina pažnju su privukle baze

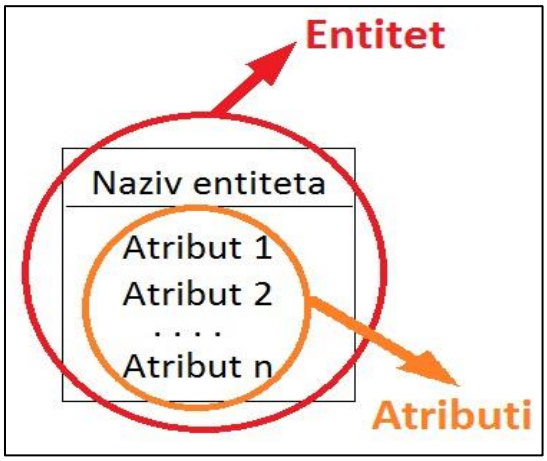
[1]*Usp.* Informacijski sustavi, Dostupno na: http://www.pfri.uniri.hr/~tudor/materijali/Informacijski%20sustavi,%20baze%20podataka.htm , (31.07.2017.).

podataka orijentirane prema objektu, da bi 2000.-ih na scenu stupile *XML* baze podataka čiji je cilj ukloniti podjelu između dokumenata i podataka kako bi se svi organizacijski informacijski resursi držali na jednom mjestu.[2]

### 1.1. Osnovni pojmovi u bazi podataka

Baza podataka je skup međusobno povezanih podataka, koje korisnici mogu pregledavati, pretraživati i uređivati.

Entiteti su osnovni elementi baze podataka koji se sastoje od atributa koji ga opisuju (Slika 2.).




Slika 2. Entitet i atributi

Atribut se sastoji od naziva kojim označavamo svojstva koja želimo znati o entitetu (npr. ime, prezime, spol, visina), zatim posjeduje vrijednost kojom se opisuje svojstvo (npr. Ivan, Horvat, Muško, 180 cm). Domena atributa je definirani skup dozvoljenih vrijednosti za atribut.

Podatak je činjenica predočena u formaliziranom obliku (npr. riječ, broj ili slika).[3]

---

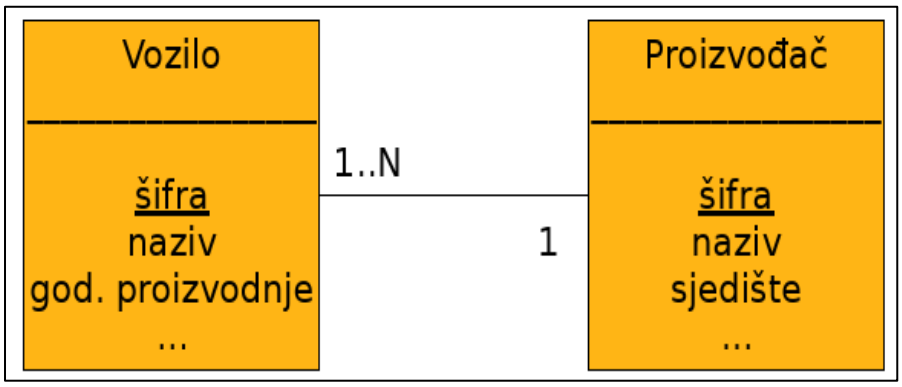
[2] *Usp.* Wikipedia, Baza podataka, Dostupno na: https://hr.wikipedia.org/wiki/Baza_podataka , (31.07.2017.).

[3] *Usp* Wikipedia, *Podatak*, Dostupno na: https://hr.wikipedia.org/wiki/Podatak, (12.20.2017.).

N-torka predstavlja jedan redak u relaciji, pritom jedna relacija ne smije sadržavati dvije jednake n-torke.

Ključ relacije je skup atributa čije vrijednosti jednoznačno određuju n-torku, pri tomu ključ mora ispunjavati svojstva minimalnosti i jednoznačnosti. Postoji primarni ključ kojeg dobiva kandidat koji najbolje odgovara tj. opisuje neki entitet, svi ostali kandidati postaju alternativni (strani, vanjski) ključevi.

Nul vrijednost označava stanje odnosno vrijednost kada se u bazu ništa ne unese i samim time vrijednost atributa ostaje nepoznata. Nul vrijednost se određuje ukoliko ne postoji neka vrijednost ili ukoliko ne poznajemo vrijednost.




Slika 3. Primjer veze između dva entiteta

Veze među entitetima se uspostavljaju između dva ili više entiteta. Vrste veza među entitetima:

- Jedan na prema jedan (1 : 1) – jedan član prvog entiteta može biti u vezi samo s jednim članom drugog entiteta
- Jedan na prema više (1 : N) – jedan član prvog entiteta može biti u vezi s više članova drugog entiteta, ali jedan član drugog entiteta može biti u vezi samo s jednim članom prvog entiteta (Slika 3.)
- Više na prema više (N : N) – jedan član prvog entiteta može biti u vezi s više članova drugog entiteta, te jedan član drugog entiteta može biti u vezi s više članova prvog entiteta

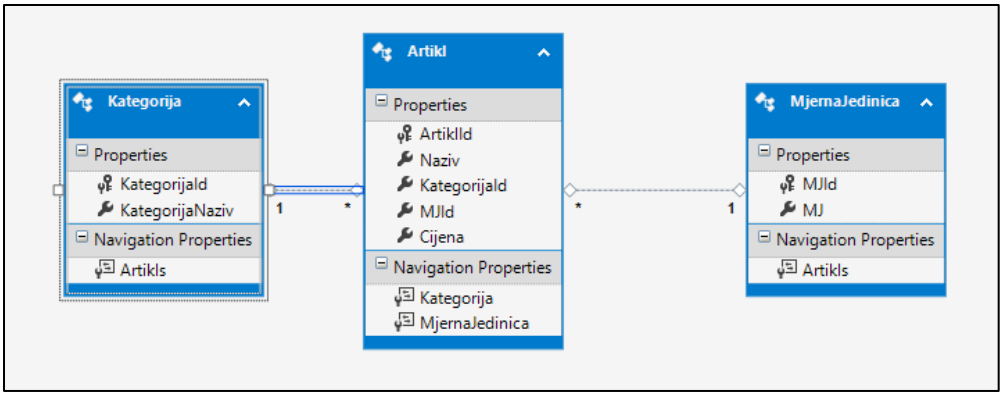
## 1.2. DBMS

Model baze podataka služi za opisivanje izgleda logičke strukture baze te čini osnovu za koncipiranje, projektiranje i implementaciju baze podataka. Model baze podataka predstavlja osnovni koncept za razvoj sustava za upravljanje bazom podataka *DBMS* (eng. *Data Base Management System*). *DBMS* je poslužitelj baze koji oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom i obavlja sve operacije s podacima.

Modeli baza podataka:

- Relacijski model – relacija se definira kao skup n-torki sa istim atributima, podaci se organiziraju u skup relacija koje su povezane i definirane određenim vezama, definicija jedne relacije naziva se relacijska shema koja se sastoji od imena relacije i popisa atributa te relacije, upravljanje relacijskom bazom podataka vrši *RDBMS*[4] sustav (Slika 4.)
- Hijerarhijski model – organizira podatke u čvorove, a čvorovi su točke povezane odgovarajućim vezama, temeljno pravilo hijerarhijske baze podataka je da jedan slog (čvor) ne može imati dva nadređena sloga
- Mrežni model – je nastao paralelno s hijerarhijskim modelom, a zasniva se na ideji mrežne strukture, osnovni koncepti mrežnog modela su polja, slogovi i setovi
- Objektni model – podaci se definiraju kroz objekte a sam model se definira kroz objektnu bazu podataka, objektni model podataka je logički model podataka koji prihvaća semantiku objekata podržanu u objektno-orijentiranom programiranju

[4] RDBMS (eng. *Relation database management system*) – je sustav za upravljanje bazama podataka zasnovan na relacijskom modelu, a koriste ga mnoge baze podataka širom svijeta

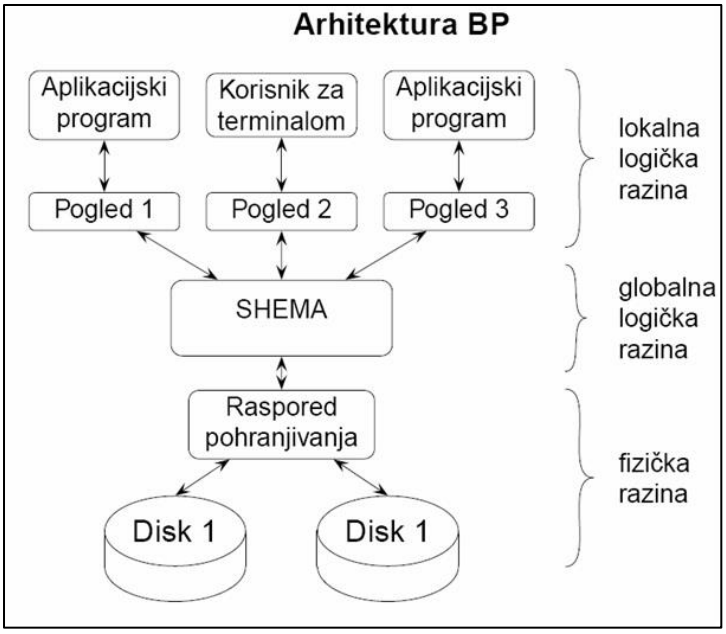



Slika 4. Primjer relacijskog modela

Postoje tri razine u arhitekturi baze podataka (Slika 5.), to su:

- Fizička razina – ona se odnosi na fizički prikaz i raspored podataka na jedinicama vanjske memorije, raspored pohranjivanja opisuje preslikavanje elemenata logičke definicije baze na fizičke uređaje
- Globalna logička razina – ona se odnosi na logičku strukturu cijele baze, zapis logičke definicije se naziva shema i njome se definiraju tipovi podataka te veze među tipovima
- Lokalna logička razina – ona se odnosi na logičku predodžbu o dijelu baze kojeg koristi neka aplikacija, zapis lokalne logične definicije je tekst ili dijagram kojim se definiraju svi lokalni tipovi podataka i veza među tipovima[5]

[5] *Usp.* Informacijski sustavi, Dostupno na: http://www.pfri.uniri.hr/~tudor/materijali/Informacijski%20sustavi,%20baze%20podataka.htm , (01.08.2017.).




Slika 5. Arhitektura baze podataka

Da bi stvorili bazu podataka potrebno je odrediti shemu, *DBMS* tada preuzima kontrolu te automatski generira raspored pohranjivanja i fizičku bazu podataka. Korisnici ne pristupaju izravno bazi podataka nego pomoću *DBMS*-a pohranjuju ili dobivaju tražene podatke. Sama komunikacija između *DBMS*-a i baze podataka se odvija pomoću jezika od kojih je najpoznatiji *SQL* koji podržava relacijski model podataka.

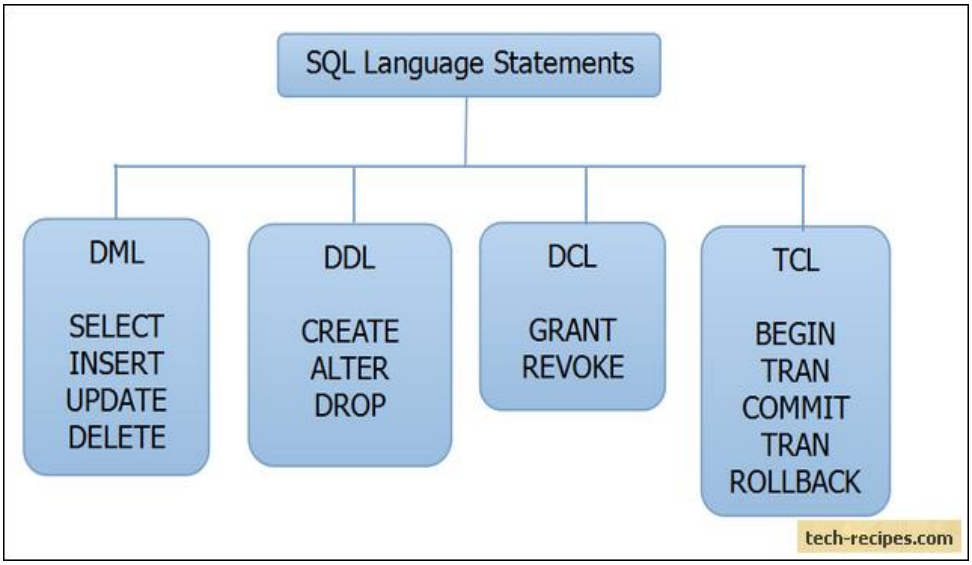
*DBMS* mora biti usklađen s *ACID*[6] pravilima. Atomarnost (engl. *atomicity*) pravilo određuje da će se sve operacije unutar transakcije izvršiti ili neće niti jedna. Konzistentnost (engl. *consistency*) pravilo određuje da baza mora korisniku javiti grešku ukoliko se pokuša unijeti pogrešan ili nedozvoljen podatak, pravila integriteta u bazi podataka nakon transakcije ostaju zadovoljena. Izolacija (engl. *isolation*) pravilo uređuje kada i kako se provode promijene u odnosu na ostale istovremene promjene u bazi. Izdržljivost (engl. *durability*) pravilo uređuje da se sve operacije tj. transakcije koje se izvedu moraju pohraniti u bazu i postaju trajne tj. ne gube se nakon pada sustava.

