**Matematički fakultet,**
**Univerzitet u Beogradu**

# MASTER RAD

# Obezbeđivanje komunikacije web aplikacije korišćenjem SQL upita i razvoj aplikacije za elektronsko poslovanje

Mentor:
Prof. dr. Dušan Tošić

Student:
Irena Pajević
10112/2011

Beograd, 2015

# Sadržaj

# 1 Uvod

Danas se skoro 75% do 85% aplikacija, na neki način, bavi nekom vrstom podataka, odnosano bazama podataka [1]. Ovi podaci pomažu kompanijama u praćenju njihovih poslovnih procesa (kao što su procesiranje porudžbine, isporuka informacija, itd). Takođe, ti podaci omogućavaju efikasno upravljanje raznim informacijama.

Čuvanje tolike količine podataka je gotovo nemoguće bez upravljanja tim podacima na pravi način. U cilju upravljanja i čuvanja informacija podataka u skladištu ili bazi podataka, postoje različite metode i modeli koji su razvijeni.

# 2 Relacioni model podataka

Relacioni model je razvio Edgar Kod (eng. *Edgar.F.Codd*) 1970. godine. U to vreme, baze podataka su se zasnivale na navigacionom modelu, kojeg je Edgar Kod smatrao neefikasnim, pre svega zbog slabih mogućnosti pretraživanja podataka u njima. Zbog mnogih nedostataka navigacionog modela Kod je predstavio model, po kojem bi se podaci skladištili u tabele umesto u povezane liste, koje su predstavljale osnovu navigacionog modela. Takođe, Kod je u svom delu naveo i velike prednosti relacionog modela u odnosu na navigacioni, koje se odnose na smeštanje, brisanje i čitanje podataka iz baze[1]. Tek 1980-tih godina relacioni model je doživeo široku komercijalnu upotrebu u poslovnom svetu.

U relacionom modelu sistem se predstavlja preko skupa relacija. Relacija se može predstaviti kao tabela, gde su kolone atributi relacije, a vrste *n*-torke relacije. Svaka relacija ima svoje ime kojim se razlikuje od ostalih relacija u bazi. Kolonu poistovećujemo sa atributom jer jedna kolona relacije sadrži vrednost jednog atributa. Atribut ima svoje ime po kojem se razlikuje od ostalih atributa u istoj relaciji. Vrednosti svih elemenata relacije su podaci istog tipa. Za svaki atribut postoji definisan skup dozvoljenih vrednosti koji se zove domen atributa. Jedan red relacije (tabele) obično predstavlja jedan primerak entiteta, ili predstavlja vezu između dva ili više primerka. Red nazivamo *n*-torka. Matematički relacija se posmatra kao skup *n*-torki pri čemu automatski sledi da se u jednoj relaciji ne sme dogoditi da postoje dve jednake *n*-torke. Broj atributa predstavlja stepen relacije, a broj *n*-torki je kardinalnost relacije. Jedna *n*-torka u relaciji je jedinstveno identifikovana jednim atributom relacije. Taj skup atributa se naziva primarni **ključ relacije**. Po definiciji, ključ *K* relacije *R* je podskup atributa od *R* koji ima sledeća svojstva:

[1] https://sh.wikipedia.org/wiki/Relacijska_baza_podataka

1. Vrednosti atributa iz *K* jednoznačno određuju *n*-torku u *R*. U relaciji *R* ne mogu postojati dve *n*-torke sa istim vrednostima atributa iz *K*.

2. Ako izbacimo iz *K* bilo koji atribut, tada se narušava navedeno svojstvo

- Sve *n*-torke u *R* su međusobno različite, pa *K* uvek postoji. Skup svih atributa zadovoljava navedeno svojstvo 1. Izbacivanjem suvišnih atributa dobija se podskup koji zadovoljava to svojstvo.
- Relacija može da ima više kandidata za ključ. Samo jedan on njih možemo proglasiti primarnim ključem. Atributi od kojih je sastavljen primarni ključ zovu se primarni atributi.

Svaka *n*-torka mora imati vrednost primarnog atributa. Veza između *n*-torke dve relacije uspostavlja se preko vrednosti atibuta tako što se primarni ključ neke relacije pojavljuje kao atribut neke druge relacije koji u tom slučaju postaje **strani ključ**. Grupisanjem atributa u relaciju postiže se tipizacija, tj. klasifikacija. Relacija se u modelu predstavlja u obliku:

nazivRelacije (id_relacije, atributRelacije0, atributRelacije1,…, atributRelacijeN)

U relacionom modelu je moguće ostvariti generalizaciju podataka preko vrednosti atributa relacija. To je moguće ostvariti ako je primarni ključ relacije koja je nadtip primarni ključ relacije koja je podtip. Posmatranjem relacija ni po čemu se ne može zaključiti da li je neka relacija nadtip ili podtip neke druge relacije pa tako za efekat generalizacije možemo reći da je samo prividan. Efekat agregacije takođe može da se postigne preko vrednosti atributa. Skup atributa koji čine primarni ključ agregirane relacije će sadržati primarne ključeve relacija koje čine agregaciju, a može sadržati i neke svoje atribute. Za agregiranu relaciju u relacionom modelu ni po čemu se ne može zaključiti da predstavlja agregaciju dve ili više postojećih relacija, odakle sledi da relacioni model ne podržava ni apstrakciju agregacije. Na osnovu relacionog modela, korišćenjem *SQL*-a i standardnog jezika relacionih sistema za upravljanje bazama podataka stvaraju se tabele, skup ograničenja, kao i pravila integriteta. Nakon toga moguće je izvršavati *SQL* upite kojima se vrše operacije nad podacima u relacionoj bazi podataka. Osnovne četiri operacije nad podacima su kreiranje (*eng. create*), čitanje (*eng. read*), ažuriranje (*eng. update*) i brisanje (*eng. delete*), koje se skraćeno zovu *CRUD* operacije. Detaljnije o relacionom modelu može se videti u literaturi [2].

# 3 Objektni model podataka

Poslednja decenija u softverskom inžinjerstvu je decnija “objektne” orijentacije. Objektna orijentacija je pristup u kome se neki sistem organizuje kao kolekcija međusobno povezanih objekata koji sarađujući ostvaruju postavljene ciljeve. U slučaju relacionih baza podataka, da bi se struktura podataka iz aplikacije mogla da uskladičti, neophodno ju je prvo transformisati u odgovarajući skup tabela relacione baze, što je ponekad složen i vremenski zahtevan postupak. U slučaju objektne baze podataka, imajući u vidu da objekti aplikacije mogu istovremeno da budu i objekti i objekti baze podataka, ovakva transformacija je nepotrebna, pa su i šanse da se ostvare bolje performanse sistema znatno veće. Osnovni element arhitekture objektne baze je objektni model. Objektni model je izveden iz OMG (*Object Menagement Group*) modela. OMG je objektni model definisan kao zajednička osnova za objektne programske jezike, komunikaciju objekata u nekoj klijent-server arhitekturi (eng. *Object Request Brockers* (*ORB*), na primer *CORBA*) i objektne baze podataka. Za više detalja pogledati literaturu [3].

U objektnim modelima podaci se definišu kroz objekte, a sam model se implementira kroz objektnu bazu podataka. Sa aspekta baza podataka, objektni model podataka čine koncepti: objekat, klasa, poruka i metod, učaurenje i nasleđivanje. U objektnom modelu, objekat se definiše kao entitet. Taj entitet ima svoje stanje i ponašanje. Stanje objekta je predstavljeno vrednostima njegovih promenljivih (ili atributa). Ponašanje objekta opisuje se skupom poruka na koje objekat može da odgovori. Komunikacija među objektima se odvija razmenom poruka. Izvršavanje odgovarajuće metode predstavlja odgovor objekta na primljenu poruku. Pristup objektima, tj. njegovim podacima, ostvaruje se metodama, funkcijama, servisima ili operacijama koje su definisane za svaki objekat. Svaki objekat ima pridružen jedinstveni identifikator. Klasa predstavlja apstrktnu predstavu skupa objekata koji imaju iste osobine. Ovim se u objektnom modelu postiže klasifikacija, kao osnovna apstrakcija podataka. Klasa se sastoji od atributa i metoda. Atributima se definiše stanje, a metodama ponašanje klase. Objekat predstavlja konkretan primerak klase. Pored klasifikacije, objektni model podržava i generalizaciju. Generalizacija predstavlja apstrakciju u kojoj se skup sličnih tipova, tj. klasa objekata predstavlja opštijim generičkim tipom, tj. nadtipom.

Objektni model podržava i agregaciju. Agregacija predstavlja apstrakciju podataka kojom se skup postojećih objekata i njihovih međusobnih veza predstavlja novim, jedinstvenim agregiranim tipom. Agregirani objekat se sastoji od komponenti koje predstavljaju objekte koji čine agregaciju, a može da ima i svoje sopstvene atribute i da stupa u vezu sa drugim objektima u modelu. Objektni model podržava i koncept učaurenja (*eng. encapsulation*), što predstavlja sposobnost objekta da sakrije svoje podatke ili da ih učini selektivno dostupnim drugim entitetima. Objektu se može pristupiti samo preko poruka definisanih za taj objekat. Ako dođe do izmene implementacije jednog stanja objekta, ta

izmena ne zahteva izmenu objekata koji su povezani sa tim objektom. Ovim se obezbeđuje određeni stepen nezavisnosti između različitih komponenti sistema koje su međusobno povezane. Zahvaljujući osobinama koje poseduje objektni model, postoje svojstva koja objektnom modelu daju prednost prilikom razvoja poslovnih aplikacija u odnosu na druge modele. To su: preklapanje metoda (*eng. overloading*), prekrivanje metoda (*eng. overriding*), kompatibilnost objektnih tipova, kasno povezivanje metoda, polimorfizam, apstraktne klase, interfejsi, itd.

# 4 Problemi transformacije između objektnog i relacionog modela

Kao rešenje koje omogućava automatizaciju procesa prevođenja između objektnog i relacionog modela, razvijeni su alati za objektno-relaciono preslikavanje (*eng. Object Relational Mapping* - *ORM* alati). Za više detalja videti literaturu [3].

U objektno-orijentisanim aplikacijama perzistencija podataka omogućava čuvanje objekata u relacionoj bazi podataka. U tom slučaju je potrebno da se izvrši transformacija podataka iz objektnog u relacioni model. Da bi process transformacije bio moguć, potrebno je rešiti mnoga neslaganja između ova dva modela. Neslaganja između ova dva modela nastaju jer ovi modeli imaju različite namene. Objektni model se koristi u objektno orijentisanim aplikacijama koje imaju za cilj modelovanje poslovnih problema, dok se relacioni model koristi u relacionim bazama podataka, gde su podaci organizovani u relaciju koja je vizuelno predstavljena dvodimenzonalnom tabelom. Problemi koji nastaju neslaganjem ova dva modela poznati su kao objektno-relaciono neslaganje (*eng. Impedance mismatch*). U nastavku slede neki primeri neslaganja.

## 4.1 Koncept identiteta

Kada se govori o identitetu objekata u objektnom modelu, potrebno je razdvojiti referencu objekta i vrednosti objekta. Objekat postoji nezavisno od svoje vrednosti. U tom kontekstu, u nekim programskim jezicima, npr. u *C#-u* i *Java-i* poređenje objekata se vrši:

- objekat1 == objekat2
- objekat1.equals(objekat2)

Na prvi način porede se reference dva objekta i ukoliko su refernece jednake znači da se radi o istom objektu. Na drugi način porede se vrednosti koje objekti sadrže, što znači da objekti mogu da imaju jednake vrednosti iako su im reference različite[2]. Referenca na objekat identifikuje i tip objekta, jer objekat može da dobije referencu na objekat iste klase ili klase koja je izvedena iz klase referentnog tipa. Veze između objekata ostvaruju se tako što jedan objekat sadrži referencu na drugi objekat. Preko ove reference saznaje se kog je tipa objekat sa kojim se uspostavlja veza. Takođe, preko ove reference pristupa se vrednostima atributa referenciranog objekta.

U relacionom modelu jedini identifikator svake *n*-torke jeste vrednost koja se nalazi u ćeliji koja predstavlja primarni ključ relacije. Zato se za svaki objekat iz objektnog modela, prilikom transformacije u relacioni model, određuje atribut koji će ga jedinstveno identifikovati (primarni ključ relacije) i koji će, prlikom povezivanja relacija, igrati ulogu reference u relaciji sa kojom se vrši povezivanja (strani ključ).

## 4.2 Koncept nasleđivanja

Jedan od najvažnijih koncepata objektnog modela jeste mogućnost da jedan objekat nasledi i proširi osobine nekog drugog objekta. Pošto relacioni model ne podržava koncept nasleđivanja, neophodno je realizovati mehanizme koji će transformisati hijerarhiju objekata u relacije tako da se postigne efekat nasleđivanja u relacionom modelu.

## 4.3 Asocijacije

Problem asocijacija odnosi se na transformacije veza koje se uspostavljaju između objekata u objektnom modelu i veza koje se ostvaruju između relacija u relacionom modelu. U relacionom modelu veze se ostvaruju korišćenjem spoljnih ključeva. U objektnom modelu postoji nekoliko vrsta asocijacija:

- 1-1 jedan na jedan (*eng. one-to-one*),
- *-* više na više (*eng. many-to-many*),
- 1-* jedan na više (*eng. one-to-many*)

Transformacija asocijacije jedan na jedan najčešće se vrši na taj način što se oba objekta predstavljaju jednom relacijom. Na taj način podaci jednog objekta proširuju se podacima drugog objekta i postaju jedinstvena relacija. Atributi relacije postaju atributi i jednog i drugog objekta.

[2] https://msdn.microsoft.com/en-us/library/bsc2ak47(v=vs.110).aspx

Kod asocijacije jedan na više, kada jedan objekat pravi vezu sa više objekata nekog tipa, transformacija se vrši tako što se naprave posebne relacije za svaki od tipova objekta. Primarni ključ objekta, koji pravi vezu, pamti se kod svih objekata sa kojima je on u vezi, tj. primarni ključ relacije sa leve strane asocijacije predstavlja se kao strani ključ relacije na desnoj strani asocijacije.

Kod asocijacije više na više transformacija se vrši na taj način što se, pored relacija koje se prave za svaki od tipova objekata, pravi i dodatna relacija koju čine primarni ključevi relacija koje stupaju u vezu.

## 4.4 Struktura i ponašanje

Interfejs u relacionom modelu predstavlja struktura relacije, a interfejs objektnog modela predstavlja ponašanje objekata izraženo preko javnih metoda. Proces transformacije objektnog u relacioni model i obratno treba da omogući preslikavanje atributa objekata koji čine njegovu strukturu u atribute relacije. Kako koncept interfejsa u objektnom modelu ne podrazumeva direktan pristup njegovim atributima, potrebno je napraviti vezu između atributa relacije i metode objekta koja obezbeđuje pristup odgovarajućem atributu tog objekta.