[6] *ACID – Atomicity, Consistency, Isolation and Durability*

### 1.3. *SQL* – jezik za rad s bazama podataka

*SQL* (eng. *Structured Query Language*) je standardni jezik za pristup i rad s bazama podataka, *SQL* je najviše korišteni programski jezik za baze podataka. *SQL* postaje 1986. godine standard Američkog nacionalnog instituta za standarde (*ANSI*), a 1987. godine postaje standard Međunarodne organizacije za standarde (*ISO*). Spada u deklarativne jezike kojima se opisuje što se želi dobiti kao rezultat, ali ne i način na koji će se doći do željenog rezultata, odnosno izvršiti procedura.

*SQL* je razvijen u *IBM*-a od strane *Donald G. Čemberlena i Rejmond F. Bojsa* u ranim 1970-im, a prvobitno je nazvan *SEQUEL* (eng. *Structured English Query Language*). Iako je *SQL* inspiriran *Coddovim* radom, *IBM*-ovci *Chamberlin i Raymond* su postali autori *SEQUEL* jezičnog dizajna. *SEQUEL* je nastao iz potrebe za manipuliranjem i preuzimanjem podataka koji su se skladištili u *IBM* bazama podataka.[7]




Slika 7. Podvrste SQL jezika[8]

[7] *Usp.* Wikipedia, SQL, Dostupno na: https://sh.wikipedia.org/wiki/SQL, (04.11.2017.)

[8] Tech-Recipes, *DML, DDL, DCL and TCL Statements in SQL with Examples*, Dostupno na: http://www.tech-recipes.com/rx/55356/dml-ddl-dcl-and-tcl-statements-in-sql-with-examples/, (13.12.2017.).

### 1.3.1. DML

DML (eng. *Data Manipulation Language*) je jezik koji u bazama podataka služi za brisanje, ažuriranje i umetanje podataka. DML manipulira bazom podataka pomoću sljedećih naredbi:

- UPDATE – naredba za ažuriranje podataka u bazi podataka
- INSERT – naredba za umetanje podataka u bazu podataka
- DELETE – naredba za brisanje podataka u bazi podataka

Primjer 1. INSERT

```
INSERT INTO Osoba (Ime, Prezime) VALUES ('Ivan', 'Ivić')
```

Navedeni primjer upisuje ime i prezime (Ivan Ivić) u tablicu *Osoba.*

Primjer 2. UPDATE

```
UPDATE Osoba SET Ime='Marko', Prezime='Markić' WHERE
OsobaID = 3
```

Navedeni primjer ažurira tablicu *Osoba* na način da će izmijeniti već postojeće podatke novima za osobu čiji je identifikacijski broj (OsobaID) tri.

Primjer 3. DELETE

```
DELETE * FROM Osoba
```

Navedeni primjer briše sve podatke iz tablice *Osoba*.

```
DELETE FROM Osoba WHERE OsobaID = 3
```

Navedeni primjer briše osobu sa identifikacijskim brojem tri iz tablice *Osoba*.

### 1.3.2. DQL

DQL (eng. *Data Query Language*) je jezik koji služi za dohvaćanje podataka unutar baze podataka. DQL koristi naredbu *SELECT* za dohvaćanje podataka. Sintaksa *SELECT* naredbe:

```
SELECT
   INTO
   FROM
      WHERE
   GROUPBY
      HAVING
   ORDER BY
```

**ALL** (*) specificira da naredba SELECT označi sve podatke iz tablice označavajući pri tomu i dvostruke podatke

```
SELECT * FROM Osoba
```

**DISTINCT** specificira da naredba SELECT označi sve podatke isključujući pri tomu dvostruke podatke

```
SELECT DISTINCT Ime FROM Osoba
```

**INTO** kreira novu tablicu, identičnu onoj iz koje SELECT naredba povlači podatke, te prikazuje dobivene rezultate u novoj tablici kao rezultat

```
SELECT * INTO NovaTablica FROM Osoba
```

**FROM** specificira mjesto tj. tablicu, poglede i spojene (INNER) tablice iz kojih naredba SELECT povlači podatke

```
SELECT * FROM Osoba
```

**WHERE** specificira uvjete koje naredba SELECT mora proći i zadovoljiti da bi dohvatila i prikazala tražene rezultate a uvjeti su: AND, OR i NOT

**BETWEEN** specificira interval vrijednosti dvaju uvjeta koji određuju koje rezultate će naredba SELECT dohvatiti i prikazati

```
SELECT * FROM Osoba WHERE ID BETWEEN 1 AND  9
```

**IN** specificira listu vrijednosti koje će naredba SELECT dohvatiti i prikazati

```
SELECT * FROM Osoba WHERE ID IN (1, 5, 9)
```

**GROUP BY** specificira grupe u koje se raspoređuju izlazni rezultati, a grupiranje se provodi prema nazivu kolone ili prema izrazu, u grupu se povezuju kolone koje imaju iste vrijednosti

```
SELECT * FROM Osoba GROUP BY Ime
```

**HAVING** specificira uvjete koji će se provoditi nad GROUP BY dijelom naredbe nakon grupiranja

```
SELECT Ime, COUNT (*) FROM Osoba GROUP BY Ime HAVING
COUNT (*) > 1
```

**ORDER BY** sortira izlazne rezultate naredbe prema zadanim kolonama, a može sortirati podatke od najmanje vrijednosti prema najvećoj (ASC), te od najveće vrijednosti prema najmanjoj (DESC), Null vrijednost se uzima kao najmanja vrijednost

```
SELECT * FROM Osoba ORDER BY Ime
```

### 1.3.3. DDL

DDL (eng. *Data Definition Language*) je jezik koji se koristi za kreiranje, izmjene i brisanje objekata u bazi podataka. DDL Koristi sljedeće naredbe za manipuliranje bazom podataka:

- CREATE – naredba pomoću koje se kreira nova tablica, baza podataka i ostalo
- ALTER – naredba pomoću koje se uređuje postojeća tablica, te opis retka tablice
- DROP – naredba pomoću koje se briše postojeći objekt u bazi podataka

Primjer 1. CREATE LOGIN

```
CREATE LOGIN [Korisnik] WITH
   PASSWORD = N'PRIMJER',
   DEFAULT_DATABASE = [Osoba],
   CHECK_EXPIRATION = OFF, CHECK_POLICY = OFF
GO
```

Primjer 2. CREATE DATABASE

```
CREATE DATABASE [Osoba] ON PRIMARY
     (NAME = N' Osoba', FILENAME = N' C:\MSSQL\Osoba.mdf',
     SIZE  = 3072KB, FILEGROWTH = 1024KB )
LOG ON
     ( NAME = N'Osoba_log', FILENAME =
     N'C:\MSSQL\Osoba_log.ldf' ,
     SIZE =  1024KB, FILEGROWTH = 10%)
GO
```

Primjer 3. CREATE TABLE

```
CREATE TABLE Osoba (
     OsobaID int IDENTITY,
     Ime nvarchar(100),
     Prezime nvarchar(100),
     Datum_Rođenja datetime,
```

```
     PRIMARY KEY (OsobaID))
```

Primjer 4. CREATE PROCEDURE

```
CREATE PROCEDURE CheckLogin
     @Email nvarchar(50),
     @Lozinka nvarchar(50)
AS
BEGIN
     SELECT Ime, Prezime FROM Osoba WHERE Email = @Email
     AND Lozinka = @Lozinka
END
```

Primjer 5. CREATE FUNCTION

```
CREATE FUNCTION Dob
     (@KorisnickoIme nvarchar(20) )
RETURNS int
AS
BEGIN
     DECLARE @DatumRodjenja datetime
     DECLARE @ReturnDob int
     SELECT @DatumRodjenja=DatumRodjenja FROM Osoba
          WHERE KorisnickoIme=@KorisnickoIme
          SELECT @ReturnDob=
DATEDIFF(Year,@DatumRodjenja,GetDate()) )
          RETURN @ReturnDob
END
```

Primjer 6. ALTER TABLE

```
a) ALTER TABLE Osoba ADD Nadimak nvarchar(100)
```

Navedeni primjer u postojeću tablicu dodaje novu kolonu *Nadimak.*

```
b) ALTER TABLE Osoba ALTER COLUMN Nadimak nvarchar(200)
```

Navedeni primjer uređuje količinu znakova koje možemo upisati u kolonu *Nadimak.*

c) ALTER TABLE Osoba DROP COLUMN Nadimak

Navedeni primjer briše iz postojeće tablice kolonu *Nadimak*.

Primjer 7. DROP TABLE

```
DROP TABLE Osoba
```

Navedeni primjer briše tablicu *Osoba* iz baze podataka.

### 1.3.4. DCL

DCL (eng. *Data Control Language*) je jezik za rad s rolama, dozvolama te kontrolira pristup bazi podataka. DCL koristi sljedeće naredbe za upravljanje bazom podataka:

- GRANT – naredba pomoću koje se dozvoljava nekom korisniku da odrađuje akcije (read/write) na određenom/-im objektu/-ima u bazi podataka
- REVOKE – naredba pomoću koje se korisnike drži dalje tj. ne dozvoljava se korisnicima uređivanje objekata u bazi podataka

Primjer 1. GRANT

```
GRANT [privilege]
ON [object]
```

```
TO [user]
```

Primjer 2. REVOKE

```
REVOKE [GRANT OPTION FOR] [permission]
ON [object]
FROM [user]
```

### 1.3.5. TCL

TCL (eng. *Transaction Control Language*) je jezik koji služi za upravljanje transakcijama u bazi podataka, a koristi sljedeće naredbe za upravljanje bazom podataka:

- BEGIN – Oznaka za početak transakcije
- COMMIT – Oznaka za potvrđivanje akcija unutar transakcije
- ROLLBACK – Oznaka za poništavanje akcija unutar transakcije

Primjer 1. Commit Transaction

```
BEGIN TRANSACTION
     SELECT * FROM Osoba
     DELETE FROM Osoba
     INSERT INTO Osoba    (ID, Ime) (1, 'Ivan')
     SELECT * FROM Osoba
COMMIT TRANSACTION
```

Primjer 2. Rollback Transaction

```
BEGIN TRANSACTION
     SELECT * FROM Osoba
     DELETE FROM Osoba
     INSERT INTO Osoba      (ID, Ime) (1, 'Ivan')
     SELECT * FROM Osoba
ROLLBACK TRANSACTION
```

# 2. WEB SERVISI

Web servisi se definiraju kao softverski sustavi dizajnirani na način koji omogućuje interoperabilne interakcije između strojeva preko računalne mreže. Sadrži sučelje koje je opisano u obliku koji je razumljiv računalu (točnije u jeziku WSDL – *Web Services Description Language*).

Začetkom web servisa smatra se projekt ARPANET (eng. *Advanced Research Projects Agency Network*) početkom 1970-ih. Zadatak je bio da se na jednom računalu izvršavaju zadaci dok mu drugo računalo šalje podatke pomoću daljinskog doziva procedure (RPC)[9]. Postoje tri vrste web servisa:

- RPC
- SOAP
- REST

## 2.1. RPC

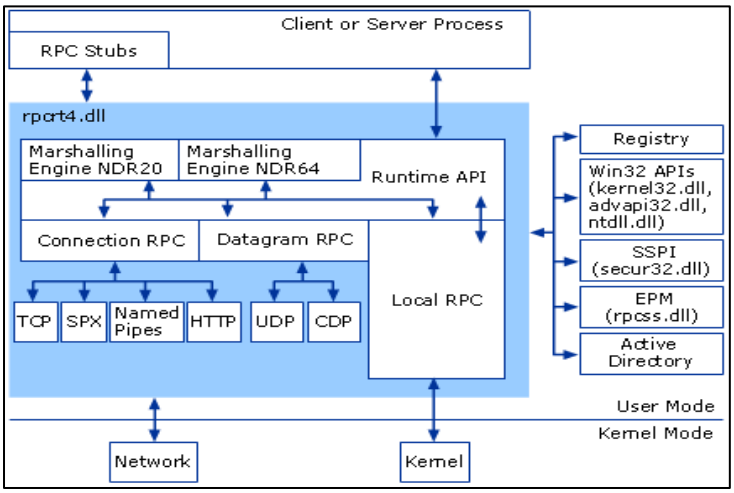
RPC je tehnologija za kreiranje distribuiranih klijent/server programa, te komunikacijska tehnika koja dozvoljava klijentu i serveru komunikaciju i razmjenu podataka. Klijent šalje zahtjev nekom udaljenom serveru da izvrši određenu proceduru, nakon što server izvrši proceduru vraća poruku s rezultatom procedure. Ovaj proces komunikacije može biti na samom računalu, na lokalnoj mreži (LAN) ili na Internetu. Zahvaljujući RPC-u važni proceduralni kodovi mogu egzistirati na različitim računalima što je iznimno važno za distribuirane aplikacije.[10]

Zamjenom protokola i komunikacijskih metoda sa standardiziranim sučeljem, RPC je stvoren za uspostavljanje i održavanje komunikacije između klijenta i procesa na nekom

[9] RPC – Remote Procedure Call

[10] *Usp.* TechNet, *How RPC Works*, Dostupno na: https://technet.microsoft.com/en-us/library/cc738291(v=ws.10).aspx, (03.08.2017.).