Da bi se prevazišli ovi problemi, potrebno je izvršiti objektno relaciono preslikavanje.

# 5 Objektno relaciono preslikavanje

Objektno relaciono preslikavanje (eng. *Object-relational mapping* - *ORM*) je tehnika koja sadrži programski skup klasa koje preslikavaju podatke iz relacione baze podataka u objekte u određenom programskom jeziku. Aplikacija poziva odgovarajući *API* (*eng. Application Programming Interface*)[3] za komunikaciju sa bazom podatka. Tako, na primer, *Oracle Pro * C* je *Oracle*-ov *API* koji omogućava čitanje, kreiranje, ažuriranje ili brisanje zapisa u *Oracle*-ovoj bazi podataka za aplikacije pisane u C programskom jeziku. Pre kompilator *Pro * C* prevodi *SQL* upite u pozive pomoću *Oracle*-ove biblioteke *SQLLIB*.

*ORM* omogućava da se tabele i skladištene procedure iz relacione baze podataka predstave kao objekti u odgovarajućem programskom jeziku, tako da umesto da pišemo tradicionalne *SQL* upite nad tom bazom podataka, možemo da koristimo metode i svojstva nad tim objektima.

[3] API - programski interfejs aplikacije koji definiše način na koji će softverske komponente komunicirati jedna sa drugom.

Tako, na primer, umesto *SQL* kode:

```
String ime =  “SELECT ime FROM dbo.osoba WHERE OsobaID = 1”
DbCommand dbc = new DbCommand();
Result res = dbc.Execute();
String imeIspis = res[0][“ime”];
```

može se raditi sa objektima, i u tom slučaju prethodni primer ima sledeću formu:

```
Osoba o = repository.GetPerson(1);
String imeIspis = o.Ime;
```

Sa *ORM*-om svaka baza podataka je predstavljena *ORM* konceptom objekata u konkretnom programskom jeziku, a tabele iz baze podataka su predstavljene klasama sa odnosima između tih klasa. *ORM* je odgovoran za preslikavanje i veze između tih klasa i baze podataka. Dakle, na nivou aplikacije, baza podataka je sada u potpunosti predstavljena klasama. Aplikacija samo treba da se bavi ovim klasama, umesto fizičkim podacima baze podataka. Sa aspekta aplikacije, nije potrebno voditi računa o tome kako da se izvrši povezivanje sa bazom podataka, kako da se izgrade *SQL* upiti, kako da se koristi mehanizam zaključavanja kojim treba da se obezbedi konkurentnost, ili kako da se rukuje distribuiranim transakcijama. Ovim aktivnostima, koje se odnose na bazu podataka, će rukovati *ORM*.

U nastavku će biti predstavljena jedna alatka *ORM*, *Entity Framework*.

# 6 Entity Framework

*Entity Framework* je *Microsoft*-ova tehnologija u *ADO.NET*-u (*eng. ActiveX Data Objects .NET*)[4] koji podržava razvoj objektno orijentisanih aplikacija. Za nešto više od 6 godina postojanja, Microsoftov Entity Framework postao je jedna od vodećih tehnologija za pristup podacima, a od verzije 6 projekat je otvoren kao Open Source i kod je dostupan pod Apache v.2 licencom. Trenutna aktuelna verzija je 6.1.2. *Entity Framework* omogućava potpunu apstrakciju relacionih baza, kreiranjem objektnog sloja između aplikacije i samih podataka. Prednosti korišćenja *Entity Framework* su:

- Osnovni razlog velike popularnosti Entity Framework-a leži u njegovoj spostobnosti da veliki deo koda generiše automatski i na taj način programerima štedi dragoceno vreme i trud.
- Automatsko ažuriranje pri promenama objekata i lako osvežavanje šeme nakon promene strukture baze.

[4] *ADO.NET* je skup softverskih komponenti koji omogućava pristup i menjanje podataka u relacionim sistemima baza podataka. *ADO.NET* je deo osnovne klase biblioteke *Microsoft .NET Framevork*-a.

- Oslobođenje zavisnosti od ''fiksiranja koda''[5] pri radu sa određenim podacima ili pri skladištenju šema.
- Preslikavanje između konceptualnog modela i šeme za skladištenje se može promeniti bez promene koda same aplikacije.

Više o definiciji *Entity Framework*-a može se pronaći u literaturi [4].

## 6.1 Istorijat

Prva verzija *Entity Framework* (*EFv1*) je bila u okviru *.NET Framework* 3.Servis Paketa 1 i *Visual Studio 2008*, koji je objavljen 11-og avgusta 2008. Ova verzija *EF* nije imala veliki uspeh i naišla je na oštre kritike stručnjaka.

Druga verzija *Entity Framework*-a, pod nazivom *Entity Framework 4.0* objavljena je u sklopu *.NET 4.0* verzije u aprilu 2010. U ovoj verziji *EF* ispravljene su brojni nedostaci prethodne verzije.

Treća verzija *Entity Framework* je verzija 4.1, koja se pojavila aprila 2011. godine, donela je inovaciju kao što je kôd-prvo (eng. *code-first*) implementaciju modela podataka (*eng. Entity Data Model-EDM*). Ova verzija je imala svoje nadogradnje koje su objavljene u junu 2011. i februaru 2012. godine. Verzija 4.1.1 uključuje ispravku greški i podržava nove tipve podataka. Vrezija 4.3.1 donela je par novih mogućnosti među kojim je i podrška za migraciju podataka.

Naredna verzija *Entity Framework*-a 5.0.0 došla je uz *.NET Framework 4.5* u avgustu 2012. godine.

Verzija 6.0 objavljena u oktobru 2013. godine i ovo je projekat otvorenog koda (*eng. open source)* pod licencom *Apache License v2*. *EF* 6.0 donosi dugačak spisak novih karakteristika, bolje performanse, asinhronu podršku za upite i ažuriranja. Ova verzija je nadograđivana i došla je do oznake 6.1.3 koja je objavljena u martu 2015. godine. Detaljnije o razvoju *Entity Framevork*-a može se pronaći u literaturi [5].

[5] Hard kodovanja (*eng. hard-coded*)


*Slika 1: Kratak istorijat Entity Framework-a*

## 6.2 Arhitektura Entity Framework-a

*Entity Framework* je komponenta *ADO.NET* i tehnologija sa snažnim akcentom na modelovanje. Jezgro *Entity Framework* je model podataka (*eng. Entity Data Model-EDM*). Ovaj model omogućava prilagođavanje preslikavanja između objekta klase i konkretne tabele iz baze podataka. Uvodi se sa namerom da se rad sa podacima olakša i učini još efikasnijim. *EDM* omogućava programeru da kreira objekat klase koji će što preciznije odgovarati domenu problema. Na slici 2 prikazan je primer *EDM*-a.


*Slika 2: Entity Data Model*

Na navedenoj slici možemo primetiti da tabele iz baze podataka nisu direktno preslikane na objekte klase.

Primetimo da se informacije o zaposlenima, adresama i telefonskim podacima nalaze smestene u tri različite tabele u bazi što je, sa stanovišta baze podataka, sasvim opravdan raspored podataka. Ali sa stanovišta klase, ove informacije nema potrebe razdvajati u zasebne klase. Na primer, opravdano je da sve informacije o zaposlenom budu smeštene u jednu klasu *Zaposleni*.

Suprotno ovom primeru, imamo primer kada se iz jedne tabele u bazi prave tri različite klase koje predstavljaju pojedinačne objekte. Postojaće tri zasebne klase objekata (*Računovodstvo*, *Marketing*, *Finansije*), ali prilikom skladištenja podataka informacije o sve tri klase će se upisivati u jednu tabelu *Odeljenje* u bazi podataka.

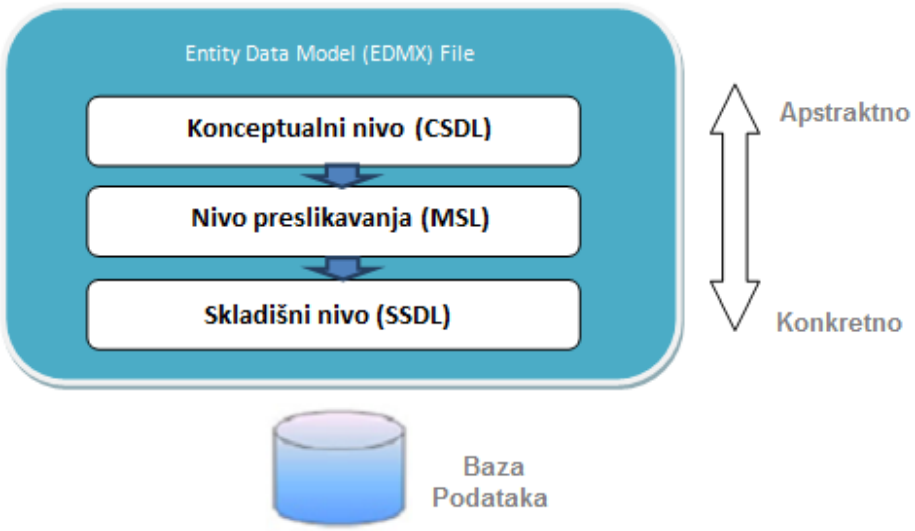
Takođe, na slici se može videti preslikavanje tabele *Pozicija* u jedni klasu *Pozicija*. Preslikavanje jedne tabele iz baze u jednu klasu je podrazumevano ponašanje *Entity Framework*.

Zahvaljujući preslikavanju, funkcionalnosti *Entity Framework*-a, programer će raditi sa objektima klase i neće razmišljati o tabelama iz baze.

## 6.3 Nivoi entiti modela podataka

*EDM* se sastoji od tri odvojena nivoa:

- konceptualnog (*eng. conceptual*)
- skladišnog (*eng. store*)
- preslikavajućeg (*eng. mapping*)




*Slika 3: Nivoi EDM*

U konceptualnom nivou *EDM*-a se nalaze objektne klase. To je nivo nad kojim programeri rade. U zavisnosti od toga kako je implementiran *Entity Framework*, konceptualni nivo može biti kreiran pomoću dizajnera ili iz koda u okruženju *Visual Studio*[6]. Model se može napraviti na osnovu već postojeće baze podataka korišćenjem dizajnera koji je podržan od strane *Entity Framework*. *Entity Framework* omogućava i obrnuti postupak: da se na osnovu modela napravljenog iz koda generiše baza podataka. Konceptualni nivo je definisan u *Conceptual Schema Definition Language* (*CSDL*) *XML* datoteci.

Skladnišni nivo *EDM*-a predstavlja šemu baze podataka. *Store Schema Definition Language* (*SSDL*) *XML* datoteka je datoteka za definisanje skladišnog nivoa *EDM*-a.

Nivo preslikavanja definiše vezu između konceptualnog i skladišnog nivoa. Na ovom nivou se definiše veza između objekta klase i konkretne tabele iz baze podataka. Programeru je ovaj nivo dostupan u *Entity Framework* dizjneru u okviru prikaza preslikanih detalja (*eng. Mapping Details Window*). Ovaj sloj je definisan pomoću *Mapping Specification Language* (*MSL*) *XML* datoteke datoteke.

Za više detalja o *EDM* nivoima videti u literaturi [6].

## 6.4 Načini implementacije Entity Framework-a

Modeliranje je osnovna karakteristika *Entity Framework*. Kada se jenom implementira model, onda se sve izmene rade nad modelom, a ne nad kolonama i redovima iz baze podataka.

Da bismo u projektu upotrebili *Entity Framework*, potrebno je prvo konfigurisati taj projekat u okruženju *Visual Studio* što najpre podrazumeva definisanje EDM-a, definisanje parametara za povezivanje sa bazom podataka, kao i dodavanje određenih projektnih referenci.

Postoje tri načina definisanja *EDM*-a:

- Baza-prvo (*eng. Database-First*)
- Model-prvo (*eng. Model-First*)
- Kôd -prvo (*eng. Code-First*)

Baza-prvo definisanje EDM-a podrazumeva generisanje objektnih klasa na osnovu postojeće baze podataka. *Entity Framework* sadrži alat *The Entity Framework Power Tools* koji omogućava veoma jednostavno i brzo generisanje objektnih klasa na osnovu postojećih objekata iz baze.

Model-prvo definisanje *EDM*-a omogućava pravljenje modela podataka u dizajneru a zatim generisanje baze podataka na osnovu napravljenog modela. Ako pravimo novu aplikaciju za koju nemamo bazu podataka, onda nam *Entity Framework Wizard* nudi opciju kreiranja praznog modela. Iz palete sa alatkama (*eng. Toolbox*) mogu se prevlačiti objekti i

[6] *Visual Studio* je razvojno okruzenje za .NET tehnologije

relacije među tim objektima. Kada napravimo željeni model, na osnovu njega *Entity Framework* nam omogućava da generišemo praznu bazu podataka.

Pored ova dva načina, postoji i treći način definisanja *EDM*-a, kôd -prvo*,* koji omogućava da se napravi model korišćenjem *POCO* klasa (eng. *Plain old CLR objects*). Ovaj način podrazumeva da se naprave *POCO* klase koje imaju istu strukturu kao i šema baze podataka sa kojom se ostvaruje preslikavanje. *POCO* klase sadrže atribute koji odgovaraju kolonama tabela, odgovarajuće *get* i *set* metode za pristup atributima i odgovarajuće konstruktore. Ove klase implementiraju interfejs *java.io.Serializable* koji je potreban za korišćenje objekata u sesiji. Pored ovih *POCO* klasa, koje odgovaraju tabelama iz baze podataka, neophodno je generisati klasu koja će vršiti povezivanje *POCO* klasa sa *ObjectContext*-om. Ta klasa opisuje način pristupa *POCO* klasama kao i oblik modela. Na osnovu ovih klasa *Entity Framework* će napraviti praznu bazu podataka.