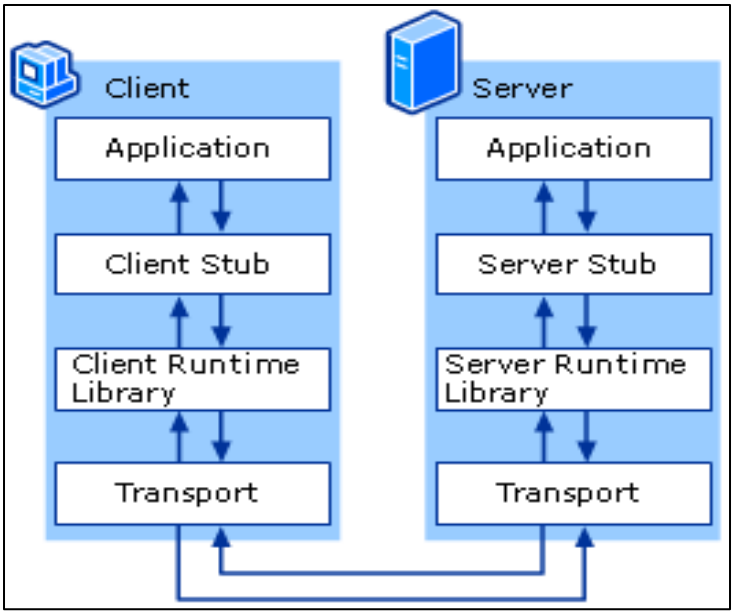
serveru, a funkcije koje su dio RPC-a mogu biti dohvaćene od strane bilo kojega programa koji komunicira s njime koristeći klijent/server metodologiju.




Slika 8. Arhitektura RPC-a[11]

Sastavni dijelovi tj. komponente RPC-a olakšavaju klijentu pozivanje procedure sa udaljenog serverskog programa, klijent i server imaju svoje vlastite adrese, tj. svaki od njih ima vlastitu memoriju koju dodjeljuju podacima koje koristi procedura. RPC proces počinje na strani klijenta, gdje klijentska aplikacija poziva lokalni skup procedura umjesto implementiranja procedura putem koda. Skup se kompajlira i označava putem klijentske aplikacije tijekom razvoja. Umjesto zadržavanja koda koji implementira udaljenu proceduru, klijentski skup kodova vraća potrebne parametre od strane klijentskog adresnog prostora i dostavlja ga klijentovom CRL-u (eng. *Client Runtime Library*). CRL zatim prevodi parametre kako je zadano u standardni NDR (eng. *Network Data Representation*) format za prijenos na server (Slika 9.).

[11] *Isto*




Slika 9. RPC proces[12]

Klijentski dio koda poziva funkciju u RPC-ovom CRL-u da pošalje zahtjev i parametre serveru. Ukoliko je server lociran na istom računalu kao i klijent, CRL može koristiti lokalnu RPC (LRPC) funkciju i propustiti RPC-ov zahtjev prema Windows sustavu za transport prema serveru. Ako je server lociran na udaljenom računalu, CRL specificira potrebni transportni protokol (Slika 10.) i propušta RPC u mrežu za transport prema serveru.

[12] *Isto.*

| PROTOCOL | RPC TYPE |
|---|---|
| **TRANSMISSION CONTROL PROTOCOL (TCP)** | Connection–oriented |
| **SEQUENCED PACKET EXCHANGE (SPX)** | Connection–oriented |
| **NAMED PIPE** | Connection–oriented |
| **HTTP** | Connection–oriented |
| **USER DATAGRAM PROTOCOL (UDP)** | Connectionless |
| **CLUSTER DATAGRAM PROTOCOL (CDP)** | Connectionless |

Slika 10. Mrežni protokoli koje podržava RPC

Kada server zaprimi RPC zahtjev, neovisno da li je lokalno ili sa udaljenog klijenta, serverski RPC prihvaća zahtjev i poziva proceduru unutar serverskoga skupa. Serverski dio koda preuzima parametre iz mreže te ih konvertira iz mrežnog formata u format koji je potreban serveru, te poziva proceduru na serveru. Udaljena procedura se pokreće, generira izlazne parametre i vraća vrijednost, a kad se udaljena procedura završi podaci se vraćaju klijentu na sličan način kako su i poslani serveru. Klijent završava proces tako što prihvaća podatke sa mreže.

## 2.2. SOAP

SOAP (eng. *Simple Object Access Protocol*) je protokol za izmjenjivanje strukturiranih informacija pri implementaciji web servisa u računalnu mrežu, a svrha mu je uvođenje i pružanje ekstenzibilnosti, neutralnosti i neovisnosti. Kao format za razmjenu poruka koristi XML (eng. *Extensible Markup Language*), oslanja se na aplikacijski protokol, najčešće HTTP (eng. *Hypertext Transfer Protocol*) ili SMTP (eng. *Simple Mail Transfer Protocol*) protokol za razmjenu poruka. SOAP dozvoljava izvođenje procesa na različitim operacijskim sustavima, kao što su npr. Windows i Linux, za komunikaciju koristeći XML. Budući su web protokoli kao što je HTTP instalirani na svim operacijskim

sustavima SOAP dozvoljava klijentima pozivanje web servisa i primanje odgovora neovisno o jeziku i platformi.[13]

SOAP je dizajniran kao objektno-pristupni protokol 1998. godine za Microsoft, a tvorci su: *Dave Winer, Bob Atkinson, Mohsen Al-Ghosein i Don Box*. Nakon što je predstavljen, SOAP je postao dio većeg sustava web servisa baziranog na WSDL (eng. *Web Services Description Language*).

SOAP posjeduje tri glavne karakteristike:

- Proširivost (engl. *Extensibility*)
- Neutralnost (engl. *Neutrality*) – SOAP može obavljati operacije nad bilo kojim protokolima kao što su: HTTP, SMTP, TCP, UDP itd.
- Nezavisnost (engl. *Independence*)

SOAP arhitektura se sastoji od nekoliko slojeva specificiranih za:

- Formate za poruke
- Uzorci poruka za razmjenu (engl. *Message Exchange Patterns* )
- Modele procesiranja poruke
- Ekstenzibilnost protokola

[13] *Usp*. Wikipedia, SOAP, Dostupno na: https://en.wikipedia.org/wiki/SOAP, (03.08.2017.)

Primjer tijela SOAP poruke:

```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="http://www.google.com"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
      <soap:Header>
      ...
      </soap:Header>
      <soap:Body>
            ...
            <soap:Fault>
            ...
            </soap:Fault>
      </soap:Body>
</soap:Envelope>
```

SOAP naredbe i protokoli:

- SOAP: skup pravila za razmjenu informacija između SOAP pošiljatelja i SOAP primatelja
- SOAP *Nodes*: fizičke ili logičke mašine sa procesorskim jedinicama koje se koriste za prevođenje, primanje i procesiranje SOAP poruka
- SOAP *protocol binding*: SOAP poruka mora djelovati u skladu sa ostalim protokolima kako bi bila poslana preko mreže
- SOAP *message*: predstavlja informacije koje se razmjenjuju između dva SOAP procesa
- SOAP *envelope*: element XML poruke koji ju identificira kao SOAP poruku
- SOAP *header*: skup jednog ili više *header* (zaglavlja) blokova označenih kod svih SOAP odašiljača
- SOAP *body*: sadrži tijelo poruke odaslane u SOAP odašiljač, interpretacija i procesiranje SOAP *body*-a je definiran od strane *header* blokova

- SOAP *fault*: u slučaju da SOAP *nodes* ne uspije procesuirati SOAP poruku, daje informaciju o grešci u SOAP *fault* element
- SOAP *sender*: proces koji emitira SOAP *message*
- SOAP *reciever*: proces koji prima SOAP *message*
- SOAP *message path*: putanja koja sadrži sve procese koje SOAP *message* označi da su dostigli zadani proces

Kao standardni format za slanje poruka izabran je XML zbog svoje raširenosti diljem svijeta, te zbog korištenja od strane velikih korporacija. Mnogo besplatno dostupnih alata olakšava tranziciju k SOAP-u. Duža XML sintaksa može biti i prednost ali i poteškoća, bolje je čitljiva za ljude te olakšava otkrivanje grešaka, no sve to dovodi do usporavanja brzine procesiranja.[14]

Primjer poruke o pogrešci (engl. *Fault message*):

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/en
velope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:
xsd="http://www.w3.org/1999/XMLSchema">
      <SOAP-ENV:Body>
         <SOAP-ENV:Fault>
         <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
         <faultstring xsi:type="xsd:string">
            Failed to locate method (GetTutorialID) in class (GetTuto
rial)
         </faultstring>
    </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

[14] *Usp*. Wikipedia, SOAP, Dostupno na: https://en.wikipedia.org/w/index.php?title=SOAP&oldid=790405857, (20. 08. 2017.).

Prednost SOAP-a je što se može koristiti s bilo kojim transportnim protokolom, često se koristi HTTP kao transportni protokol ali može se koristiti i neki drugi transportni protokol kao npr. SMTP i JMS. Kada se koristi zajedno sa HTTP-om, SOAP lako prolazi kroz postojeće zaštitne zidove (engl. *Firewall*) i posrednike (engl. *Proxy*)[15]

Kada se podnese zahtjev za SOAP web uslugu, odgovor koji se vraća može biti jedan od dva obrasca koji su uspješan odgovor ili pogreška. Kada se generira uspjeh, odgovor s poslužitelja će uvijek biti SOAP poruka. SOAP poruka o pogrešci se sastoji od sljedećih elemenata:

- **<faultCode>-** Predstavlja kod koji označava dio koda s greškom, kod pogreške može biti bilo koji od nižih vrijednosti:
- **<faultString>-** tekstualna poruka koja daje detaljan opis pogreške
- **<faultActor> (Optional)-** Tekstni niz koji označava tko je prouzročio grešku
- **<detail>(Optional)-** Element za specifične aplikacijske poruke o pogreški, aplikacija može imati specifičnu poruku o pogrešci za različite scenarije poslovne logike

U primjeru poruke o pogrešci koja je prikazana ranije, pogreška se generira ukoliko klijent pokuša koristiti metodu *TutorialID* u klasi *GetTutorial*. Poruka o pogrešci se generira u slučaju kada metoda ne postoji u definiciji klase. Kada izvršimo kod iz primjera, prikazat će se greška „Failed to locate method (GetTutorialID) in class (GetTutorial)“.[16]

## 2.3. REST

REST je vrsta resursno orijentirane arhitekture za koju se ne kreiraju procedure već resursi kojima se pristupa, a omogućava rad sa velikim brojem klijenata tj. korisnika i samim time zadovoljava svojstvo skalabilnosti (engl. *scalability*). REST je baziran na

[15] *Usp*. Isto

[16] *Usp.* Guru99, *SOAP-Simple Object Access Protocol,* Dostupno na: https://www.guru99.com/soap-simple-object-access-protocol.html, (13.12.2017.).

klijent serverskom modelu i on predstavlja skup pravila koji opisuju kako se standardi mogu koristiti za razvoj web aplikacija. Poznat je i pod oznakama ROA te RESTful, a sam naziv potječe od engleskog naziva *Representational state*. REST omogućava internetsku komunikaciju pomoću svih poznatih programskih jezika (C#, C/C++, PHP, Java itd.), te omogućuje rad na svim važnijim operacijskim sustavima (Windows, Mac, Linux itd.).