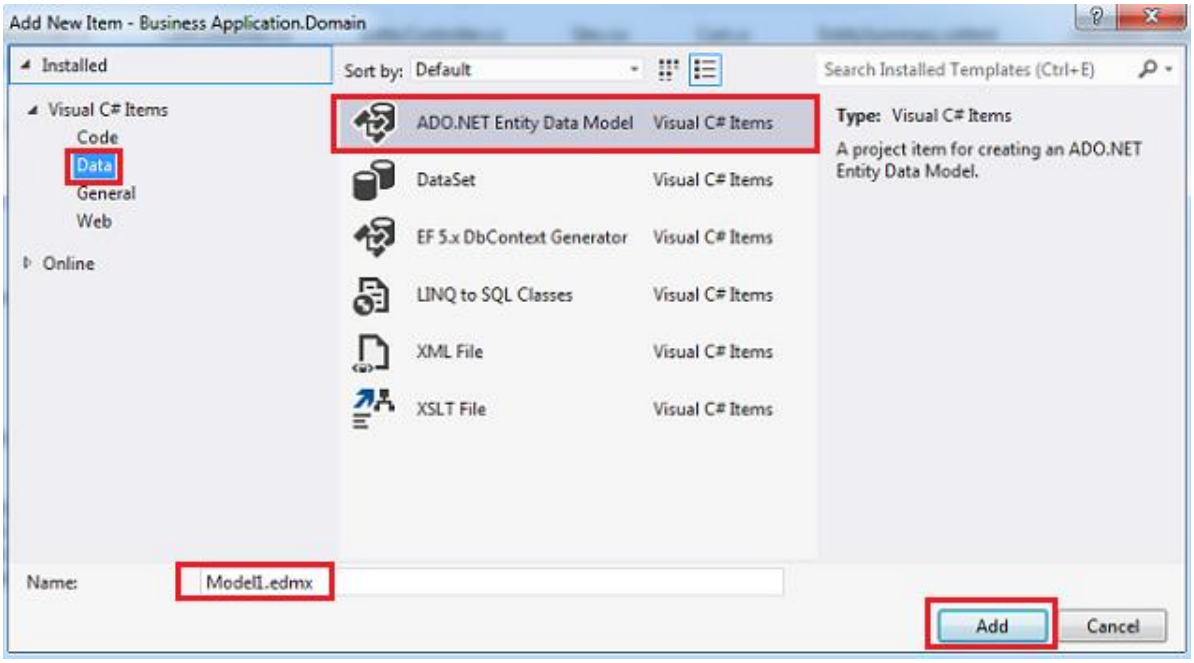
Za više detalja o *EDM* nivoima videti u literaturi [7].

## 6.5 Pravljenje jednostavnog modela podataka pristupom prvo-model

U sledećem primeru biće predstavljeno kreiranje modela na osnovu kojeg će se generisati baza podataka.

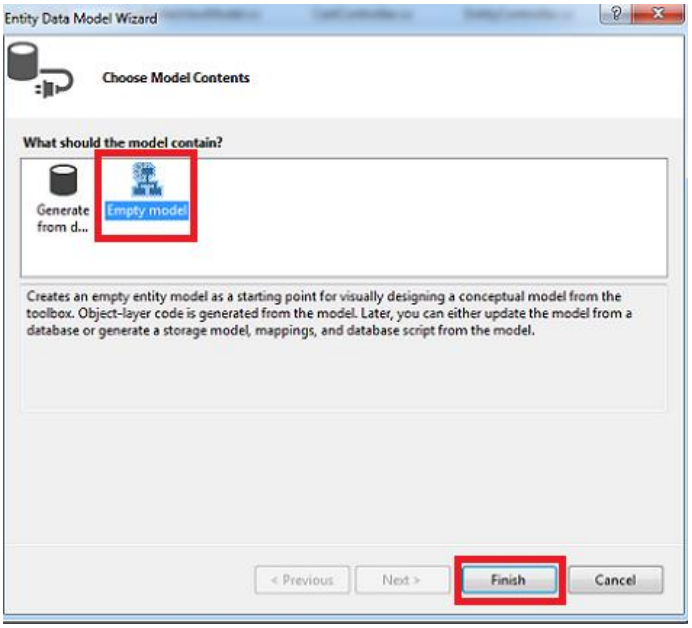
Da bismo kreirali novi model potrebno je u projekat dodati novi *ADO.NET EDM* i uraditi sledeće korake:

1. Desni klik na projekat i iz padajućeg menija odabrati pomoću: *Add* ➢ *New Item*.
2. Izabrati *ADO.NET Entity Data* Model i kliknuti na dugme *Add*.




*Slika 4: Dodavanje novog modela u projekat*

3. U sledećem koraku treba izabrati opciju kreiranja praznog modela.

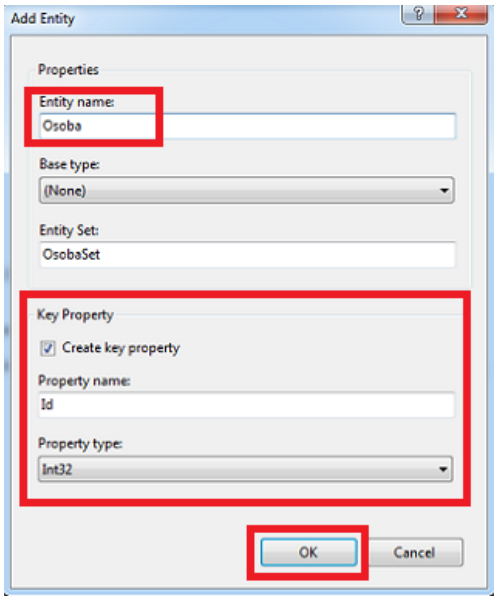



*Slika 5: Dodavanje praznog modela*

Pored opcije kreiranja praznog modela moguće je odabrati opciju generisanja modela podataka iz već postojeće baze podataka.

4. Desnim klikom tastera miša na dizajn površinu moguće je izabrati opciju *Add* ➤ *Entity* za kreiranje novog entiteta.

5. U ovom primeru biće kreiran entitet *Osoba* i biće mu dodeljeno svojstvo (*eng. property*) *Id*, koje je tipa *Int32* a koje je, izborom opcije *Create a key property*, postavljeno da bude primarni ključ.



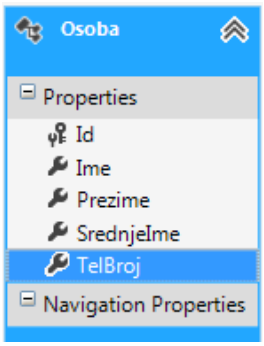

*Slika 6: Dodavanje objekta*

6. Sledeći korak je dodavanje dodatnih svojstava entitetu *Osoba*. Ta svojstva se mogu dodati izborom opcije: *Add ➢ Scalar Property.*

7. Svojstva koja dodajemo su: *Ime, Prezime, SrednjeIme I TelBroj*.

8. Da bi se podesilo automatsko generisanje primarnog ključa, potrebno je na svojstvu koje je primarni kljuc, u ovom slučaju Id, iz padajućeg menija odabrati opciju *Properties* i podesiti da opcija *StoreGeneratedPattern* bude *Identity*.

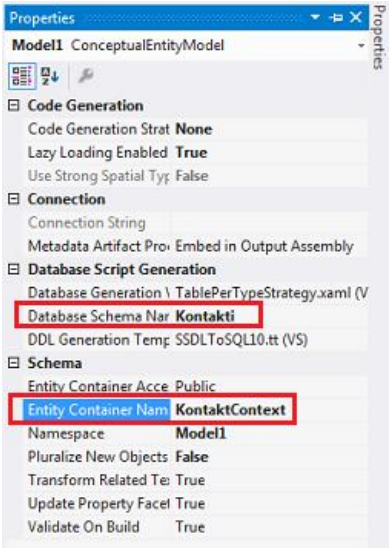
Kada završimo sa svim ovim podešavanjima, model treba da izgleda kao na slici 7.




*Slika 7: Završni model sa tipom objekta koji predstavlja Osobu*

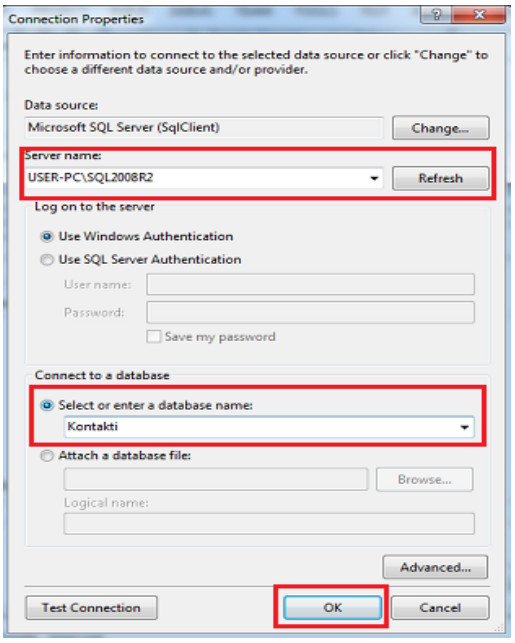
Sada imamo jednostavan konceptualni model. Da bismo generisali bazu podataka iz ovog modela padataka, postoji još nekoliko stvari koje treba da uradimo.

9. Potrebno je da promenimo još nekoliko osobina ovog modela. Desnim klikom miša na dizajin površinu i izborom opcije *Properties*, u polju *Database Schema Name*, treba upisati željeno ime baze koja će biti generisana na osnovu ovog modela. U ovom primeru ime baze će biti *Kontakti*.




*Slika 8: Promena podešavanja modela*

10. Generisanje šeme baze podataka vrši se izborom opcije *Generate Database Script from Model.* Potrebno je konfigurisati povezivanje na server na kojem će se nalaziti baza podataka koja se generiše na osnovu modela.




*Slika 9: Kreiranje nove konekcije sa bazom podataka koju će Entity Framework koristiti za kreiranje skripta koji će generisati bazu podataka na osnovu konceptualnog modela*

11. U sledećem koraku se generiše baza podataka na osnovu sledećeg skripta:

```
SET QUOTED_IDENTIFIER OFF;
GO
USE [Kontakti];
GO
IF SCHEMA_ID(N'Kontakti') IS NULL EXECUTE(N'CREATE SCHEMA [Kontakti]');
GO
-- ------------------------------------------------
-- Kreiranje tabela
-- ------------------------------------------------
CREATE TABLE [Kontakti].[OsobaSet]
(
    [Id] int IDENTITY(1,1) NOT NULL,
    [Ime] nvarchar(max)  NOT NULL,
    [Prezime] nvarchar(max)  NOT NULL,
    [SrednjeIme] nvarchar(max)  NOT NULL,
    [TelBroj] nvarchar(max)  NOT NULL
);
GO
-- ------------------------------------------------
-- Kreiranje primarnog ključa
-- ------------------------------------------------

-- Creating primary key on [Id] in table 'OsobaSet'
ALTER TABLE [Kontakti].[OsobaSet]
ADD CONSTRAINT [PK_OsobaSet]
    PRIMARY KEY CLUSTERED ([Id] ASC);
GO
```

Nakon pokretanja ovog skripta, baza podataka je kreirana. *Entity Framework* dizajner je moćan alat za kreiranje i ažuriranje konceptualnog, skladišnog i preslikanog nivoa. Ovaj alat pruža podršku za dvosmerni razvoj modela. Alat omogućava da se započne sa čistom dizajin površinom i napravi model, ili da se počne sa bazom podataka koja već postoji i da se generišu konceptualni, skladišni i nivo preslikavanja *ORM*. Alat takođe podržava ponovno kreiranje baze iz modela i ažuriranje modela prilikom promena nad bazom.

Pored generisanja baze podataka, *Entity Framework* generiše i skup klasa koje odgovaraju entitetima koji su deo modela. U nastavku sledi primer kreiranja i ubacivanja instance tipa *Osoba* u bazu podataka, koji je pisan sintaksom programskog jezika *C#*.

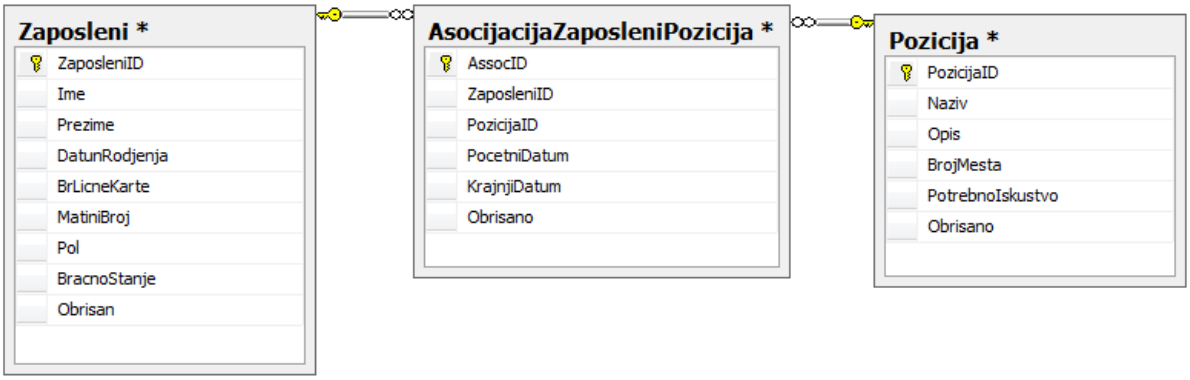
```
using (var context  = new KontaktContext())

       var osoba = new osoba { Ime = "Robert",
                               SrednjeIme="Allen",
                               Prezime = "Doe",
                               TelBroj = "867-5309"
                             };
       context.People.Add(osoba);

       osoba = new osoba { Ime = "John",
                           SrednjeIme="K.",
                           Prezime = "Smith",
                           TelBroj = "824-3031"
                                 };
       context.People.Add(osoba);

       osoba = new osoba { Ime = "Billy",
                           SrednjeIme="Albert",
```

```
                        Prezime = "Minor",
                        TelBroj = "907-2212"
                      };
        context.People.Add(osoba);

        osoba = new osoba { Ime = "Kathy",
                        SrednjeIme="Anne",
                        Prezime = "Ryan",
                        TelBroj = "722-0038"
                      };
        context.People.Add(osoba);
        context.SaveChanges();
}
using (var context = new KontaktContext())
{
        foreach (var osoba in context.People)
        {
                System.Console.WriteLine("{0} {1} {2}, Phone: {3}",osoba.Ime,
                osoba.SrednjeIme,osoba.Prezime, osoba.TelBroj);
        }
}
```

## 6.6 Kreiranje Modela iz postojeće baze podataka

*Entity Framework* sadrži alate za automatsko generisanje klasa na osnovu konceptualnog modela definisanog u *Entity Data Model*-u. Objektni servisi predstavljaju objektni sloj na nivou aplikacije koji se sastoji od generisanih klasa.

Posmatraćemo jednostavan primer. Imamo bazu koja se sastoji iz tri tabele: *Zaposleni*, *Pozicije* i *AsocijacijaZaposleniPozicija*. U tabeli *Zaposleni* nalaze se informacije o zaposlenima, tabela *Pozicije* sadrži informacije o svim raspoloživim mestima zaposlenja i tabela *AsocijacijaZaposleniOsoba* sadrži vezu između zaposlene osobe i njegove vrste posla. *Zaposleni* može izvršavati više poslova pa se tako u tabeli *AsocijacijaZaposleniPozicija* jedan isti zaposleni moze javiti vise puta. U bazi podataka postoji i pogled koji povezuje ove tri tabele i kao rezultat vraća informacije o zaposlenom i o poziciji koju on zauzima u firmi. U bazi podataka pogledi se koriste samo da bi izlistali neki podaci. Nad pogledima nije moguće vršiti ažuriranje, ubacivanje, niti brisanje podataka. Ovo je takođe slučaj i sa *Entity Framework*, on podržava samo izlistavanje podataka iz pogleda.




*Slika 10: Jednostavna baza podataka sa tri tabele Zaposleni, Pozicije i AsocijacijaZaposleniPozicija*

Kreiranje modela iz postojeće baze podataka se sastoji iz sledećih koraka:

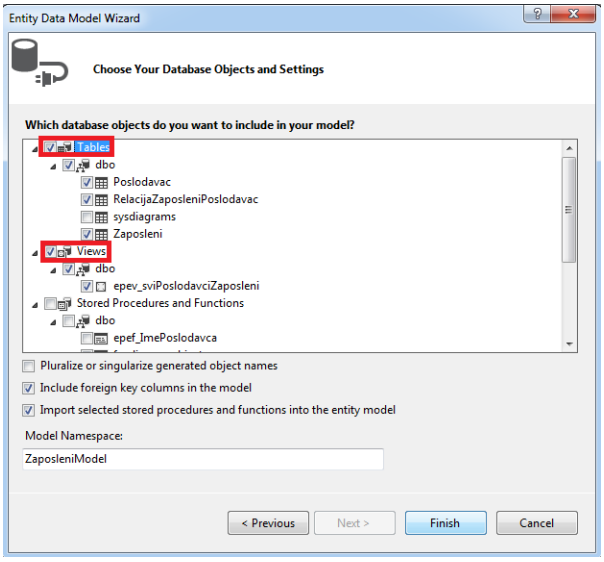
1. Desnim klikom na projekat odabrati *Add* ➢ *New Item*.

2. Odabrati *ADO.NET Entity Data Model*

3. Odabrati opciju Generate from database (kao na slici 5).

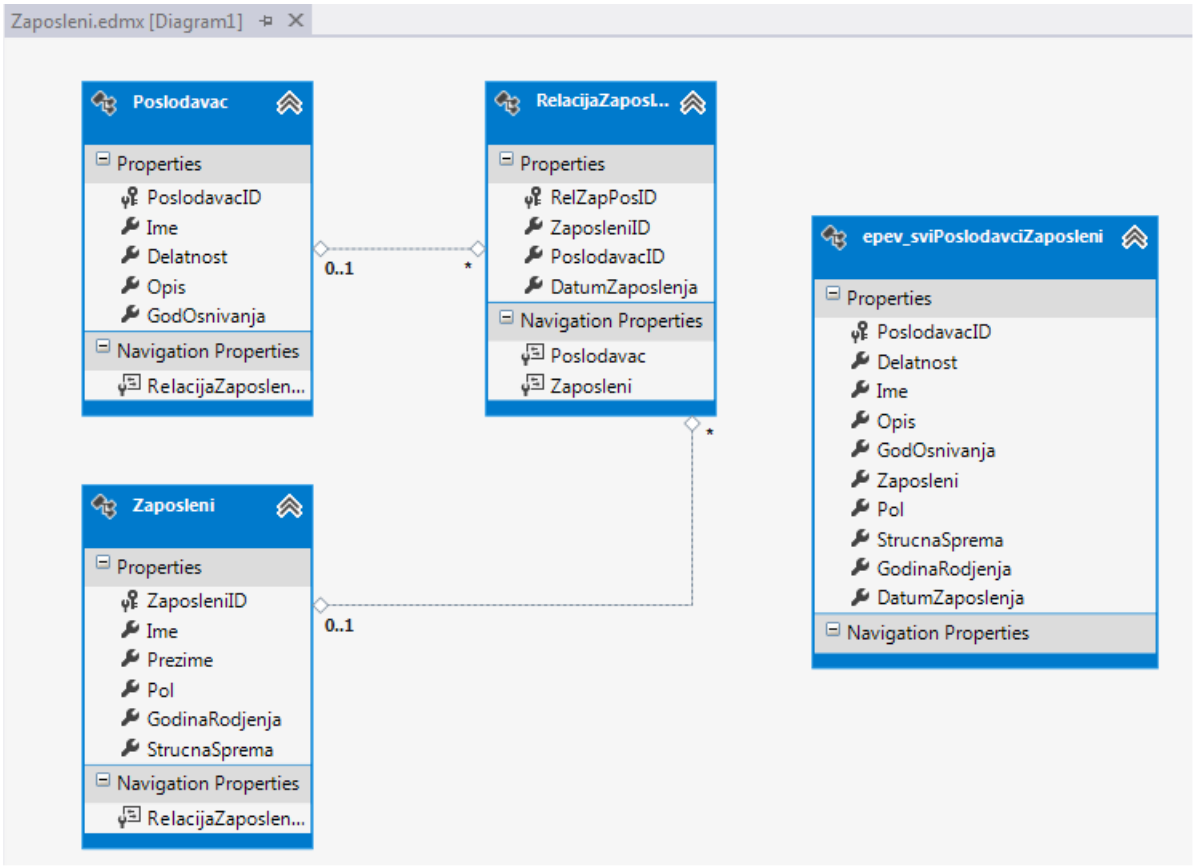
4. Moguće je odabrati već kreiranu konekciju ili napraviti novu konekciju. Prilikom pravljenja nove konekcije potrebno je odabrati odgovarajuću instancu *SQL* servera i odabrati način pristupa serveru. Kada se ovo podesi, korisno je kliknuti na dugme *Test Connection*, da bi se izvršila provera konekcije sa serverom. Nakon toga treba odabrati konkretnu bazu podataka koja se nalazi na tom serveru.

5. U sledećem koraku su prikazane sve tabele, pogledi, procedure i funkcije koje su skladištene u odabranoj bazi podataka. Ovde je moguće označiti šta iz objekata baze podataka želimo da prebacimo u model.




*Slika 11: Označavanje tabela, pogleda, funkcija i procedura koje će model sadržati*

Kada kliknemo da dugme *Finish*, dobijamo model koji se sastoji od entiteta *Zaposleni*, *Poslodavac* i *AsocijacijaZaposleniPoslodavac* i pogleda *epev_sviPoslodavciZaposlenih*.




*Slika 12: Model Zaposleni*

Primer za kreiranje modela korišćenjem *cod-first* pristupa biće predstavljen kroz primer kreiranja veb aplikacije u poglavlju 8.

# 7 Upiti nad Entity Data Model-om

Upit je izraz koji preuzima podatke iz izvora podataka (*eng. data source*). Upiti se obično izražavaju specijalizovanim jezikom upita, kao što je *SQL* za relacione baze podataka i *XQuery* za *XML*. Za detalje pogledati literaturu [8]. Programski jezik integrisanih upita (*eng. Language-Integrated Query - LINQ*) nudi jednostavan, konzistentan model za rad sa podacima iz raznih vrsta izvora podataka. U *LINQ* upitima se uvek radi sa programskim

objektima. *LINQ* upit se sastoji od tri akcije: dobijanja izvora podataka, kreiranja upita i izvršavanje upita.

Upiti nad *EDM-om* se mogu pisati na više načina:

- *LINQ to Entities*
- *Entity SQL*

Svaki od načina ima svoje prednosti. Neki od načina se biraju na osnovu ličnih prioriteta, a drugi će biti izabrani da bi se iskoristile sve prednosti odabranog načina. Takođe, mogu se koristiti posebne metode (neke bazirane na nivou *LINQ* a neke bazirane na nivou *Entity Framework* klase - *ObjectQuery*[7]) da bi se upit izvršio. Svaki od ovih stilova upita će rezultirati dematerijalizovanim objekatom. *EntitiClient API* je manje poznat način za izvršavanje upita korišćenjem *Entity Framework API*, koji omogućava izvršavanje upita na objektnom nivou.

## 7.1 LINQ to Entities

*LINQ to Entities* je jezik integrisanih upita (*LINQ*) koji programerima omogućava da pišu upite nad *Entity Framework*-ovim konceptualnim modelom koristeći *Visual Basic* ili *Visual C#*. *LINQ to Entities* konvertuje *LINQ* upite, izvršava te upite nad konceptualnim modelom, a vraća rezultate koji mogu da se koriste od strane *Entity Framework*-a.

Generička klasa *ObjectQuery* generiše upit koji vraća kolekciju objekata. Ovi upiti se obično konstruišu nad postojećim objektom konteksta, umesto da se pišu nad samim podacima iz baze podataka. Ovaj kontekst pruža informacije o povezivanju i metapodacima koji su potrebni da bi se upit kreirao i izvršio. Generička klasa *ObjectQuery* implementira generički interfejs *IQueryable*[8], koji na osnovu svojih metoda omogućava *LINQ* upitima da se postepeno grade. Takođe, instanca generičke klase *ObjectQuery*, koja implementira generički interfejs *IQueryable*, služi kao izvor podataka za *LINQ to Entities* upite. U upitu se tačno navode informacije koje želimo da preuzmemo iz izvora podataka. U upitu se može odrediti kako treba da se sortiraju, grupišu i oblikuju informacije koje će biti rezultat tog izvešenog upita. *LINQ* upit se smešta u promenljivu. Ova promenljiva ne učestvuje ni u kakvoj akciji i ne vraća rezultate, već samo čuva informacije samog upita. Nakon kreiranja upita, taj upit se može izvršiti i rezultati tog upita se mogu preuzeti.

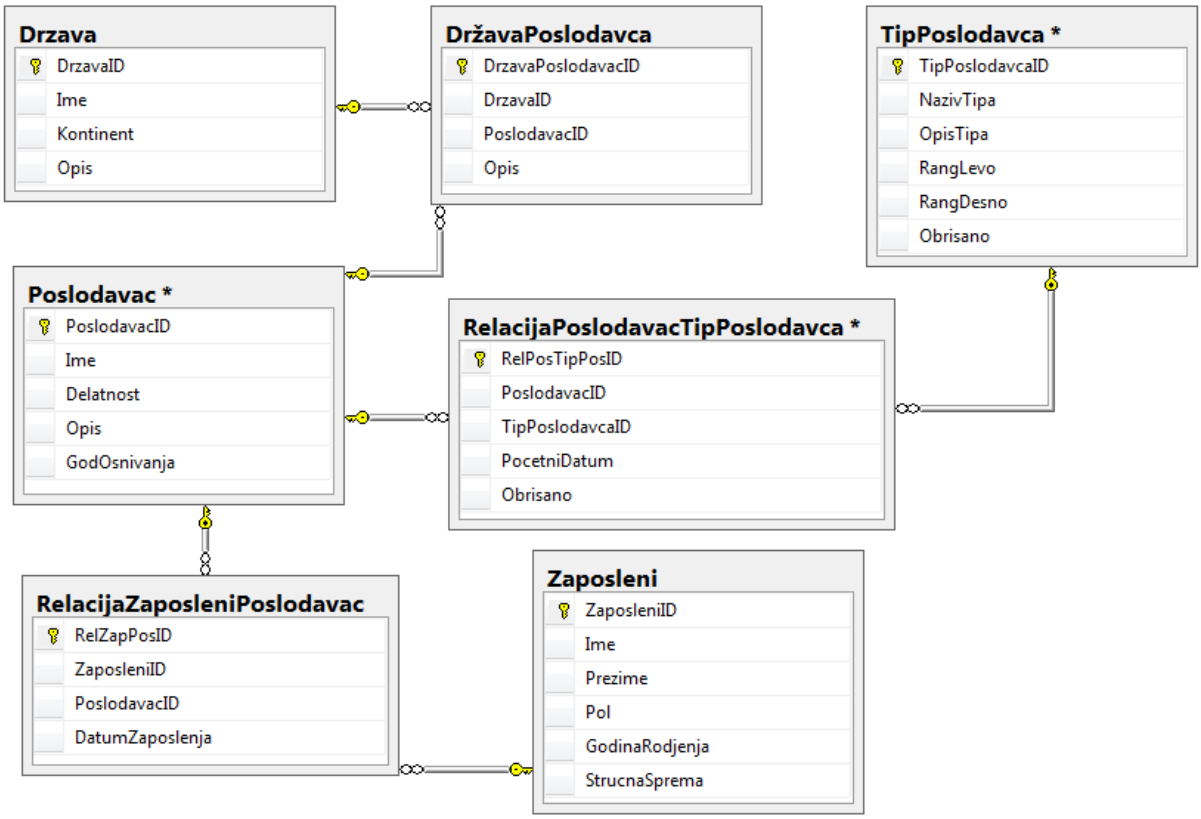
[7] Sprovodi zajedničku funkcionakcionalnost za izvršavanje upita nad *EDM*
[8] *Iqueryable* omogućava izvršavanje upita nad odgovarajućom izvorom podataka, gde je tip podatka poznat

Postoje dve sintakse za pisanje *LINQ to Entities* upita:

- Sintaksa iskaza (*eng. query expression syntax*)
- Sintaksa zasnovana na metodu (*eng. method-based query syntax*)

U nastavku sledi primer sintakse iskaza.

Posmatramo kontekst *BazaZaposlenjaContex* koji je izgrađen nad bazom podataka *BazaZaposlenja* koja se sastoji od sledećih tabela: *Zaposleni, Poslodavac, TipPoslodavca, RelacijaPoslodavacTipPoslodavca, RelacijaZaposleniPoslodavac, DržavaPoslodavca, Država*.




*Slika 16: Šema tabela baze podataka BazaZaposlenja*

U nastavku je naveden primer koji koristi metod *select* da vrati sve redove iz tabele *Zaposleni*. Prilikom ispisa rezultata biće prikazana samo imena zaposlenih. Svi primeri će uključivati sledeće klase:

```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Objects;
using System.Globalization;
using System.Data.EntityClient;
using System.Data.SqlClient;
using System.Data.Common;
```

```
    using (BazaZaposlenjaContex context = new BazaZaposlenjaContex())
```

```
{
    IQueryable<Zaposleni> zaposleniUpit = from Zaposleni in context.Zaposleni
                                          select zaposleni;

    Console.WriteLine("Imena zaposlenih:");
    foreach (var prom in zaposleniUpit)
    {
        Console.WriteLine(prom.Ime);
    }
}
```

Metod *select* izvlači samo ime zaposlenog iz tabele *Zaposleni* pri čemu se ispisuju sva imena zaposlenih.

```
using (BazaZaposlenjaContex context = new BazaZaposlenjaContex())
{
    IQueryable<string> zaposleniImena =
        from z in context.Zaposleni
        select z.Ime;

    Console.WriteLine("Imena zaposlenih:");
    foreach (String zaposleniIme in zaposleniImena)
    {
        Console.WriteLine(zaposleniIme);
    }
}
```

U sledećem primeru koristi se metod s*elect* da bi se kolone *Zaposleni.ZaposleniId*, *Zaposleni.Ime*, *Zaposleni.Prezime*, *Zaposleni.Pol* projektovale u niz anonimnih tipova.

```
using (BazaZaposlenjaContex context = new BazaZaposlenjaContex())
{
    var upit =
        from zaposleni in context.Zaposleni
        select new
        {
            ZaposleniId = zaposleni.ZaposleniId,
            ZaposleniIme = zaposleni.Ime,
            ZaposleniPrezime = zaposleni.Prezime,
            ZaposleniPol = zaposleni.Pol

        };

    Console.WriteLine("Informacije o zaposlenima:");
    foreach (var zapInfo in upit)
    {
        Console.WriteLine("Zaposleni Id: {0}
                           Zaposleni Ime: {1}
                           Zaposleni Prezime: {2}
                           Zaposleni Pol: {3} ",
                zapInfo.ZaposleniId,
                zapInfo.ZaposleniIme,
                zapInfo.ZaposleniPrezime,
                zapInfo.ZaposleniPol);
    }
}
```

*LINQ to Entities* je takođe podržan od strane *Oracle*, *DB2*, *MySQL*.