Prema *Royu Fieldingu*, REST treba zadovoljiti sljedeće kriterije:

- *Općenitost sučelja (engl. Generality Of Interfaces)* – sve web aplikacije trebaju implementirati svoje sučelje na isti način. Dijeljenjem istih zaključaka druge aplikacije znaju kako pozvati vašu, a mi znamo kako pozvati druge aplikacije.
- *Samostalni raspored komponenata (engl. Indepedent Deployment of Components)* – jednom kada se aplikacija i REST sučelje implementiraju i provedu, mora biti sposobna za implementaciju i re-implementaciju, i izvršavanje bilo kojeg REST sučelja bez prepisivanja ili mijenjanja postojećeg
- *Enkapsuliranje naslijeđenih sustava (engl. Encapsulate Legacy Systems)* – postojeće aplikacije koje nisu implementirane u REST-u mogu biti pomiješane s REST sučeljem, stvorivši pritom REST aplikaciju
- *Posredničke komponente za smanjenje latencije interakcije (engl. Intermediary Components To Reduce Interaction Latency)* – budući REST upotrebljava sučelje, implementiraju se ili dodaju komponente u vidu slojeva, kao što su fizički serveri za rad sa klijentskim resursima
- *Naglašavanje skalabilnosti interakcije komponenti (engl. Emphasizing The Scalability Of Component Interactions)* – suprotan je prethodnom kriteriju
- *Provođenje sigurnosti (engl. Enforce Security)* – razmjenjivanje informacija preko interneta može biti opasno, hakeri to mogu upotrijebiti da probiju sistem. REST principi eliminiraju mnoge od tih rizika

Sastavni dijelovi (koncepti) prema *Royu Fieldingu*:

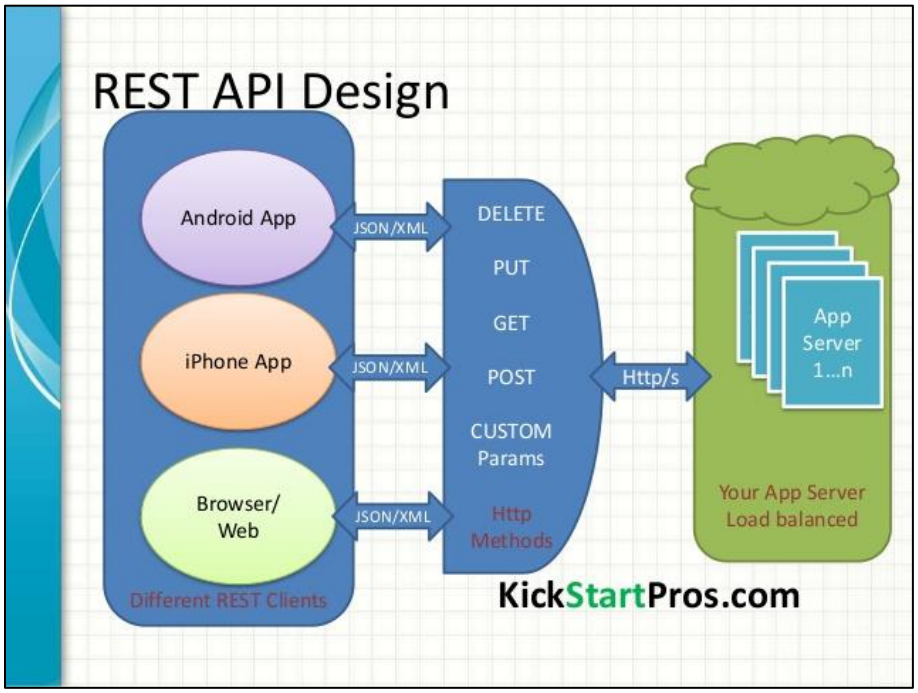
- *Izvor (engl. Resource)* – logički izvor je bilo koji koncept koji se može adresirati i označiti koristeći se globalnim identifikatorima. Obično se svim izvorima može pristupiti sa URI-em pri implementaciji REST-a preko HTTP-a
- *Server* – logički server je mjesto gdje su izvori smješteni, zajedno bez miješanja podataka unutar skladišnog prostora
- *Klijent (engl. Client)* – logički klijenti traže zahtjeve logičkom serveru za pokretanje operacije nad njegovim izvorima. Na primjer, klijent može zahtijevati ispis izvora, stvaranje izvora, osvježenje izvora novim podacima, brisanje izvora i ostalo
- *Zahtjev i odgovor (engl. Request and Response)* – komunikacija između klijenta i servera je organizirana sa zahtjevom od klijenta ka serveru, a odgovor na zahtjev od servera ka klijentu
- *Predstavljanje (engl. Representation)* – reprezentacija je dokument koji prezentira trenutni status izvora

Kako bi se izvršile operacije na izvorima, HTTP se koristi s više tipova: GET, PUT, POST, DELETE, CONNECT, HEAD, itd. Dok REST koristi samo neke od njih, a to su: GET, PUT, POST i DELETE.[17]

- GET – klijent može zatražiti status izvora pozivajući HTTP GET zahtjev serveru, koristeći URI izvor. REST zahtjeva da operacija ne prouzrokuje bilo kakvu posljedicu na izvorišni status
- PUT – služi za kreiranje nove metode. Ako je izvor ranije kreiran tada se on neće ponovno rekreirati.
- POST – REST zahtjeva POST zahtjev klijenta za ažuriranje odabranog izvora sa informacijama koje dobiva od klijenta ili da ga kreira ukoliko ne postoji

[17] *Usp.* Spring,*Understanding REST*, Dostupno na: https://spring.io/understanding/REST, (07.12.2017.).

- DELETE – ova operacija uklanja izvor nepovratno




Slika 11. Primjer REST API dizajna[18]

[18] Devi Kiran G, *REST API Design*, Dostupno na: https://www.slideshare.net/DeviKiranGonuguntla/rest-api-design, (13.12.2017.).

## 2.4. SOAP ili REST?

SOAP je kreirala tvrtka Microsoft i postoji mnogo duže od REST-a, što mu daje određenu prednost, no međutim REST se pokazao kao izuzetno jednostavniji od SOAP-a prilikom pristupa web servisima.

Budući da koristi HTTP, REST nudi brojne pogodnosti u odnosu na SOAP:

- REST nudi veću mogućnost izbora formata
- Zbog povezivanja sa JSON-om, REST se ocjenjuje lakšim za korištenje
- Zahvaljujući JSON-u, REST nudi bolju podršku klijentima
- REST nudi bolje performanse, brži je te koristi manje mrežnom prostora[19]

Prednosti SOAP-a u odnosu na REST:

- SOAP pruža dodatne garancije za sigurnost podataka
- SOAP lakše prolazi kroz zaštitne zidove i *proxie*
- U nekim slučajevima dizajniranje SOAP servisa može biti manje kompleksno nego u slučaju REST-a[20]

Najbolji protokol za kreiranje servisa je onaj koji najbolje odgovara nekoj organizaciji ili klijentu, a najviše se u zadnje vrijeme koriste REST zajedno s JSON formatom zbog svoje jednostavnosti za korisnike ali i razvojne programere, te iz razloga što koristi manje mrežnog prostora.

[19] *Usp.* Angela Stringfellow,SOAP vs. REST: The Differences and Benefits Between the Two Widely-Used Web Service Communication Protocols, (24.08.2017.).

[20] *Usp*. *Isto*

# 3. WCF

WCF je akronim za *Windows Communication Foundation* i on predstavlja sučelje, odnosno alat za izgradnju servisno-orijentiranih aplikacija na Windowsu. Servisi se mogu izgrađivati i bez WCF-a, ali sa WCF-om je to mnogo brže i lakše. WCF je Microsoft implementacija seta standarda koji definiraju interakciju između dva servisa.[21]

U WCF-u je svakom servisu dodijeljena posebna adresa, koja pruža dva važna elementa, a to su lokacija servisa i transportni protokol koji se koristi za komunikaciju sa servisom. WCF podržava sljedeće transportne protokole:

- HTTP/HTTPS – je protokol za komunikaciju između poslužitelja i klijenta na webu, HTTPS je sigurnija verzija HTTP protokola koja koristi SSL/TLS za zaštitu i skrivanje prometa.
- TCP – kratica od engleskog naziva *Transmission Control Protocol*, njenim korištenjem aplikacija na nekom hostu umreženom u računalnu mrežu kreira virtualnu konekciju prema drugom hostu i putem veze prenosi podatke. TCP pouzdano isporučuje podatke od pošiljatelja prema primatelju, osim toga pruža mogućnost višestrukog istovremenog povezivanja prema jednoj aplikaciji od strane više klijenata, a najčešće se koristi za web poslužitelje.

## 3.1. Contracts

U WCF-u svi servisi su izloženi ugovorima (eng. *contracts*), a oni predstavljaju neutralnu platformu i standardni put za opisivanje što servis radi. Neki od WCF ugovora su [22]:

[21] *Usp.* Lowy, J., *Programming WCF Services*, III izdanje, O'Reilly Media, str 1., 2010.,(21.09.2017.)

[22] *Usp*. *Isto* str. 7

*Service contract*

Veže zajedno više povezanih operacija u jednu funkcionalnu jedinicu. Može definirati *service-level* postavke kao što je *namespace* nekog servisa te druge postavke. U većini slučajeva *service contract* se definira kreiranjem sučelja (interface) u programskom jeziku po vlastitom izboru te pozivanjem *ServiceContractAttribute* atributa unutar sučelja. Definira koje operacije klijent može izvršiti na servisu, a definira se i implementira na sljedeći način:

```
[ServiceContract]
interface IMyContract
{
    [OperationContract]
    string MyMethod(string text);
    //Will not be part of the contract
    string MyOtherMethod(string text);
}
class MyService : IMyContract
{
    public string MyMethod(string text)
    {
        return "Hello" + text;
    }
    public string MyOtherMethod(string text)
    {
        return "Cannot call this method over WCF";
    }
}
```

*Data contract*

Definira koji tipovi podataka mogu proći prema servisu te od servisa, WCF implicitno definira ugovor za ugradnju tipova podataka kao što su *int* i *string*, a možemo i sami definirati eksplicitni ugovor za tipove podataka po osobnoj potrebi. Tip podatka se može upotrijebiti u bilo kojem dijelu poruke, na primjer kao parametar ili povratni (eng. *return*) tip podatka. Ukoliko servis koristi samo jednostavne tipove podataka nema potrebe za korištenjem *data conracta.*

*Fault contract*

Definira koje se pogreške mogu pojaviti na servisu te kako ih servis rješava i obavještava klijenta.

*Message contract*

Opisuje format poruke, na primjer, deklarira koji će element poruke ići u zaglavlje a koji u tijelo poruke, koju razinu zaštite će se pružiti elementima poruke.

*Operation contract*

*Operation contract* definira parametre i tip podatka kojeg vraća neka operacija. Kada kreiramo izgled koji definira *service contract*, pokrećemo *operation contract* pozivanjem *OperationContractAttribute* atributa za svaku definiciju metode koja je dio ugovora (contract).

## 3.2. Hosting

Svaki WCF servis mora biti postavljen u Windows proces koji se naziva *host process*, jedan *host process* može podržati nekoliko servisa, a sam servis može biti postavljen na više *host processes.* Host može biti pružen od strane *Internet Information Services* (IIS) te od strane *Windows Activation Service* (WAS).

*IIS Hosting*

Glavna prednost postavljanja servisa na Microsoft-ov IIS web server je da *host process* se pokreće automatski pri prvom klijent-ovom zahtjevu, a IIS upravlja životnim ciklusom *host processa.* Objavljivanje na IIS-u je slično objavljivanju klasičnog ASMX web servisa, potrebno je kreirati virtualni direktorij u IIS-u i stvoriti *.svc* datoteku.

Datoteka *.svc* se koristi za identifikaciju servisnog koda unutar datoteka i klasa.[23]

Primjer *.svc* datoteke:

```
<%@   ServiceHost
Language = „C#“
Debug = „true“
CodeBehind = „/App_Code/MyService.cs“
Service = „MyService“  %>
```

Umjesto definiranja *.svc* datoteke, možemo direktno u aplikacijskoj *web.config* datoteci pri odjeljku, *serviceHostingEnviroment*, definirati tip servisa i informacije o njegovoj adresi. Pri tom možemo definirati onoliko servisa koliko želimo kao na sljedećem primjeru[24]:

[23] *Usp*. *Isto* str. 12

[24] *Usp*. *Isto* str. 13

```
<system.serviceModel>
      <serviceHostingEnviroment>
            <add relativeAddress = „MyService.svc“ service =
            „MyNamespace.MyService“/>
            <add relativeAddress = „MyOtherService.svc“ service =
            „MyOtherService“/>
      </serviceHostingEnviroment>
      <services>
            <service name = „MyNamespace.MyService“>
                  …
            </service>
            <service name = „MyOtherService“>
                  …
            </service>
      </services>
</system.serviceModel>
```

*WASS Hosting*

WASS je alat koji omogućava objavljivanje (podizanje) web stranica kao i olakšano objavljivanje servisa, omogućavajući pri tomu korištenje bilo kojeg transportnog protokola. WASS je dizajniran da postupak objavljivanja izgleda identično postupku objavljivanja servisa na IIS-u. Potrebno je pravilno popuniti *.svc* datoteku ili pružiti pravilne informacije unutar *config* datoteke. Budući je WAS sistemski servis nije potrebno prethodno pokrenuti proces objavljivanja servisa. Kada stigne prvi upit od strane nekog klijenta, WASS ga presreće, pokreće proces objave našeg servisa i prosljeđuje mu upit.[25]

[25] *Usp*. *Isto* str. 19

### 3.3. Bindings

*Bindings* su objekti koji se koriste za specificiranje komunikacijskih detalja koji su potrebni za komuniciranje WCF servisnog endpointa sa drugim endpointima. Svaka krajnja točka WCF servisa zahtijeva *binding* kako bi se što bolje specificirala. *Binding* mora specificirati transportni protokol (npr. HTTP ili TCP) koji će se koristiti u komunikaciji WCF servisa.