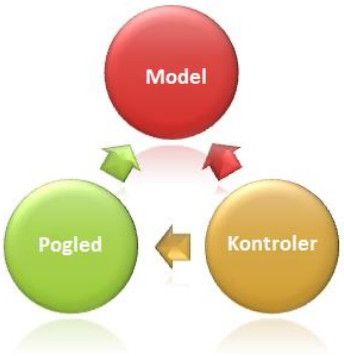
## 7.2 Entity SQL

*Entity SQL* (*eSQL*) je upitni jezik koji je sličan tradicionalnom *SQL*-u[9] i koji omogućava pisanje upita za konceptualni model u *Entity Framework*-u[10]. Zapravo *eSQL* je prva osmišljena sintaksa za pisanje upita nad objektima. Kako konceptualni model predstavlja podatke kao objekte i relacije, tako *eSQL* omogućava pisanje upita nad tim objektima i relacijama u obliku koji je sličan tradicionalnim upitima u *SQL* upitnom jeziku. *Entity Framework* radi sa strogo definisanim provajderima podataka koji prevode generisani *eSQL* kôd u čvrsto definisane upite. *EntityClient* provajder obezbeđuje način da se izvrši *eSQL* naredba nad *EDM*-om, pri čemu je rezultat te naredbe široki opseg podataka uključujući i skalarne i tabelarne rezultate. Kada su kreirani *EntityCommand*[11] objekti, onda je moguće specifikovati uskladištene procedure ili upite koji se dodeljuju *eSQL* upitnom stringu u okviru System.Data.EntityClient.EntityCommand.CommandText svojstva. Kada aplikacija u toku rada generiše *eSQL* upit, treba imati u vidu ograničenja u dužini komandne naredbe. *eSQL* ne sprovodi ograničenja na dužinu teksta upita. *EntityDataReader* izlaže rezultate izvršavanja *EntityCommand* nad *EDM*. Da bi se izvršila komanda i prosledili rezultati koje vraća *EntityDataReader*, potrebno je pozoviti klasu *ExecuteReader*[12].

# 8 Korišćenje Entity Framework-a u okviru ASP.NET MVC 4 veb aplikacije

*ASP.NET* je *Microsoft*-ova slobodna platforma koja podržava tri različite tehnologije za pravljenje veb aplikacija. Tehnologije koje su podržane su: Veb stranice, Veb forme i *MVC* tehnologija. *MVC* je relativno nova tehnologija u okviru *ASP.NET*-a koja je veoma popularan i afirmisan koncept dizajniran da se koristi kao obrazac u razvoju softvera. Ova tehnologija koristi *MVC* koncept[13], koji je troslojan i sastoji se iz:

- Model *(eng. Model*)
- Pogled *(eng. View)*
- Kontroler *(eng. Controller)*




[9] *SQL - Structured Query Language* je programski jezik za posebr ıravljanje podacima koji su skladišteni u relacionom sistemu za upravljanje bazama podataka.
[10] https://msdn.microsoft.com/en-us/library/vstudio/bb399560(v=vs.100).aspx
[11] *EntityCommand* - Predstavlja komandu za izvršavanje upita nad *Entity Data Model*-om.
[12] *ExecuteReader* – Iščitava redove iz izvora podataka .
[13] http://www.asp.net/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

Model predstavlja jezgro aplikacije koji obezbeđuje logiku za podatke aplikacije. Model sadrži objekte preuzete iz baze podataka.

Pogled je deo aplikacije koji obezbeđuje prikaz podataka.

Kontroler je deo aplikacije koji obezbeđuje interakciju korisnika i same aplikacije. Posao kontrolera je da čita podatke iz pogleda, da kontroliše unos korisnika i da prosleđuje ulazne podatke u model.

Troslojna struktura *MVC*-a omogućava nezavisan pristup slojevima. Moguće je usresrediti se na izgled prikaza bez zavisnosti od poslovne logike aplikacije. Takođe, ovakav pristup olakšava i testiranje same aplikacije. Prednost *MVC* je i ta što na različitim delovima aplikacije mogu raditi razilčiti programeri istovremeno. *MVC* model obezbeđuje potpunu kontrolu nad *HTML*, *CSS*, and *JavaScript* klasama.

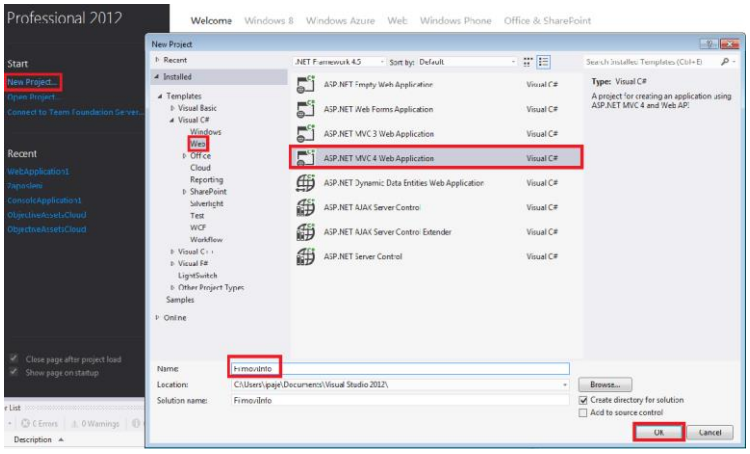
## 8.1 Osnovne funkcionalnosti ASP.NET MVC veb aplikacije

Osnovne funkcionalnosti svake aplikacije su kreiranje, čitanje, ažuriranje i brisanje podataka. U nastavku će biti predstavljen jednostavan primer veb aplikacije koja demonstrira ove četri osnovne funkcionalnosti. FimoviInfo je veb aplikacija na kojoj ćemo demonstrirati kreiranje *ASP.NET MVC 4* aplikacije koristeći *Entity Framevork 5* i *Visual Studio 2012*. Aplikacija će prikazati listu filmova, omogućiće korisniku da unesu nove filmove, ažurira ili obriše već postojeće. Aplikacija će koristiti pristup *cod-first*. Za detalje pogledati u literaturi [9].

### 8.1.1 Kreiranje MVC veb aplikacije

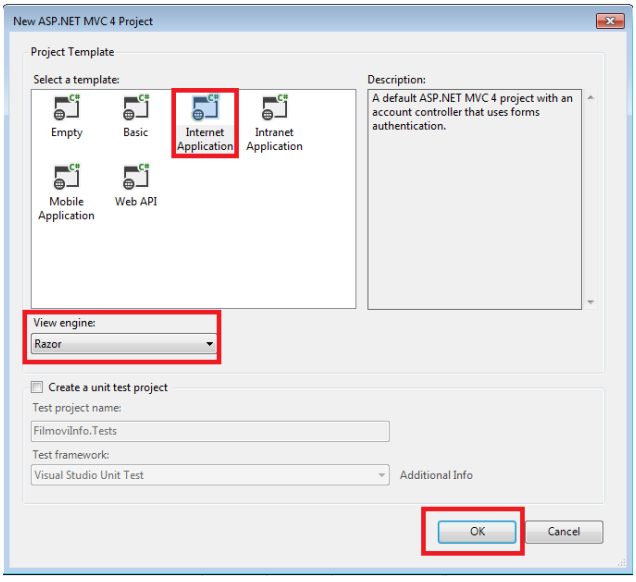
U *Visual Studio 2012* biće konfigurisan novi projekat[14], pre čemu će se koristiti *ASP.NET MVC 4 Web Application* šablon.

[14] *File -> New Project*




*Slika 17: Kreiranje ASP.NET MVC 4 veb aplikacije*

U dijalog prozoru *New ASP.NET MVC 4 Project* treba odabrati *Internet Application* šablon. Za potrebe ove aplikacije biće korišćen *Razor* pokretač (*eng. Razor Engine*)[15] zato će opcija *View engine* biti odabrana. Među ponuđenim opcijama je i opcija da se kreira testni projekat (*Create a unit test project*). Ova opcija će u ovom primeru biti onemogućena.




*Slika 18: Kreiranje ASP.NET MVC 4 veb aplikacije*

Nakon kreiranja projekta, biće urađeno par izmena koje se odnose na izgled početne strane i menija sajta. Izmene se vrše na *Layot* strani (*Views\Shared_Layout.cshtml*).

```
<!DOCTYPE html>
```

[15] *Razor* je *ASP.NET* sintaksa programiranja koja se koristi za kreiranje dinamičkih veb stranica sa *C#* ili *Visual Basic* programskim jezikom. *Razor* omogućava integrisanje *C#* ili *Visual Basic*-a *sa HTML*-om. *Razor* je razvijen u junu 2010-te godine a uz verziju *Microsoft Visual Studio 2010* i pojavio se 2011-te godine.

```
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>@ViewBag.Title - Filmovi Info</title>
        <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <meta name="viewport" content="width=device-width" />
        @Styles.Render("~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
    </head>
    <body>
        <header>
            <div class="content-wrapper">
                <div class="float-left">
                    <p class="site-title">@Html.ActionLink("Filmovi Info", "Index",
"Home")</p>
                </div>
                <div class="float-right">
                    <section id="login">
                        @Html.Partial("_LoginPartial")
                    </section>
                    <nav>
                        <ul id="menu">
                            <li>@Html.ActionLink("Home", "Index", "Home")</li>
                            <li>@Html.ActionLink("About", "About", "Home")</li>
                            <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                            <li>@Html.ActionLink("Filmovi", "Index", "Film")</li>
                            <li>@Html.ActionLink("Glumci", "Index", "Glumac")</li>
                            <li>@Html.ActionLink("Zanrovi", "Index", "Zanr")</li>
                        </ul>
                    </nav>
                </div>
            </div>
        </header>
        <div id="body">
            @RenderSection("featured", required: false)
            <section class="content-wrapper main-content clear-fix">
                @RenderBody()
            </section>
        </div>
        <footer>
            <div class="content-wrapper">
                <div class="float-left">
                    <p>&copy; @DateTime.Now.Year - Filmovi Info</p>
                </div>
            </div>
        </footer>

        @Scripts.Render("~/bundles/jquery")
        @RenderSection("scripts", required: false)
    </body>
</html>
```

Sledeća izmena je na Home strani ( *Views\Home\Index.cshtml*). Tu će biti obrisana podrazumevana poruka.[16]

```
@{
    ViewBag.Title = "Home Page";
}
@section featured {
```

[16] Deo teksta koji je označen u crvenoj boji je obrisan iz Home view-a.

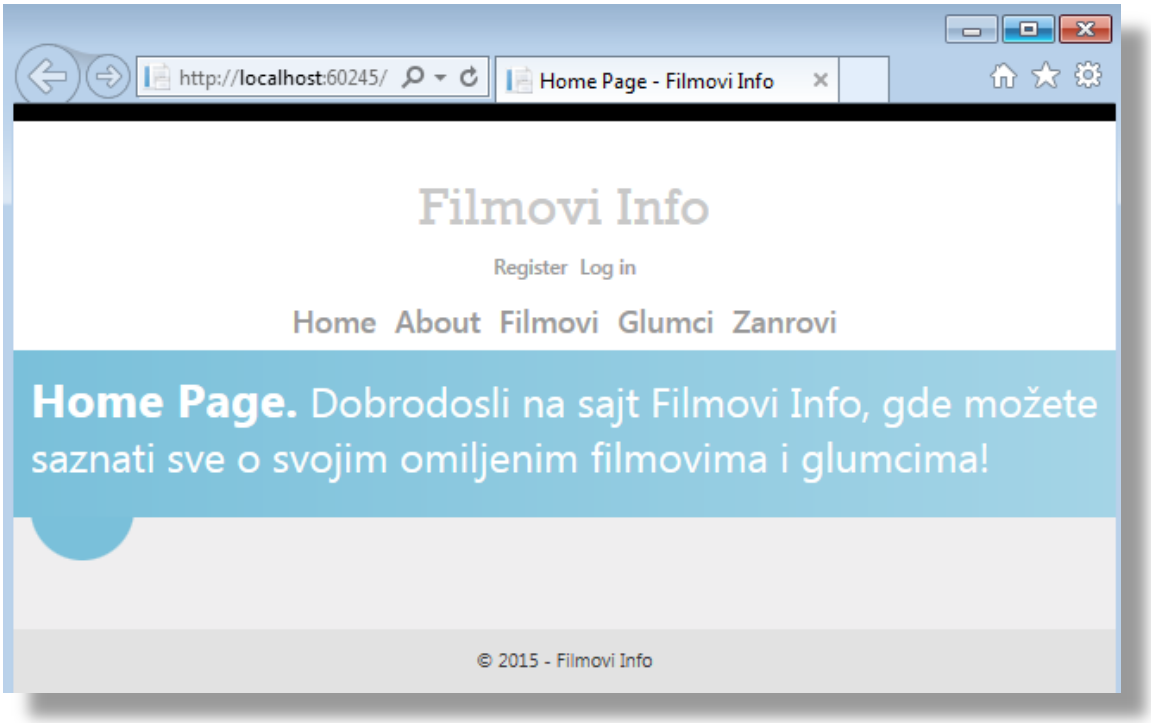
```
    <section class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1>@ViewBag.Title.</h1>
                <h2>@ViewBag.Message</h2>
            </hgroup>
            <p>
                To learn more about ASP.NET MVC visit
                <a href="http://asp.net/mvc" title="ASP.NET MVC
Website">http://asp.net/mvc</a>.
                The page features <mark>videos, tutorials, and samples</mark> to help
you get the most from ASP.NET MVC.
                If you have any questions about ASP.NET MVC visit
                <a href="http://forums.asp.net/1146.aspx/1?MVC" title="ASP.NET MVC
Forum">our forums</a>.
            </p>
        </div>
    </section>
       }
```

U kontroleru *HomeController* (*Controllers\HomeController.cs*) biće promenjena pozdravna poruka, koja se pojavljuje na *Home* strani.

```
namespace FilmoviInfo.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC
application.";[17]

            return View();
        }
    }
}
```

Nakon ovih izmena, izgled veb aplikacije u *Internet Explorer* brauzeru je kao na slici 19:

[17] Dobrodosli na sajt Filmovi Info, gde možete saznati sve o svojim omiljenim filmovima i glumcima.