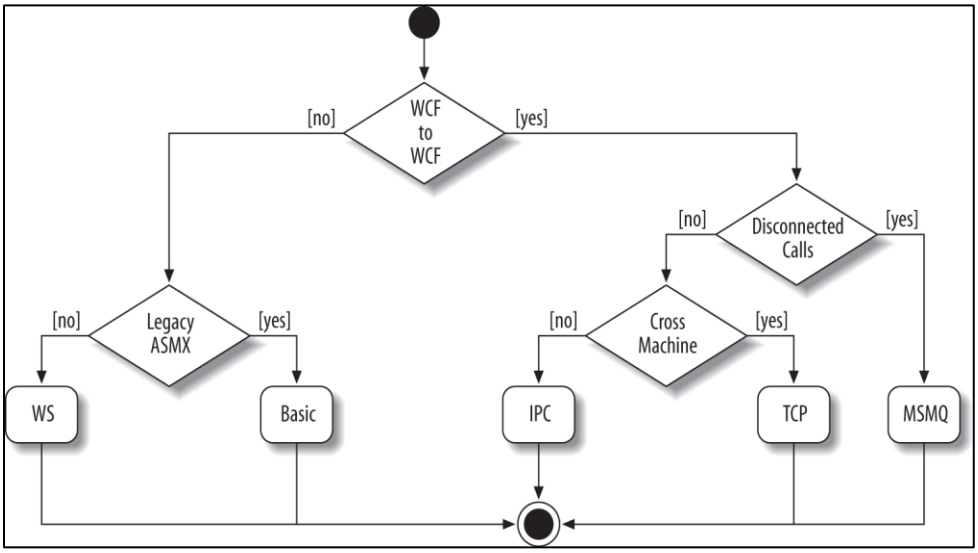
Informacije u *bindingu* mogu biti veoma jednostavne ali i veoma kompleksne, uobičajeno *binding* specificira samo transportni protokol koji se mora koristiti kako bi se povezao sa krajnjom točkom. *Binding* je dosljedan skup pravila odnosno mogućnosti koji se odnose na transportne protokole, komunikacijski uzorak, sigurnost, prevođenje poruke, pouzdanost i interoperabilnost. Sve što trebamo je odrediti željeni scenarij za naš servis, za koji zatim WCF donosi samostalno odluku za sve aspekte komunikacije. *Bindings* omogućavaju korištenje iste servisne logike na različitim primjerima tj. projektima.[26]

Najčešće korišteni *bindings*:

- *Basic binding:* dizajniran za predstavljanje WCF servisa kao nasljednog ASMX web servisa kako bi stariji klijenti mogli raditi sa novim servisom. *Basic binding* pruža izgled nasljednog web servisa na mreži koji komunicira preko osnovnog web servis profila. Kada se koristi od strane klijenta, ovaj *binding* omogućava novim WCF klijentima da rade sa starim ASMX servisima
- *TCP bindings:* koristi TCP za komunikaciju između računala na internetu. Podupire različite mogućnosti, uključujući pouzdanost, transakcije, sigurnost, a i optimiziran je za komunikaciju između WCF servisa. Kao rezultat zahtijeva i klijenta i servis za korištenje WCF-a
- *IPC binding:* najsigurniji *binding* budući ne može prihvatiti poziv izvan računala. Podupire različite mogućnosti slično kao i *TCP binding*

[26] *Usp. Isto* str. 24

- *Web Service (WS) binding:* koristi HTTP ili HTTPS protokol kao transport te pruža mnoštvo mogućnosti na internetu kao što su: pouzdanost, transakcije te sigurnost. Kreiran je za interoperabilnost sa bilo kojom stranom koja podupire WS standarde
- *MSMQ binding:* koristi MSMQ za transport te nudi podršku za odbijene pozive

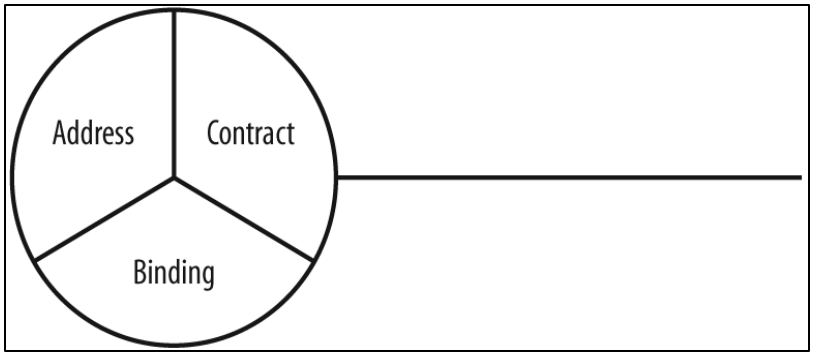



Slika 12. Odabir binding-a[27]

### 3.4. Endpoints

Svaki servis sadrži adresu koja definira gdje se servis nalazi, *binding* koji definira kako komunicirati sa servisom, te *contract* (ugovor) koji definira što servis radi. WCF pojednostavljuje odnose triju spomenutih dijelova servisa u formu koja se predstavlja kao *endpoint* (krajnja točka). *Endpoint* je dakle kombinacija adrese, *contracta i bindinga* (Slika 13.).

[27] Safari, *Programming WCF Services, 3rd Edition by Juval Lowy*, Dostupno na: https://www.safaribooksonline.com/library/view/programming-wcf-services/9781449382476/ch01s06.html, (07.12.2017.)




Slika 13. Endpoint

Svaki servis mora prikazati najmanje jedan *endpoint,* a svaki *endpoint* ima točno jedan *contract*. Svi *endpointi* nekog servisa imaju jedinstvene adrese, a svaki pojedinačni servis može prikazati više *endpointa*. Ovi *endpointi* mogu koristiti iste ili različite *bindinge* te mogu prikazati iste ili različite *contracte.* Između različitih *endpointa* nekog servisa ne postoje zajednički odnosi (relacije). *Endpointe* možemo urediti administrativno (koristeći *config* datoteku) ili programski.[28]

*Administrativno konfiguriranje Endpointa*

Administrativno konfiguriranje *endpointa* se vrši na način da se detalji *endpointa* postavljaju unutar *config* datoteke, kao na sljedećem primjeru:

[28] *Usp. Isto* str. 30

```
<system.serviceModel>
      <services>
            <service name = „MyNamesoace.MyService“>
                  <endpoint
                  Address = „http://localhost:8080/MyService“
                  Binding = „wsHttpBinding“
                  Contract = „MyNamespace.IMyContract“
                  />
            </service>
      </services>
</system.serviceModel>
```

Administrativno konfiguriranje se koristi u većini slučajeva iz razloga što pruža mogućnost jednostavnog mijenjanja adrese, *bindinga* i *contracta* servisa bez ponovnog *rebuilda(*ponovna izgradnja) servisa.

*Programsko konfiguriranje Endpoint-a*

Nasuprot administrativnom konfiguriranju *endpointa* imamo programsko konfiguriranje u kojemu umjesto popunjavanja *config* datoteke, potrebnim informacijama, popunjavamo *ServiceHost* instancu.

### 3.5. Behaviours

*Behaviours* je tip koji mijenja odnosno produžava (nadopunjuje) funkcionalnost servisa ili klijenta. Primjeri korištenja *Behavioursa*:

- Kontrolira da li su meta podaci objavljeni na servisu
- Dodaje sigurnosne dodatke servisu, kao što je autorizacija
- Zapisuje informacije o porukama
- Pokreće sve dodatne operacije kada je poruka primljena

## 3.6. Sigurnost

Kao i u drugim klijent-server aplikacijama, tako i u WCF-u, servis treba identificirati (eng. *authentication*) poziv i onoga tko stoji iza poziva, te treba ustvrditi koju razinu korisničkih prava isti ima prije nego što mu dopusti pristup važnim, osjetljivim informacijama. Važno je zaštititi odnosno osigurati poruke dok putuju između klijenta i servisa.

*Identifikacija (engl. Identification)*

Identifikacija je postupak koji provjerava i utvrđuje da je pozivač (eng. *caller*) servisa stvarno onaj kojim se i predstavlja. Klijent zahtijeva i identifikaciju servisa, što je posebno važno sa klijentima koji pozivaju preko Interneta zbog opasnosti od zlonamjernih virusa koji prate klijentov DNS servis i pokušava preoteti klijentov poziv.[29] WCF pruža mnogo identifikacijskih mehanizama a neki od njih su sljedeći:

- *Nema autentičnosti (engl. No authentication)*: Servis ne identificira onoga koji poziva, te dopušta pristup svima koji šalju poziv
- *Windows autentifikacija (engl. Windows authentication)*: Onaj tko šalje poziv servisu istovremeno pruža i svoju propusnicu (npr. karta ili token) koju servis koristi za provjeru Windowsa
- *Korisničko ime i lozinka (engl. Username and password)*: Onaj tko šalje poziv pruža servisu korisničko ime i lozinku, koje servis zatim koristi za identifikaciju istoga na Windows-u računu ili npr. bazi podataka
- *Prilagođeni mehanizam (engl. Custom mechanism)*: WCF dopušta programeru zamjenu ili unos novom mehanizma sa bilo kojim protokolom i identifikacijskim tipom

[29] *Usp.* Rahul Rajat Singh, *A Neginners Tutorial on Custom Forms Authentication in ASP.NET MVC Application*, Dostupno na: https://www.codeproject.com/Articles/578374/AplusBeginner-splusTutorialplusonplusCustomplusF, (07.12.2017.).

*Autorizacija*

Autorizacija je postupak u kojem se utvrđuje koja administrativna prava ima pozivač, tj. koje operacije su dopuštene da obavlja na servisu. Proces autorizacije pozivača se smatra uspješnim ukoliko se pokaže da je pozivač u stvarnosti onaj za kojeg se predstavlja, što znači da je autorizacija neodvojiva i nezamisliva bez procesa identifikacije. Kada se autorizira neka operacija, operacija objavljuje i zahtjeva da joj se može pristupiti samo s određenim pravima, koja servis provjerava da li postoje kod pozivača nakon čega mu dopušta ili odbija pristup servisu.

# 4. MVC

*Model-View-Controller* (MVC) opisuje način strukturiranja aplikacije, te odgovornost i suradnju svakog od dijelova MVC strukture. Ideja koju predstavlja MVC je da svaki dio koda ima svrhu, a te svrhe mogu biti različite. Dio koda sadržava i upravlja podacima, dio koda daje izgled aplikaciji, a dio koda određuje koje će funkcije imati aplikacija.[30]

## 4.1. Model

Model je dio MVC arhitekture koji sadrži podatke aplikacije i poslovnu logiku. Kako bi dodali novi model u projekt, desnim klikom na Model mapu, kliknemo *Add* te odaberemo novu klasu (eng. class). Zatim u novo kreiranu klasu dodajemo kod, slično kao u sljedećem primjeru.

[30] *Usp.* Pastor Pablo, *MVC for Noobs*, (24.03.2010.), Dostupno na: https://code.tutsplus.com/tutorials/mvc-for-noobs--net-10488, (04.12.2017.).

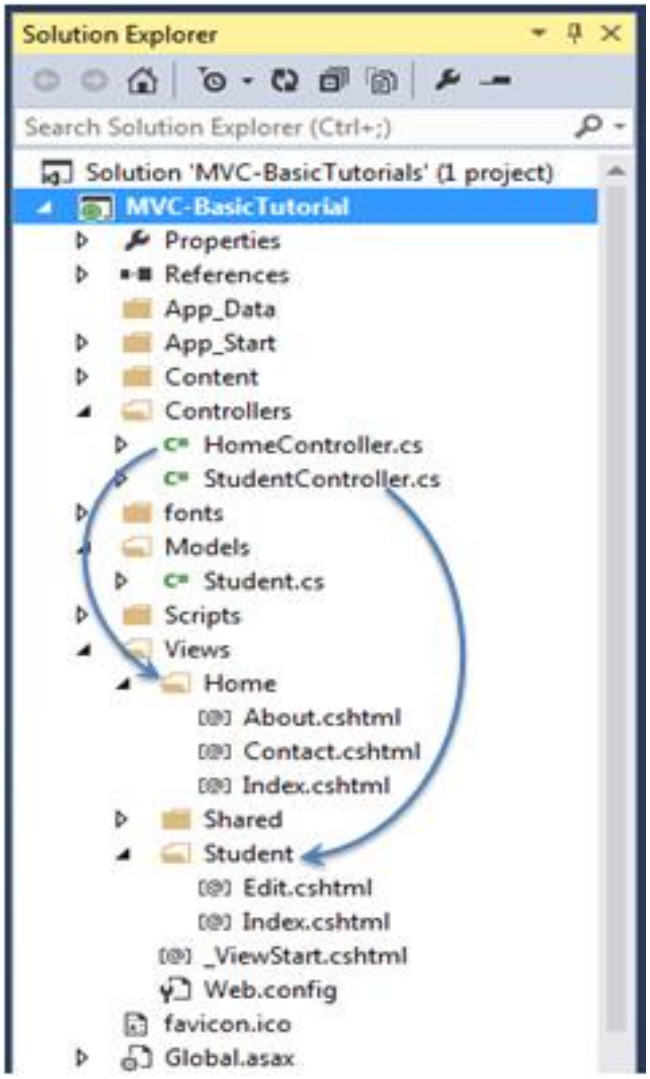
Primjer klase modela:

```
using System;
using System.Collections.Generic;

namespace PoslovniSistem.Models
{
     public partial class Artikl
     {
          public int ArtiklId { get; set; }
          public string Naziv { get; set; }
          public int KategorijaId { get; set; }
          public int MJId { get; set; }
          public Nullable<decimal> Cijena { get; set; }

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

### 4.2. View

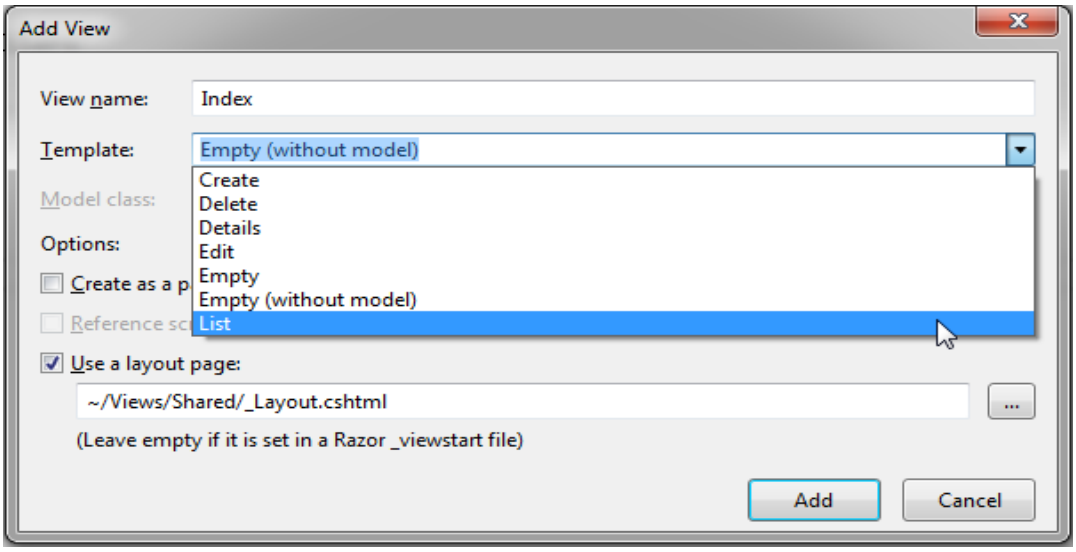
View (pogled) predstavlja korisničko sučelje koje prikazuje podatke iz modela prema korisniku, te mu omogućava upravljanje podacima. Pogledi (eng. *view*) su spremljeni unutar View mape koja sadrži različite mape pogleda za svaki kontroler sa istim imenom kao što ga ima sam kontroler (Slika 15.).[31]

[31] *Usp.* Stackoverflow, *What is ViewModel in MVC?*, Dostupno na: https://stackoverflow.com/questions/11064316/what-is-viewmodel-in-mvc, (07.12.2017.).




Slika 15. View mape za kontrolere

Kako bi smo kreirali novi view, potrebno je nakon desnog klika miša na kontroler za koji želimo dobiti view, odabrati opciju „Add View“, nakon čega damo ime potrebnog view-a (npr. Indeks) te odabrati koju funkciju želimo dodijeliti našem view-u (Slika 16.).




Slika 16. Dodavanje novog view-a

Primjer koda kojeg ćemo dobiti kreiranjem novog view-a:

```
@model IEnumerable<PoslovniSistem.Models.Artikl>

@{
ViewBag.Title = "Index";}
<h2>Index</h2>
<p>
      @Html.ActionLink("Create New", "Create")
</p>
<form asp-controller="Artikls" asp-action="Index" method="get">
      <p>
            Genre: @Html.DropDownList("searchKategorija", "All")
            Naziv: <input type="text" name="SearchString" />
            <input type="submit" value="Filter" />
      </p>
</form>
<table class="table">
      <tr>
            <th>
                  @Html.DisplayNameFor(model => model.Naziv)
            </th>
            <th>
                  @Html.DisplayNameFor(model => model.Cijena)
            </th>
            <th>
                  @Html.DisplayNameFor(model =>
                  model.Kategorija.KategorijaNaziv)
            </th>
            <th>
                  @Html.DisplayNameFor(model =>
                  model.MjernaJedinica.MJ)
            </th>
            <th></th>
      </tr>
@foreach (var item in Model) {
      <tr>
            <td>
                  @Html.DisplayFor(modelItem => item.Naziv)
            </td>
            <td>
                  @Html.DisplayFor(modelItem => item.Cijena)
            </td>
            <td>
                  @Html.DisplayFor(modelItem =>
                  item.Kategorija.KategorijaNaziv)
            </td>
            <td>
                  @Html.DisplayFor(modelItem =>
                  item.MjernaJedinica.MJ)
      </tr>
}
</table>
```

### 4.3. Controller

Kontroler (eng. *controller*) je klasa dobivena iz osnovne klase *System.Web.Mvc.Controller* . Kontroler klasa sadrži javne metode, zvane *Action* metode. Kontroler i njegove *action* metode obrađuju nadolazeće zahtjeve pretraživača, povlači neophodne modele podataka i vraća odgovarajući odgovor.[32] Da bi dodali novi kontroler u Visual Studiu desnim klikom na *Controller* mapu odaberemo opciju *Add* te kliknemo na *Controller*. Otvori se *Add Scaffold*[33] okvir gdje se odabere opcija „*MVC 5 Controller - Empty*“, zatim odaberemo ime kontrolera. Na kraju postupka dobivamo kod kao što je prikazano u sljedećem primjeru:

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

namespace MVC_BasicTutorials.Controllers
{
        public class StudentController : Controller
      {
            // GET: Student
            public ActionResult Index()
            {
            return View();
            }
      }
}
```

[32] *Usp.* Microsoft, *Controller and Action Methods in ASP:NET MVC Applications*, Dostupno na: https://msdn.microsoft.com/en-us/library/dd410269(v=vs.100).aspx, (12.07.1017.).

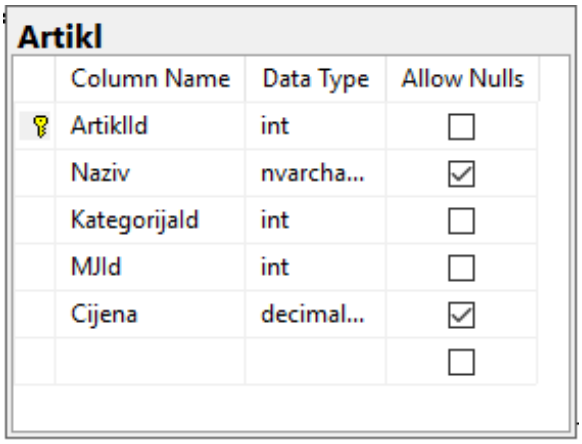
[33] Scaffolding- je automatsko generiranje izgleda za ASP.NET web aplikaciju, reducira vrijeme potrebno za razvijanje kontrolera, view-a i ostalo

# 5. Projekt „Poslovni Sustav “

Koristeći *MVC, Entity Framework i ASP.NET Scaffolding* kreirat ćemo web aplikaciju koja pruža sučelje bazi podataka, koju ćemo kreirati u *Microsoft SQL Server Managment Studio*. U projektu će biti prikazano kako možemo automatski generirati kodove koji omogućavaju korisniku prikaz, uređivanje, kreiranje i brisanje podataka iz baze podataka. Generirani kodovi će biti usklađeni s odgovarajućim kolumnama u tablicama baze podataka.

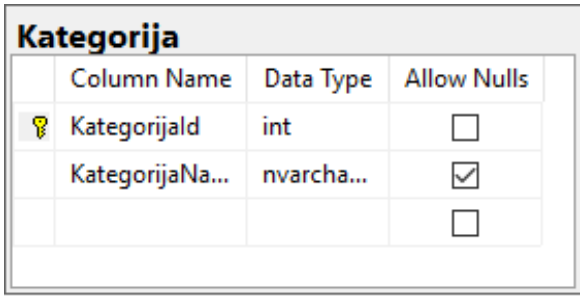
## 5.1. Kreiranje baze podataka

U *Microsoft SQL Server Managment Studiu* ćemo kreirati novu bazu podataka, unutar *Object Explorera,* pod nazivom „PoslovniSustav“. Dobiti ćemo praznu bazu podataka u kojoj ćemo kreirati tablice pomoću *New Query* opcije, a kreirat ćemo sljedeće tablice:



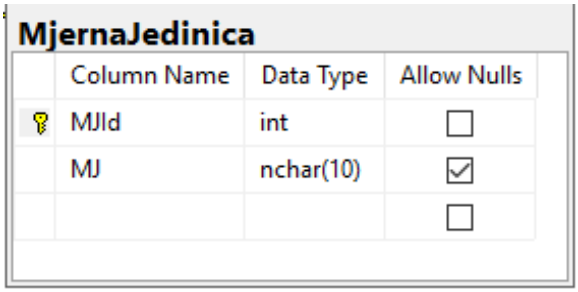

Slika 17. Tablica „*Artikl“*

Tablica *Artikl* će imati primarni ključ *ArtiklId,* kao i strane ključeve *MJId i KategorijaId* koji će omogućiti povezivanje s tablicama *Kategorija i MjernaJedinica.*




Slika 18. Tablica „*Kategorija*“

U tablici će *KategorijaId* imati vrijednost primarnog ključa tablice *Kategorija*.




Slika 19. Tablica „*MjernaJedinica*“

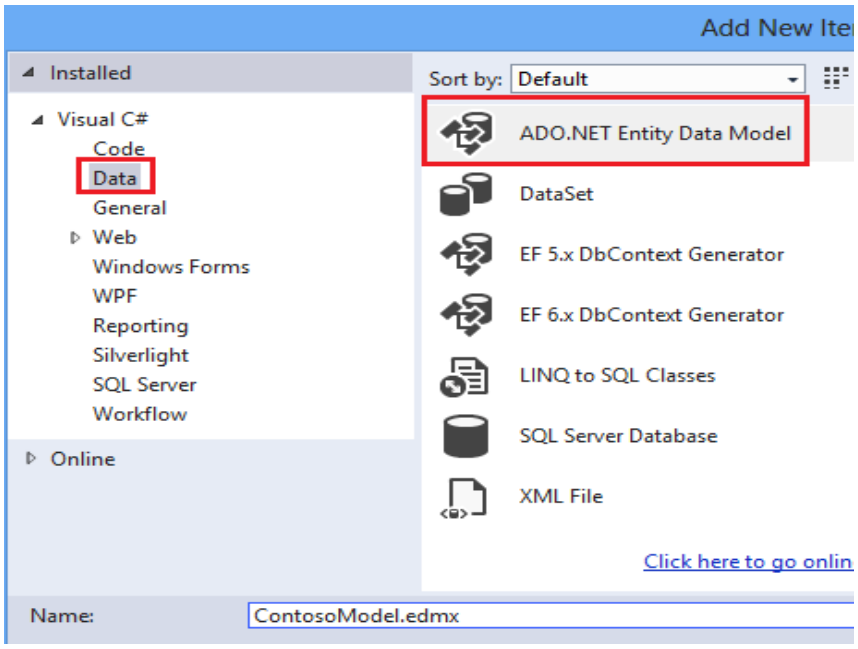
U tablici će *MJId* imati vrijednost primarnog ključa tablice *MjernaJedinica*.

## 5.2. Kreiranje ASP.NET Web aplikacije

U *Visual Studiu* kreiramo novi projekt i odaberemo **ASP.NET Web Application** predložak (eng. *template*), kojem dodijelimo ime „*PoslovniSustav*“. Nakon što se pojavi *New ASP.NET Project* prozor, odaberemo opciju **MVC** predložak i kliknemo na **OK** za kreiranje aplikacije.

Sljedeći korak je generiranje *Entity Framework* modela iz baze podataka koju smo prethodno kreirali. Modeli koje stvorimo su u biti klase (eng. *class*) koje se koriste za rad sa podacima, svaki model obuhvaća po tablicu iz baze podataka i sadrži svojstva koja se podudaraju sa kolumnama u tablici.

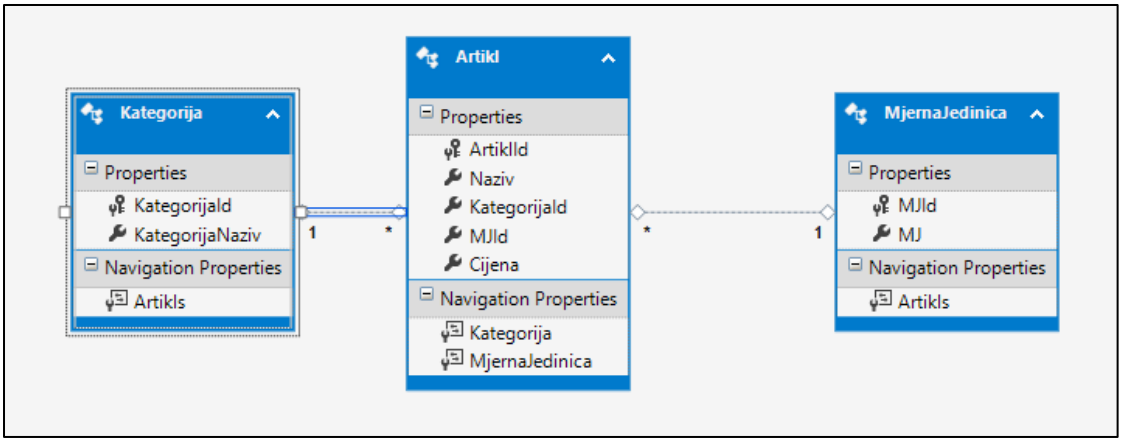
Za kreiranje novog modela potrebno je desnim klikom na mapu **Models**, zatim odabrati opciju **Add** i unutar nje odabrati opciju **New Item**, nakon čega dobivamo prozor u kojemu odabiremo opciju **ADO.NET Entity Data Model** (Slika 17.).