*Slika 19: izgled veb aplikacije*

### 8.1.2 Kreiranje Data Model-a

Kreiranje *EDM*-a biće izvršeno primenom *cod-first* pristupa. *POCO* klase će sadržati atribute koji će odgovarati kolonama tabela, odgovarajuće *get* i *set* metode za pristup tim atributima i odgovarajuće konstruktore. Na osnovu ovih klasa biće generisane tabele baze podataka.

U nastavku sledi primer *POCO* klase *Film*, sa atributima *FilmID*, *Naziv*, *Opis*, *Obrisano* i sa odgovarajućim *set* i *get* metodima.

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

namespace FilmoviInfo.Models
{
    public class Film
    {
        public Guid FilmID { get; set; }
        public string Naziv { get; set; }
        public string Opis { get; set; }
        public int Obrisano { get; set; }

        public virtual ICollection<RelacijaFilmGlumac> RelacijaFilmoviGlumci { get;
set;}
    }
}
```

*Entity Framework* tumači atribut koji u nazivu sadrži *ID* kao primarni ključ odgovarajuće tabele koja odgovara klasi, a u ovom slučaju to je tabela *Film* i kolona *FilmID*.

Atribut *RelacijaFilmoviGlumci* je relacioni atribut. Relacioni atribut skladišti druge unose koji su u relaciji sa objektom ove klase. U ovom slučaju atribut *RelacijaFilmoviGlumci* će skladištiti sve relacije sa klasom *Glumci*. Relacioni atribut se obično definiše kao virtual *ICollection* kako bi mogao da iskoristi određene funkcionalnosti *Entity Framework*-a.

Folder *Models* će sadržati klasu RelacijaFilmGlumac koja se sastoji od sledećih atributa:

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

namespace FilmoviInfo.Models
{
    public enum TipUloge
    {
        A, B
    }

    public class RelacijaFilmGlumac
    {

        public Guid RelacijaFilmGlumacID { get; set; }
        public Guid FilmID { get; set; }
        public Guid GlumacID { get; set; }
        public TipUloge? TipUloge { get; set; }
        public string Opis { get; set; }
        public int Obrisano { get; set; }

        public virtual Film Film { get; set; }
        public virtual Glumac Glumac { get; set; }

    }
}
```

Klasa *Glumac* je navedena u nastavku.

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

namespace FilmoviInfo.Models
{
    public class Glumac
    {
        public Guid GlumacID { get; set; }
           public string Ime { get; set; }
           public string Prezime { get; set; }
           public string UmetnickoIme { get; set; }
           public string Pol { get; set; }
           public DateTime DatumRodjenja { get; set; }
           public int Obrisano { get; set; }

        public virtual ICollection<RelacijaFilmGlumac> RelacijaFilmoviGlumci { get;
set; }
```

```
    }
}
```

### 8.1.3 Kreiranje Database Context

Pored ovih *POCO* klasa koje odgovaraju tabelama iz baze podataka, neophodno je generisati klasu *DbContext*[18], koja će vršiti povezivanje *POCO* klasa sa *ObjectContext*-om. *ObjectContext* je osnovna klasa za manipulisalje upitima i rad sa entitetskim podacima kao objektima.

U okviru projekta biće napravljen folder *DataAccess* (desni klik na ime projekta *Add* ➤ *New Folder*) i u njemu klasa *SumetnostContext*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Data.Entity;
using FilmoviInfo.Models;


namespace FilmoviInfo.DataAccess
{
    public class SUmetnostContext   : DbContext
    {
        public DbSet<Film> Filmovi { get; set; }
        public DbSet<RelacijaFilmGlumac> RelacijaFilmoviGlumci { get; set; }
        public DbSet<Glumac> Glumci { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }


   }
```

Kreira se *DbSet*[19] za svaku entitetsku klasu koja je prethodno napravljena. Skup entiteta se odnosi na tabelu baze podataka, a entitet se odnosi na pojedinačan red iz tabele.

Metod *OnModelCreating* obezbeđuje da imena generisanih tabela ne budu u množini, na primer da se tabela *Glumac* ne generiše kao *Glumacs* (da su imena tabela na engleskom jeziku ovo bi imalo većeg smisla, na primer da se tabela zove *Actor*, onda bi se bez primenjivanja ove metode generisalo *Actors*).

[18] *System.Data.Entity.DbContext*
[19] *DbSet* klasa predstavlja skup entiteta nad kojima će se izvršavati operacije za kreiranje, čitanje, ažuriranje i brisanje.

### 8.1.4 SQL Server Express LocalDB

*LocalDB* je verzija *SQL Server Express Database Engine*[20] koja započinje rad na zahtev i radi u korisničkom režimu. *LocalDB* je verzija koja je posebno napravljena za programere. Veoma je jednostavna za instalaciju i ne zahteva nikakva podešavanja, a ipak omogućava rad sa *T-SQL* jezikom, kao i rad sa klijentskom stranom kao regularni *SQL Server Express*. U stvari, programeri koji koriste *SQL Server* više ne moraju da instaliraju i podešavaju punu instancu *SQL Server Express* na svojim lokalnim mašinama. *SQL Server Express* se ne koristi u produkciji[21] veb aplikacije. Ako jednostavnost i ograničenja *LocalDB* odgovaraju aplikaciji koja je koristi, u produkciji te aplikacije, moguće je nastaviti korišćenje *LocalDB*. *LocalDB* radi u posebnom režimu izvršenja *SQL Server Express* koji omogućava rad sa bazom podataka, kao sa *.mdf* fajlom. *LocalDB* fajlovi baze podataka čuvaju se u folderu *App_Data* veb projekta. *LocalDB* se podrazumevano instalira uz *Visual Studio 2012* i kasnijih verzija, dok se za verziju *Visual Studio 2010 LocalDB* mora instalirati naknadno.

Ova veb aplikacija će koristiti *LocalDB*, tako da će baza biti smeštena u *App_Data* folderu projekta kao *.mdf* fajl. U okviru *Web.config* fajla potrebno je dodati string za konekciju u okviru kolekcije *connectionStrings*.

```
<add name="SUmetnostContext" connectionString="Data Source=(LocalDb)\v11.0;Initial
Catalog=SedmaUmetnost;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\
SedmaUmetnost.mdf" providerName="System.Data.SqlClient" />
```

*Entity Framework* (podrazumevano) string za konekciju traži po imenu koje mora biti isto kao I *DbContext* klasa. U ovom primeru ta klasa je SUmetnostContext. Moguće je da se ne navede string za konekciju i u tom slučaju *Entity Framework* će automatski napraviti taj string, ali *.mdf* fajl neće biti smešten u *App_data* folderu.

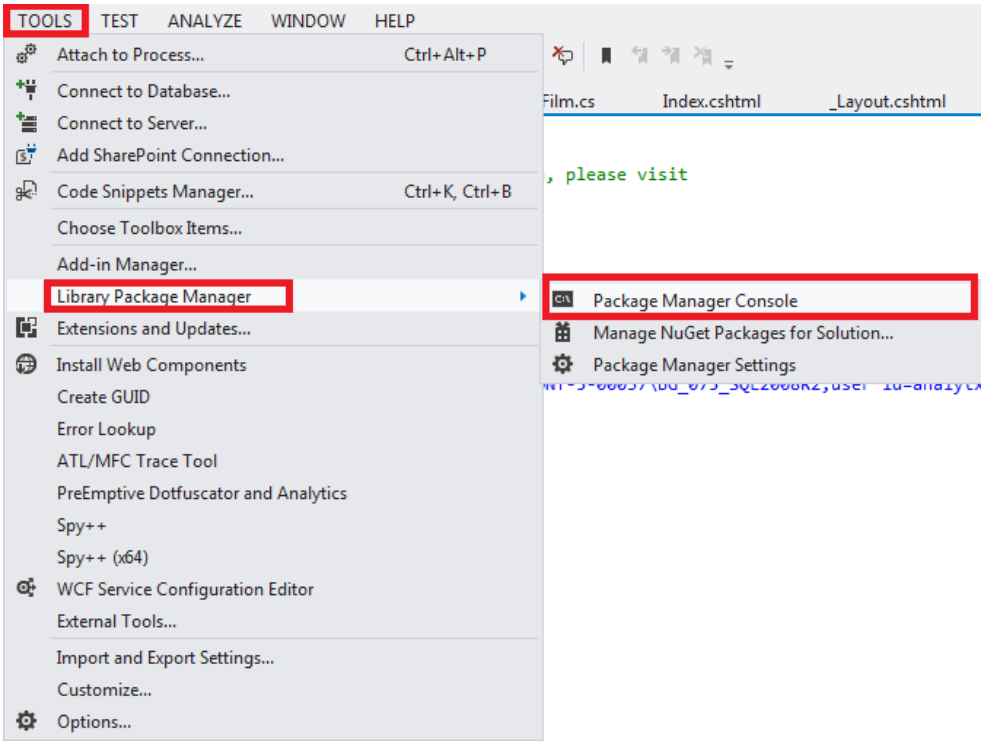
### 8.1.5 Kôd-prvo migracija

Kada se od početka pravi neka aplikacija, model baze podataka će se često menjati i svaki put kad dođe do neke promene, biće neophodna sinhronizacija sa bazom podataka. Svaki put kada dođe do promene modela moguće je podesiti da *Entity Framework* automatski vrši brisanje i ponovno kreiranje baze podataka. Brisanje i ponovno kreiranje je moguće izvršavati u ranim fazama kreiranja aplikacije, ali kada aplikacija pređe u produkciju, onda se obično očekuje da se šema baze podataka ažurira, a ne briše i ponovo kreira. Kod

[20] *Microsoft SQL Server Express* je verzija *SQL Server*-a koja omogućava upravljanje relacionim bazama podataka i koji je besplatan za preuzimanje i može se slobodno koristiti.
[21] Kada je aplikacija u upotrebi od strane krajnjeg korisnika.

pristupa *cod-first* funkcija migracije podataka omogućava da se podaci ažuriraju bez prethodnog brisanja i ponovnog kreiranja. Migracija se radi na sledeći način:

1. Iz linije menija odabrati tab *Tools* ➤ *Library Package Manager* ➤ *Package Manager Console*

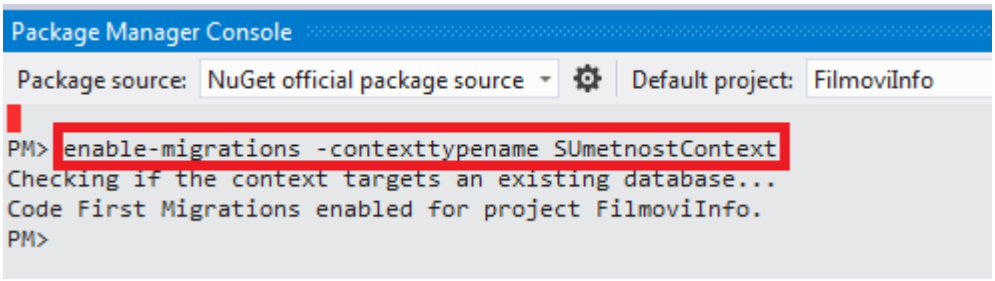



*Slika 20: Prvi korak migracije*

2. Uneti komandu

```
enable-migrations -contexttypename SUmetnostContext
```

Nakon uspešno izvršene komande, dobija se sledeca poruka:




*Slika 21: pokretanje komande u Package Manager Console-i*

Ovim je napravljen *Migration* folder u okviru projekta, a u okviru tog foldera je generisana klasa *Configuration.cs* koja će biti korišćena za migraciju. Ova klasa sadrži metod *Seed* koji se poziva prilikom kreiranja baze podataka kao i prilikom ažuriranja posle izmena

na modelu podataka. Svrha metoda *Seed* je da omogući ubacivanje test podataka u bazu podataka nakon pravljenja ili ažuriranja modela podataka pristupom kôd-prvo.

### 8.1.6 Korišćenje metoda Seed

Nakon kreiranja baze podataka primenom pristupa kôd-prvo migracija, izvršava se poziv metoda *Seed.* Takođe, metod *Seed* se poziva svaki put kad se ažurira baza podataka. Svrha metoda *Seed* je da omogući da se podaci unesu u bazne tabele pre nego što aplikacija po prvi put pristupa bazi podataka. U ranijim verzijama pristupa kôd-prvo, dok nije postojala opcija migracije, metod *Seed* je ubacivao testne podatke jer sa svakom promenom modela tokom razvoja baza podataka je morala biti potpuno obrisana i ponovo napravljena od početka. Sa pojavom mogućnosti migracije, testni podaci se, nakon izmene modela, zadržavaju u bazi tako da obično nije potrebno uključiti test podatke u metod *Seed*.

U nastavku biće uneti testni podaci u metod *Seed*.

```
namespace FilmoviInfo.Migrations
{
    using System;
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;
    using FilmoviInfo.Models;

    internal sealed class Configuration :
DbMigrationsConfiguration<FilmoviInfo.DataAccess.SUmetnostContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = false;
        }

        protected override void Seed(FilmoviInfo.DataAccess.SUmetnostContext context)
        {
            var films = new List<Film>
            {
                new Film { FilmID = Guid.NewGuid(),   Naziv = "Now You See Me",
                    Opis = "An FBI agent and an Interpol detective track a team of
                            illusionists who pull off bank heists during their
                            performances and reward their audiences with the money",
                    Obrisano = 0 },
                new Film { FilmID = Guid.NewGuid(),   Naziv = "Catch Me If You Can",
                    Opis = "A true story about Frank Abagnale Jr., who, before his
                            19th birthday, successfully conned millions of dollars'
                            worth of checks as a Pan Am pilot, doctor, and legal
                            prosecutor.",
                    Obrisano = 0 }
            };
            films.ForEach(s => context.Filmovi.AddOrUpdate(p => p.Naziv, s));
            context.SaveChanges();

            var glumci = new List<Glumac>
```

```
            {
                new Glumac {GlumacID =  Guid.NewGuid(), Ime = "Jon",
                            Prezime = "Hamm",
                            UmetnickoIme = "Jon Hamm",Pol = "M",
                            DatumRodjenja = DateTime.Parse("1971-03-10"),
                            Obrisano = 0, },

                new Glumac {GlumacID =  Guid.NewGuid(), Ime = "Leonardo",Prezime =
                            "DiCaprio", UmetnickoIme = "Leonardo DiCaprio",Pol = "M",
                            DatumRodjenja = DateTime.Parse("1974-11-11"), Obrisano =
                            0, }
            };
            glumci.ForEach(s => context.Glumci.AddOrUpdate(p => p.UmetnickoIme, s));
            context.SaveChanges();

            var relacija = new List<RelacijaFilmGlumac>
            {
                new RelacijaFilmGlumac {
                    RelacijaFilmGlumacID = Guid.NewGuid(),
                    FilmID = films.Single(s => s.Naziv == "Now You See Me").FilmID,
                    GlumacID = glumci.Single
                               (c => c.UmetnickoIme == "Jon Hamm" ).GlumacID,
                    TipUloge = TipUloge.Glavna
                },
                new RelacijaFilmGlumac {
                    RelacijaFilmGlumacID = Guid.NewGuid(),
                    FilmID = films.Single
                             (s => s.Naziv == "Catch Me If You Can").FilmID,
                    GlumacID = glumci.Single
                             (c => c.UmetnickoIme == "Leonardo DiCaprio" ).GlumacID,
                    TipUloge = TipUloge.Glavna
                },
                new RelacijaFilmGlumac {
                    RelacijaFilmGlumacID = Guid.NewGuid(),
                    FilmID = films.Single
                            (s => s.Naziv == "Catch Me If You Can").FilmID,
                    GlumacID = glumci.Single
                            (c => c.UmetnickoIme == "Tom Hanks" ).GlumacID,
                    TipUloge = TipUloge.Sporedna
                }

            };
            foreach (RelacijaFilmGlumac e in relacija)
            {
                var enrollmentInDataBase = context.RelacijaFilmoviGlumci.Where(
                    s =>
                         s.Film.FilmID == e.FilmID &&
                         s.Glumac.GlumacID == e.GlumacID).SingleOrDefault();
                if (enrollmentInDataBase == null)
                {
                    context.RelacijaFilmoviGlumci.Add(e);
                }
            }
            context.SaveChanges();
        }
    }
}
```

Metod *Seed* uzima objekat konteksta baze podataka[22] kao ulazni parametar, a u telu metoda se taj objekat koristi za dodavanje novih entitete u bazu podataka. Za svaku vrstu

[22] *FilmoviInfo.DataAccess.SUmetnostContext*

entiteta (*Film*, *Glumac* i *RelacijaFilmGlumac*), u kodu se kreira kolekcija novih objekata koja se dodaje odgovarajućim *DbSet* atributima, a zatim se izmene čuvaju u bazi podataka. Nije neophodno da se posle svake grupe entiteta pozove metod *SaveChanges()*, kao što je to urađeno u ovom primeru, ali ovo je urađeno da bi se lakše locirao problem ako dođe do neke greške prilikom unošenja entiteta u bazu podataka.