Slika 20. Kreiranje novog modela

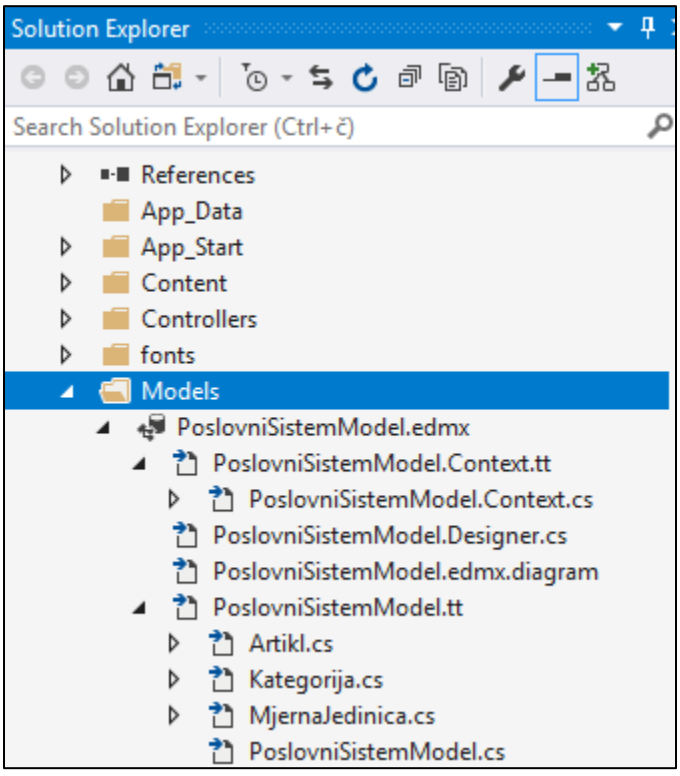
U sljedećem *Entity Data Model Wizard* prozoru odabiremo opciju **EF Designer from database** koja kreira modele u *Entity Framework* dizajneru iz postojeće baze podataka, gdje možemo odabrati povezivanje baze podataka, postavke za model i objekte baze podataka koje želimo uključiti u model. Nakon što kliknemo **Next** dobit ćemo prozor u kojem treba odabrati **New Connection** kako bi kreirali novu konekciju s već izrađenom bazom podataka (Slika 18.).

Nakon što smo dovršili sve korake kreiranja novog modela u projektu se treba pojaviti dijagram koji prikazuje tablice te njihova svojstva i relacije među tablicama (Slika 21.).




Slika 21. Prikaz dijagrama

Mapa *Models* sada sadrži nove datoteke koje su vezane za model koji je generiran iz baze podatka (Slika 22.). Tu se nalaze *Artikl.cs, Kategorija.cs i MjernaJedinica.cs* datoteke koje sadrže klase modela koji predstavljaju tablice baze podataka.

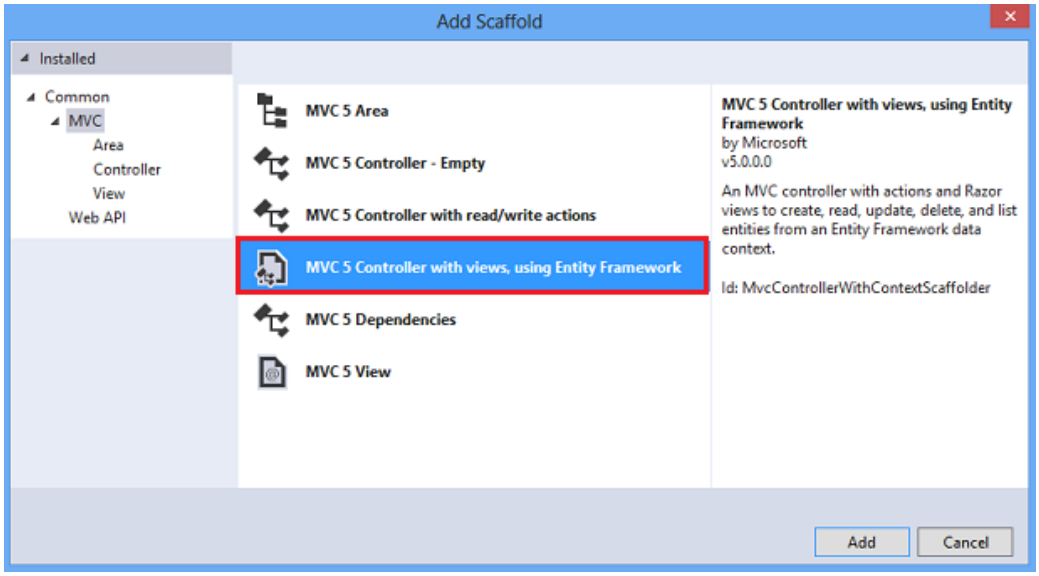



Slika 22. Prikaz mape *Models*

### 5.3. Dodavanje Scaffold-a

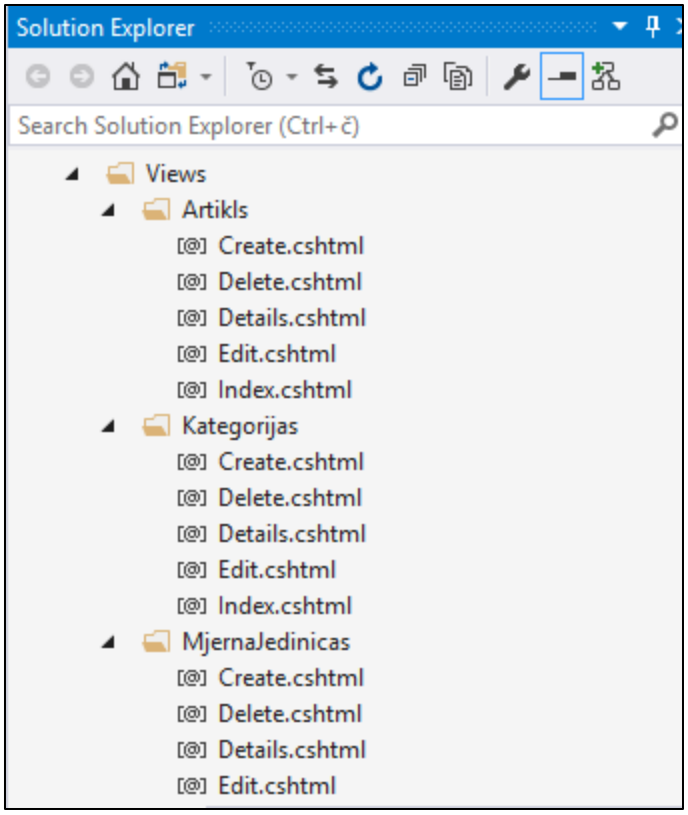
Sljedeći korak u kreiranju aplikacije je generiranje koda koji će pružiti standardne operacije nad podacima za svaki od modela koje smo kreirali, a koje će biti omogućeno dodavanjem *scaffolda*. Dodavanjem *scaffolda* stvorit će se kontroleri i view-ovi za svaki od modela koje smo kreirali (Artikl, Kategorija, MjernaJedinica).

Kako bi dodali novi kontroler u već postojeću **Controllers** mapu, kliknemo desni klik na mapu **Controllers** i odaberemo **Add-New Scaffolded Item**. U prozoru koji se pojavio odaberemo opciju **MVC 5 Controller with views, using Entity Framework** (Slika 23.). Ova opcija će generirati kontroler i view-ove za ažuriranje (eng. *update*), brisanje (eng. *delete*), kreiranje (eng. *create*) i prikazivanje (eng. *display*) podataka u modelu.




Slika 23. Dodavanje kontrolera i view-a za model

Nakon što kliknemo **Add** stvorit će se novi prozor u kojem odabiremo model za koji želimo stvoriti kontroler i view-ove, prvo za model *Artikl* a kasnije u ponovljenim postupcima treba dodati kontrolere i view-ove i za modele *Kategorija* te *MjernaJedinica*. Nakon završetka procesa generiranja kodova za postojeće modele, unutar projektne mape možemo vidjeti nove kontrolere i view-ove unutar njihovih mapa (Slika 24.).




Slika 24. Prikaz mapa Controllers i Views

Prikaz generiranih kodova kontrolera *ArtiklsController.cs* u kojem smo dodali funkciju za pretraživanje artikala a koja nije nastala generiranjem koda iz modela pomoću *scaffolda* :

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web;
using System.Web.Mvc;
using PoslovniSistem.Models;
using System.Threading.Tasks;

namespace PoslovniSistem.Controllers
{
    public class ArtiklsController : Controller
    {
        private PoslovniSustavDatabaseEntities db = new
PoslovniSustavDatabaseEntities();
```

```
        // GET: Artikls
        public ActionResult Index(string searchString, int?
searchKategorija)
        {
            // Pretraživanje po KATEGORIJI.
            var kat = db.Kategorijas.OrderBy(q =>
q.KategorijaNaziv).ToList();
            ViewBag.searchKategorija = new SelectList(kat,
"KategorijaId", "KategorijaNaziv", searchKategorija);
            int kategorijaId = searchKategorija.GetValueOrDefault();

            IQueryable<Artikl> artiklii = db.Artikls
                .Where(c => !searchKategorija.HasValue ||
c.KategorijaId == kategorijaId)
                .OrderBy(d => d.ArtiklId)
                .Include(d => d.Kategorija);
            var sql = artiklii.ToString();

            // Pretraživanje po NAZIVU.
            if (!String.IsNullOrEmpty(searchString))
            {
                artiklii = artiklii.Where(s =>
s.Naziv.Contains(searchString));
            }
            return View(artiklii.ToList());

            var artikls = db.Artikls.Include(a =>
a.Kategorija).Include(a => a.MjernaJedinica);
            return View( artikls.ToList());
        }