U telu metoda *Seed* koristi se i metod *AddOrUpdate*. Pošto se metod *Seed* poziva prilikom svake migracije, podaci se ne mogu samo ubaciti u bazu podataka, jer može da se desi da redovi koje pokušavamo da dodamo već postoje u bazi podataka pošto je već izvrsena migracija. *AddOrUpdate* metod sprečava greške koje će se desiti ako se pokuša unošenje reda koji već postoji i omogućava da se unesu izmene koje su napravljene prilikom testiranja aplikacije. Prvi parametar koji je prosleđen metodu *AddOrUpdate* omogućava da se proveri da li taj red već postoji u bazi. U gornjem primeru

```
context.Filmovi.AddOrUpdate(p => p.Naziv, s))
```

atribut *Naziv* moguće je koristiti za proveru jer je naziv filma jedinstven[23].

U delu koda koji dodaje relacije između filma i glumca ne koristi se metod *AddOrUpdate*. Ovde se proverava da li relacija već postoji, i ako ne postoji unosi je u bazu podataka. Ovaj pristup će sačuvati promene koje su se desile na relaciji kad se migracija pokrene. U okviru petlje *foreach* proverava se da li svaki element liste već postoji u bazi.Ako relacija nije pronađena u bazi podataka, onda se ta relacija dodaje. Kada se migracija pokreće po prvi put, baza podataka će biti prazna, tako da će sve relacije biti dodate.

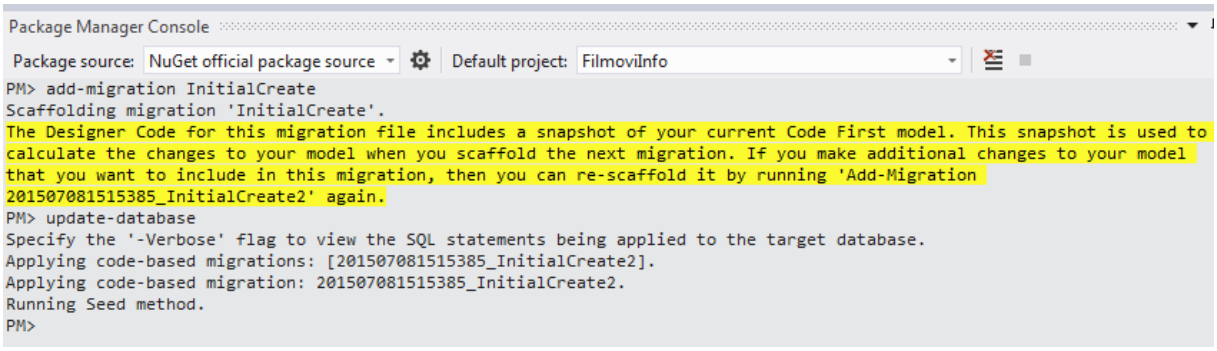
### 8.1.7 Pravljenje i izvršavanje prve migracije

U *Package Manager Console* prozoru potrebno je uneti sledeće komande:

- add-migration InitialCreate
- update-database

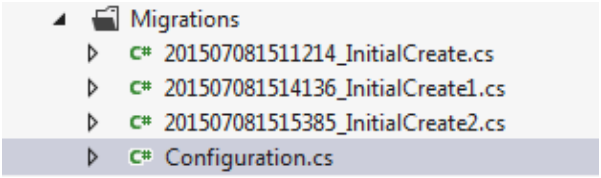
Kad se navedene komande izvrše uspešno, onda se u *Package Manager Console* prozoru ispiše sledeća poruka:

[23] pretpostavka je da je naziv filma jedinstven




*Slika 23: uspešno izvršena migracija*

Svaka izvršena komanda za migraciju u folderu *Migrations*, dodaje klasu *201507081511214_InitialCreate.cs* koja sadrži kôd koji kreira bazu podataka.



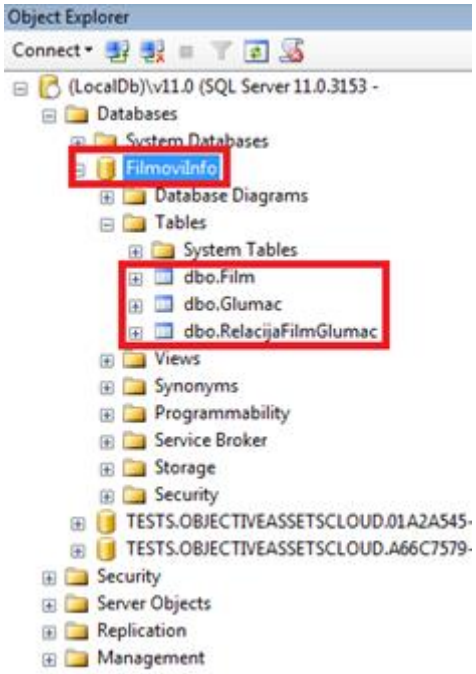

*Slika 24: Kreirana klasa nakon migracije*

```
namespace FilmoviInfo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class InitialCreate : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Film",
                c => new
                    {
                        FilmID = c.Guid(nullable: false),
                        Naziv = c.String(),
                        Opis = c.String(),
                        Obrisano = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.FilmID);

            CreateTable(
                "dbo.RelacijaFilmGlumac",
                c => new
                    {
                        RelacijaFilmGlumacID = c.Guid(nullable: false),
                        FilmID = c.Guid(nullable: false),
                        GlumacID = c.Guid(nullable: false),
                        TipUloge = c.Int(),
                        Opis = c.String(),
                        Obrisano = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.RelacijaFilmGlumacID)
                .ForeignKey("dbo.Film", t => t.FilmID, cascadeDelete: true)
```

```
                .ForeignKey("dbo.Glumac", t => t.GlumacID, cascadeDelete: true)
                .Index(t => t.FilmID)
                .Index(t => t.GlumacID);

            CreateTable(
                "dbo.Glumac",
                c => new
                    {
                        GlumacID = c.Guid(nullable: false),
                        Ime = c.String(),
                        Prezime = c.String(),
                        UmetnickoIme = c.String(),
                        Pol = c.String(),
                        DatumRodjenja = c.DateTime(nullable: false),
                        Obrisano = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.GlumacID);

        }

        public override void Down()
        {
            DropIndex("dbo.RelacijaFilmGlumac", new[] { "GlumacID" });
            DropIndex("dbo.RelacijaFilmGlumac", new[] { "FilmID" });
            DropForeignKey("dbo.RelacijaFilmGlumac", "GlumacID", "dbo.Glumac");
            DropForeignKey("dbo.RelacijaFilmGlumac", "FilmID", "dbo.Film");
            DropTable("dbo.Glumac");
            DropTable("dbo.RelacijaFilmGlumac");
            DropTable("dbo.Film");
        }
    }
}
```

*update-database* komanda poziva metod *Up()* koji kreira tabele baze podataka, a zatim se pokreće metod *Seed* da bi se popunile kreirane tabele.

Baza podataka *FilmInfo* je kreirana.




*Slika 25: Baza podataka FilmoviInfo*

Rezultat izvršavanja *Seed* metoda se može videti izčitavanjem podataka iz kreiranih tabela:

Results | Messages

| | FilmID | Naziv | Opis | Obrisano |
|---|---|---|---|---|
| 1 | 93F4F11D-6203-4D18-982A-0D5B06BD2DF9 | Now You See Me | An FBI agent and an Interpol detective track a t... | 0 |
| 2 | 3FE234E8-2BF1-4A03-9313-60DDDAFBFEF8 | Forrest Gump | Forrest Gump, while not intelligent, has accident... | 0 |
| 3 | 8F71BB21-4D62-48AB-939D-D538485DFB9C | Catch Me If You Can | A true story about Frank Abagnale Jr., who, befo... | 0 |

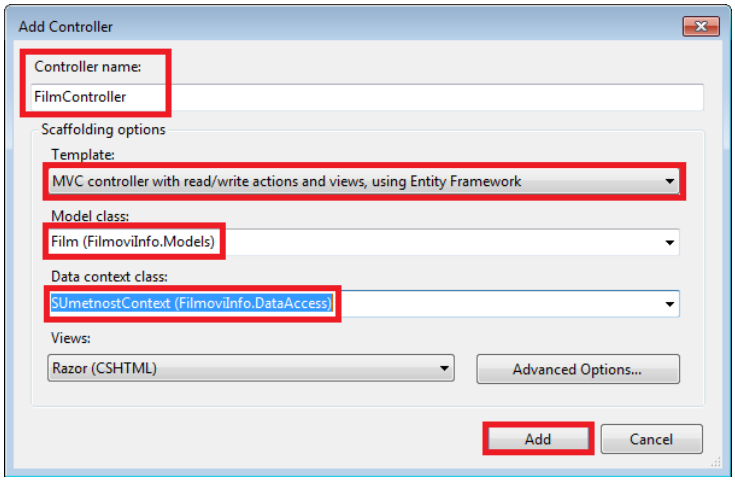
| | GlumacID | Ime | Prezime | UmetnickoIme | Pol | DatumRodjenja | Obrisano |
|---|---|---|---|---|---|---|---|
| 1 | C9EF7685-ABE0-4D73-8D82-331E018DB3E9 | Sally | Field | Sally Field | Z | 1946-11-04 00:00:00.000 | 0 |
| 2 | B5A74813-E448-4493-B86E-83B9D06BE0E1 | Leonardo | DiCaprio | Leonardo DiCaprio | M | 1974-11-11 00:00:00.000 | 0 |
| 3 | 5C67CE75-5903-438E-86BD-BF536952B3F6 | Jon | Hamm | Jon Hamm | M | 1971-03-10 00:00:00.000 | 0 |
| 4 | 2D1E4F2F-90BC-4EB9-8EA0-D248F7798251 | Tom | Hanks | Tom Hanks | M | 1956-07-09 00:00:00.000 | 0 |

| | RelacijaFilmGlumacID | FilmID | GlumacID | TipUloge | Opis | Obrisano |
|---|---|---|---|---|---|---|
| 1 | 7F524771-39C3-4FD7-A690-0C5F8AD303EE | 8F71BB21-4D62-48AB-939D-D538485DFB9C | 2D1E4F2F-90BC-4EB9-8EA0-D248F7798251 | 1 | NULL | 0 |
| 2 | 8E60F55E-FAEA-450E-A247-0F4DD37F3023 | 93F4F11D-6203-4D18-982A-0D5B06BD2DF9 | 5C67CE75-5903-438E-86BD-BF536952B3F6 | 0 | NULL | 0 |
| 3 | D5F940A5-6CF2-4A09-AD11-21609F15C9D8 | 3FE234E8-2BF1-4A03-9313-60DDDAFBFEF8 | C9EF7685-ABE0-4D73-8D82-331E018DB3E9 | 0 | NULL | 0 |
| 4 | D1437596-30B4-4871-AFBB-E6B0A716C919 | 3FE234E8-2BF1-4A03-9313-60DDDAFBFEF8 | 2D1E4F2F-90BC-4EB9-8EA0-D248F7798251 | 0 | NULL | 0 |
| 5 | 906A896D-371E-4A3F-8BD0-F7A9F0B9C7FA | 8F71BB21-4D62-48AB-939D-D538485DFB9C | B5A74813-E448-4493-B86E-83B9D06BE0E1 | 0 | NULL | 0 |

*Slika 26: Podaci iz tabela baze podataka FilmoviInfo*

# 9 Kontroleri i pogledi

1. Kontroler *Film* se kreira desnim klikom tastera miša na folder *Controllers* u *Solution Explorer* i odabirom opciju *Add* ➢ *Controller*. *Add Controller* prozor treba da izgleda kao na slici:



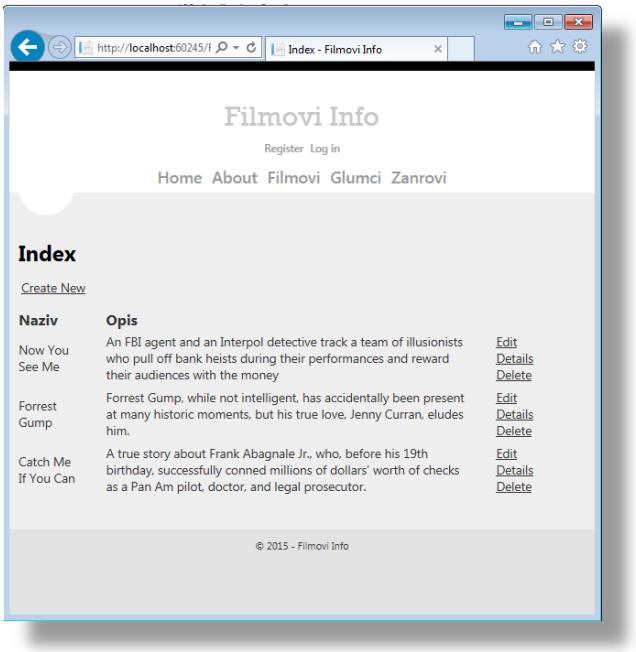

*Slika 27: Pravljenje kontrolera*

2. U kreiranoj klasi kontrolera *FilmController.cs* metod *Index* prikazuje listu filmova iz *Filmovi* entiteta. A *view Index.cshtml* prikazuje sve postojeće atribute:

```
@model IEnumerable<FilmoviInfo.Models.Film>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Naziv)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Opis)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Obrisano)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Naziv)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Opis)
        </td>
        <td>
```

```
            @Html.DisplayFor(modelItem => item.Obrisano)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.FilmID }) |
            @Html.ActionLink("Details", "Details", new { id=item.FilmID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.FilmID })
        </td>
    </tr>
}

</table>
```