        // GET: Artikls/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new
HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Artikl artikl = db.Artikls.Find(id);
            if (artikl == null)
            {
                return HttpNotFound();
            }
            return View(artikl);
        }
```

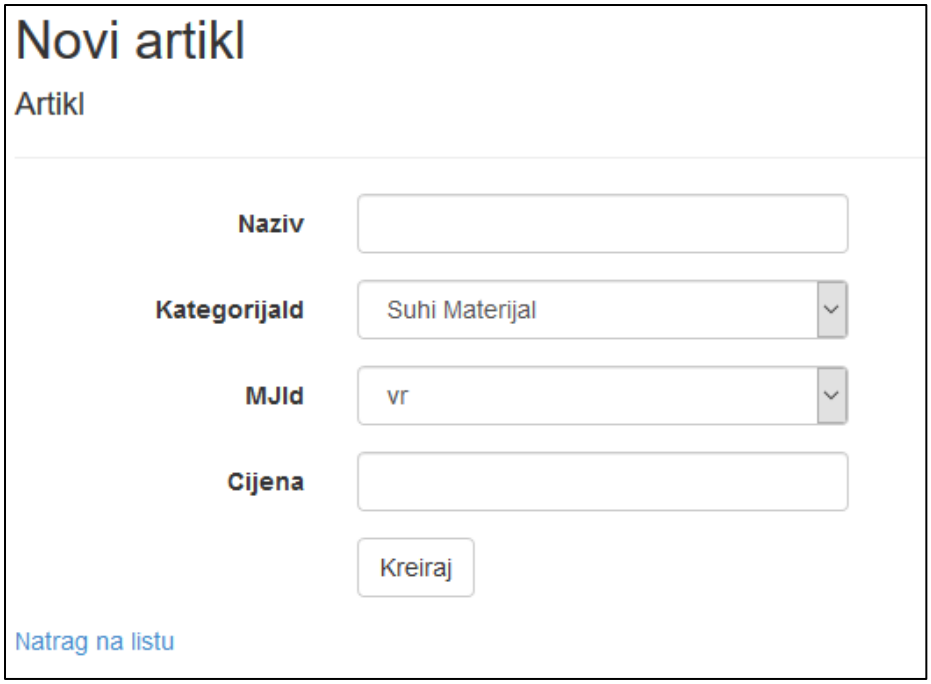
Unutar kontrolera *ArtiklsController* pod „`public ActionResult Indeks`“ smo dodali funkciju, koja nije nastala generiranjem koda iz modela, za pretraživanje artikala po nazivu i po kategoriji kojoj pripadaju određeni artikli.

### 5.4. Pregled projekta

Nakon generiranja koda u kojem smo dobili kontrolere i view-ove, te dodavanja nove funkcije aplikaciji koja omogućava pretraživanje artikala po nazivu i kategoriji, web aplikacija je dobila sljedeći izgled i mogućnosti. Na slici 25. možemo vidjeti izgled stranice „Artikli“ i mogućnosti koje nudi, od pregleda detalja pojedinog artikla, do kreiranja novog artikla, brisanja te ažuriranja.


Slika 25. Pregled početne stranice „Artikli“

Na slici 26. možemo vidjeti kako izgleda forma za kreiranje novog artikla, gdje možemo upisati naziv i cijenu artikla, te odabrati u padajućem izborniku kategoriju i mjernu jedinicu. Također imamo i opciju povratka na listu ukoliko želimo odustati od kreiranja novog artikla.




Slika 26. Kreiranje novog artikla

Na slici 27. možemo vidjeti formu koja nam pruža sve detalje o artiklu koji smo odabrali. Također unutar forme imamo opciju povratka na listu ali i mogućnost uređivanja artikla.

Slike ostalih dijelova aplikacije nalaze se u prilogu.

# 6. ZAKLJUČAK

Baze podatka su neizostavan dio svakog informacijskog sustava, a služe za pohranjivanje podataka, kojima možemo pristupati putem nekog korisničkog sučelja koji može biti pokrenut na jednom računalu ili na više povezanih računala koji su dio neke organizacije i na kraju preko interneta putem web aplikacije. Te iste podatke osim što možemo pregledati možemo i uređivati, brisati, kao i dodavati nove u istu bazu. Baze podataka se sastoje od entiteta koji se mogu jednoznačno odrediti, te se na taj način mogu izdvojiti unutar neke okoline i promatrati. Entiteti se sastoje od atributa koji ga opisuju. Entiteti također sadržavaju i atribut koji ga najbolje (jednoznačno) opisuje te on dobiva status primarnog ključa, dok svi ostali atributi se mogu koristiti kao alternativni (vanjski) ključevi pomoću kojih možemo spojiti više entiteta relacijama.

Cilj aplikacije je dohvaćanje podataka pomoću upita (engl. *Query*), u ovom slučaju artikala za poslovni sustav neke trgovine, te da omogući kreiranje kategorija u koje će se artikli svrstavati radi lakšeg pretraživanja u bazi podataka. U aplikaciji je osim pregleda omogućeno i upravljanje bazom podataka, tako možemo upisivati, uređivati te brisati artikl, kategoriju ili mjernu jedinicu.

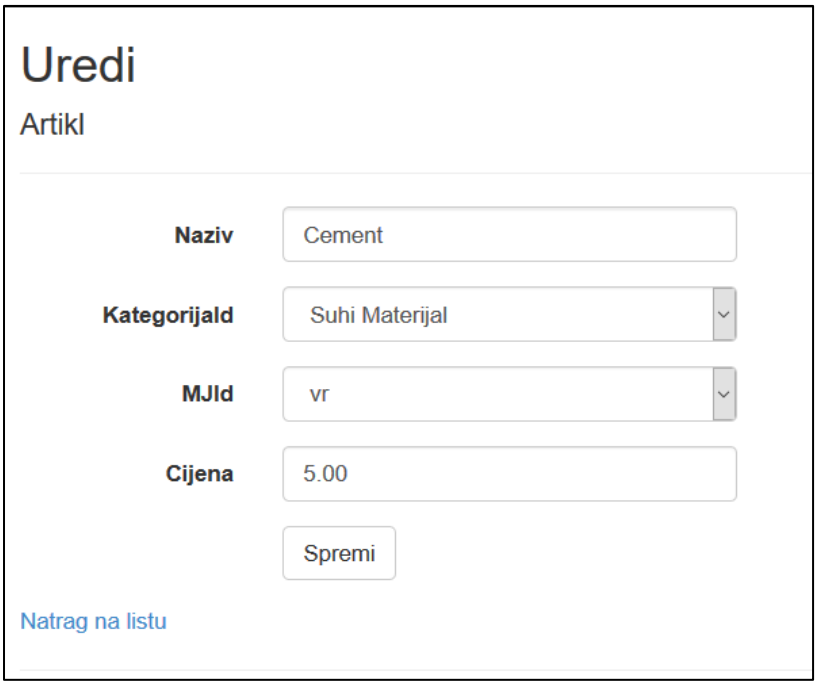
Pri izradi projekta upotrijebljena je MVC tehnologija, koja omogućava brži razvoj web aplikacija u odnosu na druge tehnologije u kojima je potrebno više pisanja koda da bi dobili sve funkcije i izgled aplikacije. MVC posjeduje predloške koje možemo upotrijebiti pri izradi, a također generira kodove iz modela pomoću funkcije *scaffold,* te nam značajno olakšava i ubrzava izradu aplikacije, a značajno smanjuje pisanje koda, što na kraju ovu tehnologiju i čini najboljom za izradu web aplikacija.

# LITERATURA

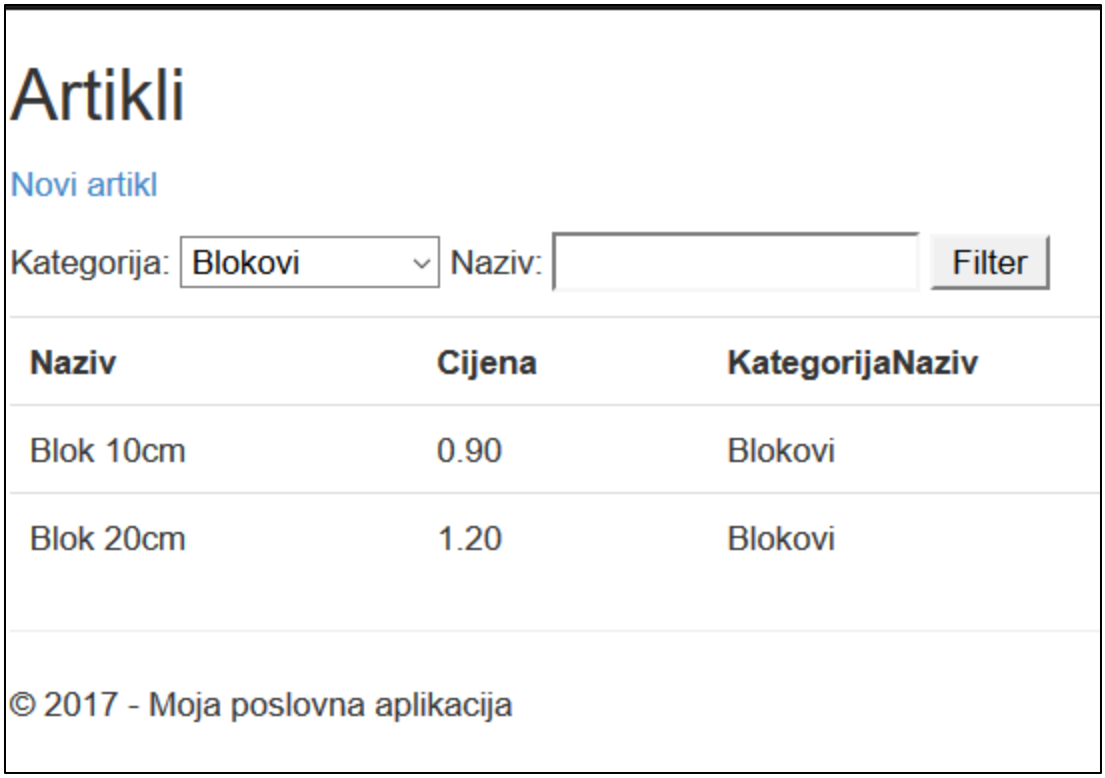

1. Devi Kiran G, REST API Design, Dostupno na: https://www.slideshare.net/DeviKiranGonuguntla/rest-api-design, (13.12.2017.).
2. Informacijski sustavi, Dostupno na: http://www.pfri.uniri.hr/~tudor/materijali/Informacijski%20sustavi,%20baze%20podataka.htm, (04.12.2017.)
3. Loxy, J. (2010) Programming WCF Services, III izdanje, Naklada: O'Reilly Media
4. Guru99, SOAP-Simple Object Access Protocol, Dostupno na: https://www.guru99.com/soap-simple-object-access-protocol.html, (13.12.2017.)
5. Microsoft, Controller and Action Methods in ASP:NET MVC Applications, Dostupno na: https://msdn.microsoft.com/en-us/library/dd410269(v=vs.100).aspx, (07.12.2017.)
6. Pastor, P. (24.03.2010.) MVC for Noobs, Dostupno na: https://code.tutsplus.com/tutorials/mvc-for-noobs--net-10488 , (04.12.2017.)
7. Rahul R. S., A Neginners Tutorial on Custom Forms Authentication in ASP.NET MVC Application, Dostupno na: https://www.codeproject.com/Articles/578374/AplusBeginner-splusTutorialplusonplusCustomplusF, (07.12.2017.)
8. Safari, *Programming WCF Services, 3rd Edition by Juval Lowy*, Dostupno na: https://www.safaribooksonline.com/library/view/programming-wcf-services/9781449382476/ch01s06.html, (07.12.2017.)
9. Spring, Understanding REST, Dostupno na: https://spring.io/understanding/REST, (07.12.2017.)
10. Stackoverflow, What is ViewModel in MVC?, Dostupno na: https://stackoverflow.com/questions/11064316/what-is-viewmodel-in-mvc, (07.12.2017.)
11. Stringfellow, A. SOAP vs. REST: The Differences and Benefits Between the two Web Service Communication Protocols, Dostupno na: https://stackify.com/soap-vs-rest/, (04.12.2017)
12. TechNet, How RPC Works, Dostupno na: https://technet.microsoft.com/en-us/library/cc738291(v=ws.10).aspx, (03.08.2017.)
13. Web Services Glossary, Web service, https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/, (04.12.2017)
14. Wikipedia, Baza podataka, Dostupno na: https://hr.wikipedia.org/wiki/Baza_podataka, (04.12.2017.)
15. Wikipedia, Podatak, Dostupno na: https://hr.wikipedia.org/wiki/Podatak, (12.20.2017.)
16. Wikipedia, SOAP, Dostupno na: https://en.wikipedia.org/wiki/SOAP, (04.12.2017.)
17. Wikipedia, SQL, Dostupno na: https://sh.wikipedia.org/wiki/SQL, (04.12.2017.)

## PRILOG 1. Slike izrađene aplikacije

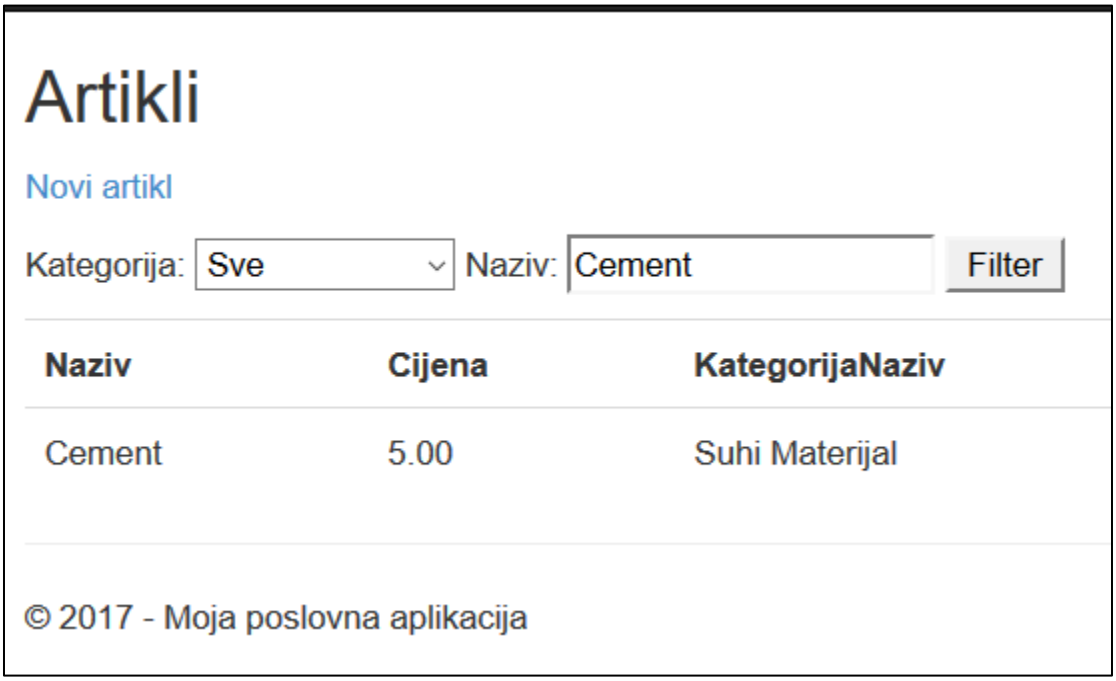

Slika 1. Prikaz detalja odabranog artikla



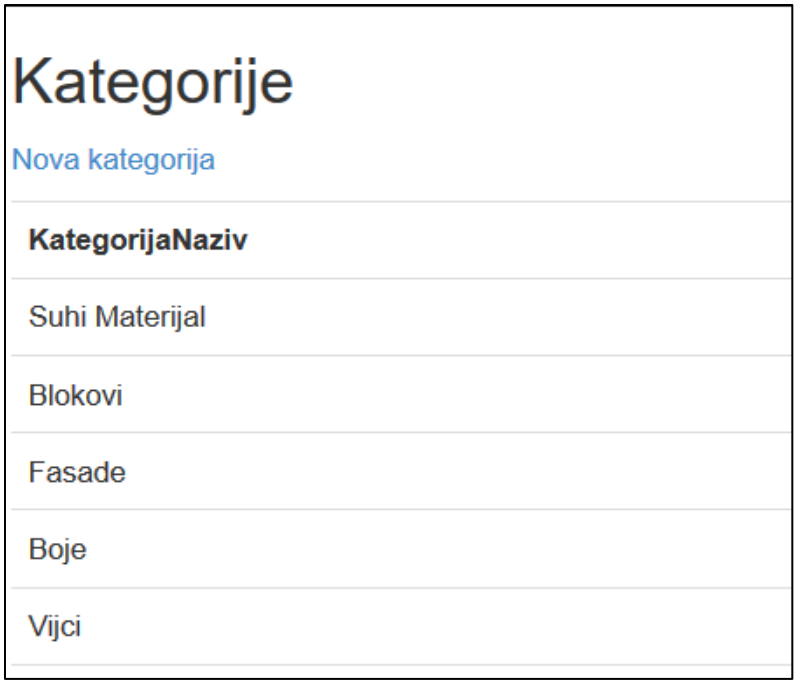

Slika 3. Uređivanje odabranog artikla




Slika 4. Pretraživanje po kategoriji




Slika 5. Pretraživanje po nazivu artikla




Slika 6. Pregled stranice „Kategorije“


Slika 7. Pregled stranice „Mjerne Jedinice“