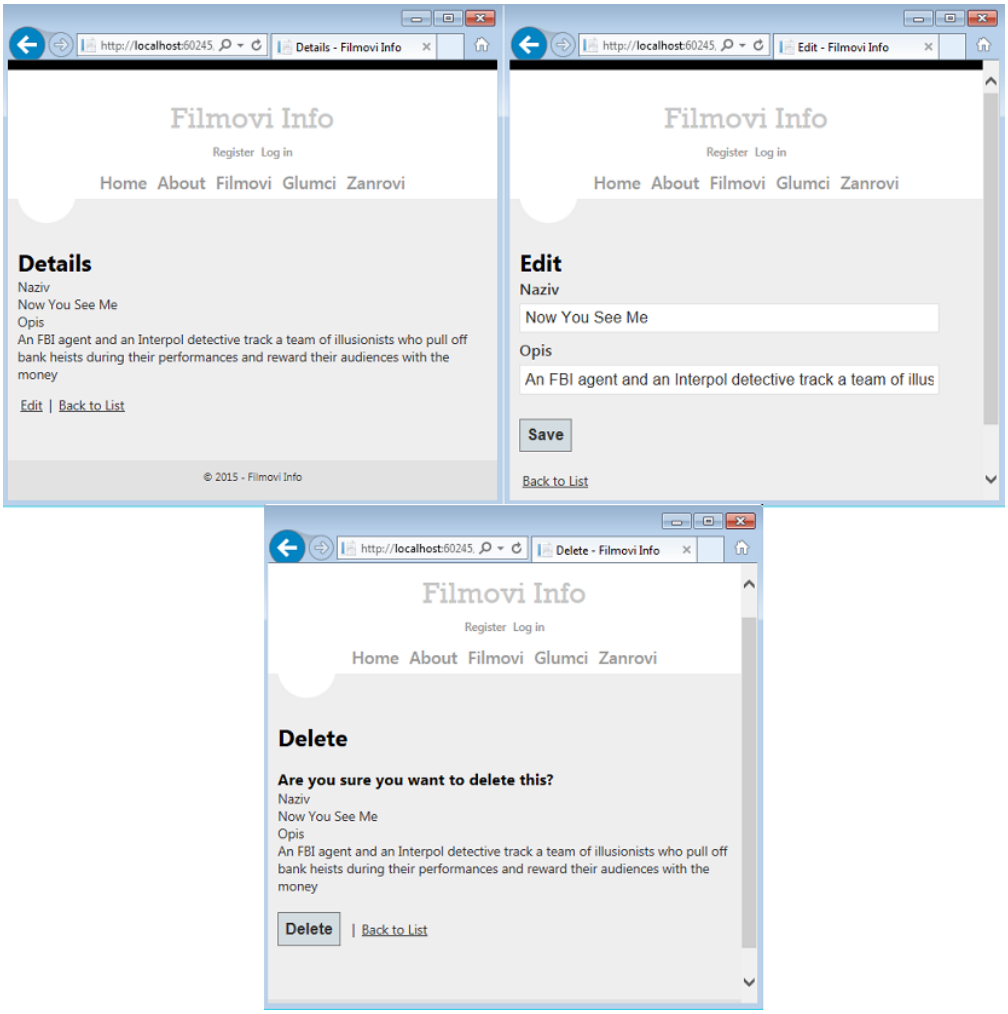
Nakon pokretanja[24] projekta, dobija se stranica koja izgleda kao na slici:




*Slika 28: izgled aplikacije*

Opcije pregleda, ažuriranja i brisanja su također napravljene.

[24] Projekat je moguće pokrenuti i pomoću prečice CTRL+F5

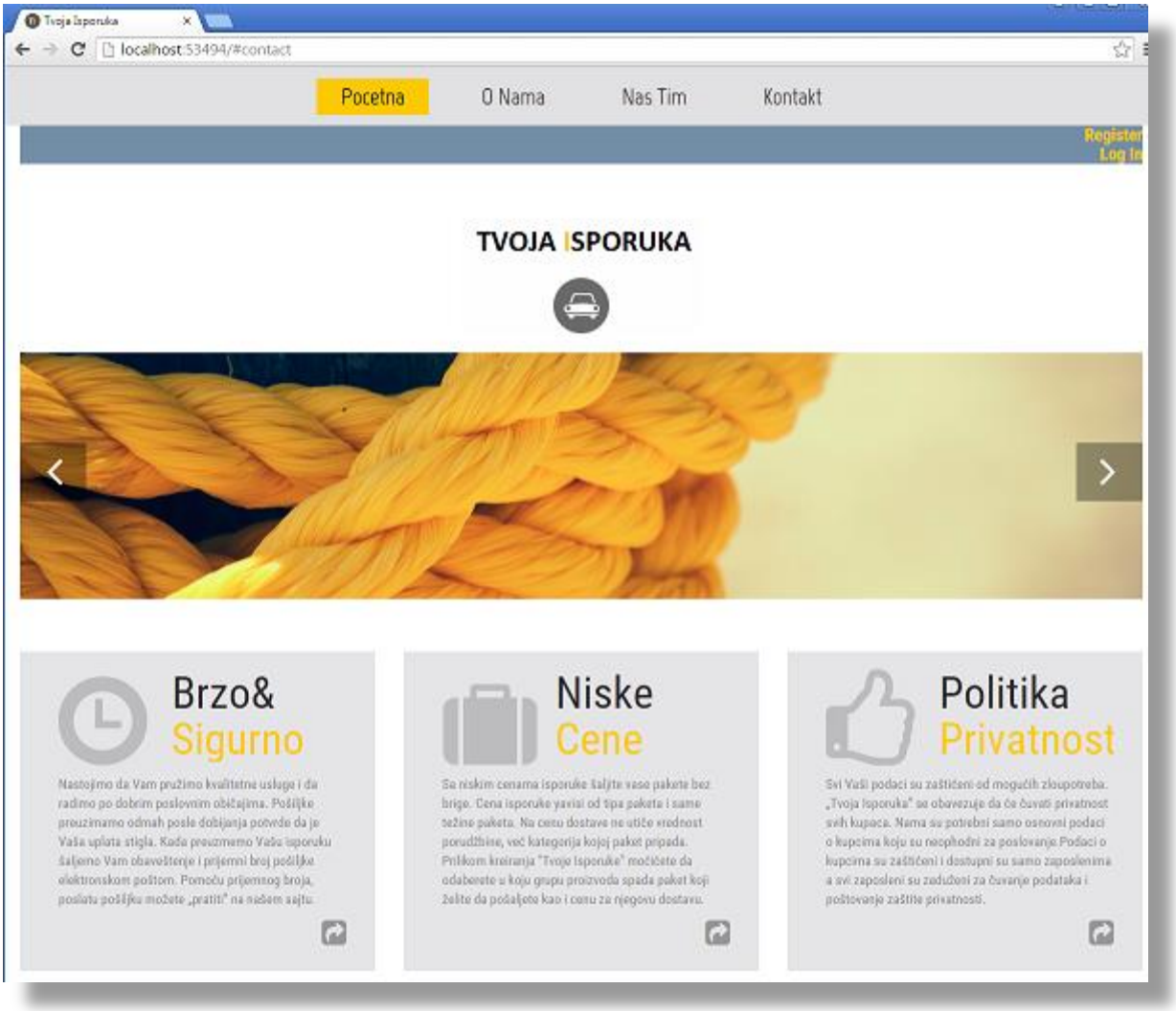



*Slika 29: Tri stranice za detalj, ažuriranje i brisanje podataka*

# 10 Aplikacija

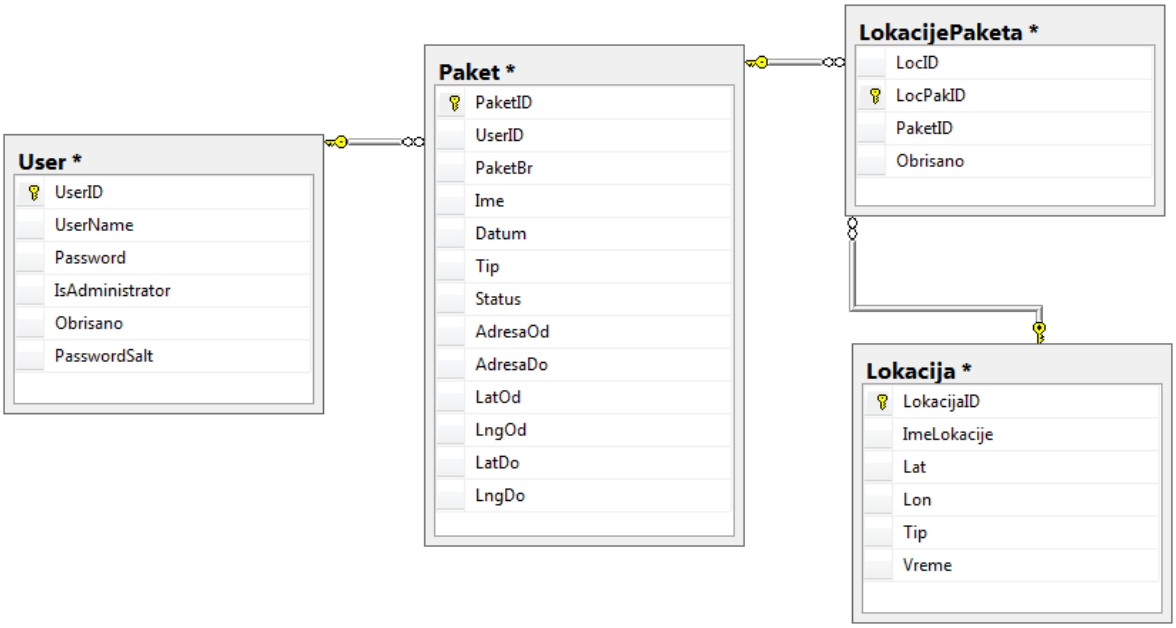
Alikacija je pisana u *ASP .NET MVC 4* okviru (*Microsoft .NET Framework Version* 4.5,51209) sa korišćenjem *Entity Framework* verzije v4.0.30319. Sistem za upravljanje bazom podataka je *MS SQL Server 2012* i razvojno okruženje je *Microsoft Visual Studio Professional 2012* verzija 11.0.50727.1.

Tvoja Isporuka je veb aplikacija koja korisnoku omogućava da naruči isporuku pošiljke sa jedne adrese ne drugu adresu.



## 10.1 Model baze podataka

Na diagramu su prikazane tabele baze podataka sa odgovarajućom atributima.

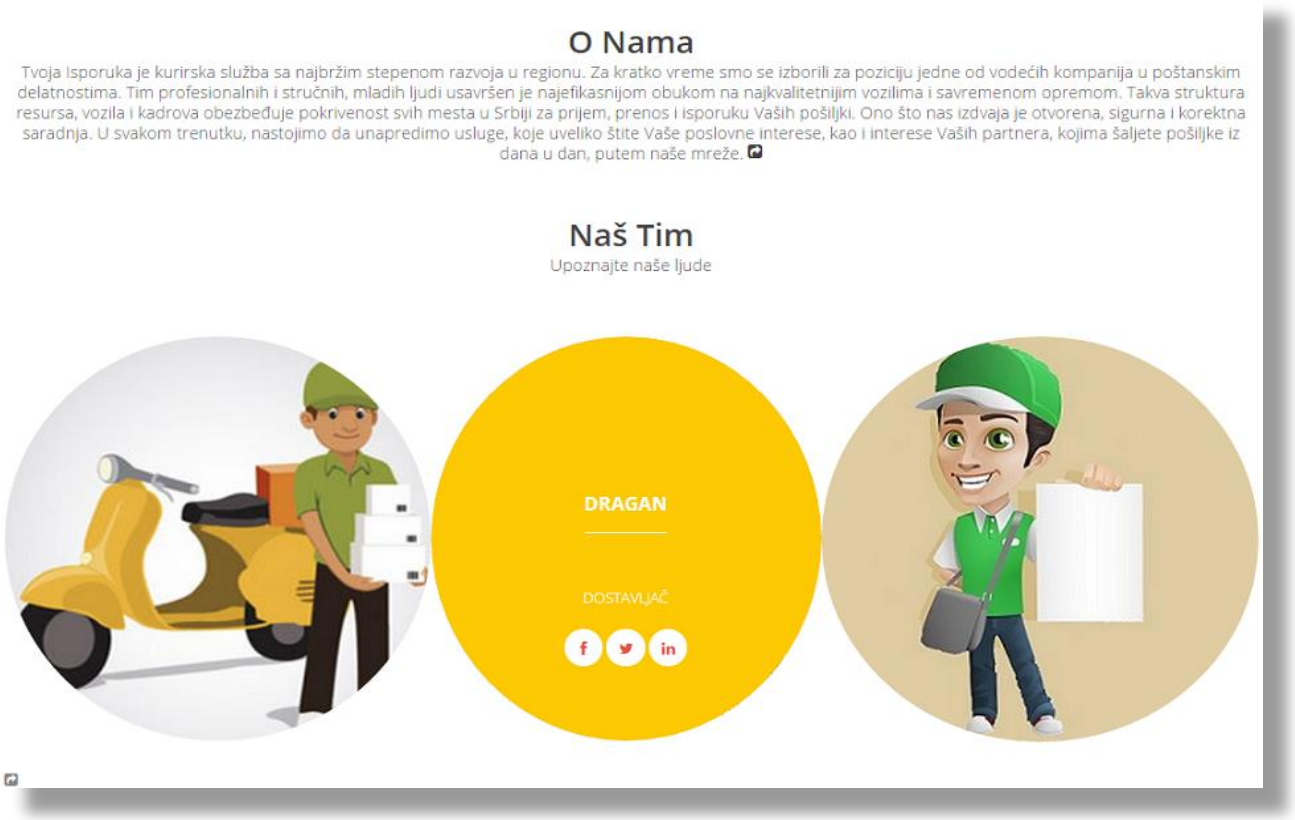



Baza podataka TIsporuka se sastoji od sledećih tabela:

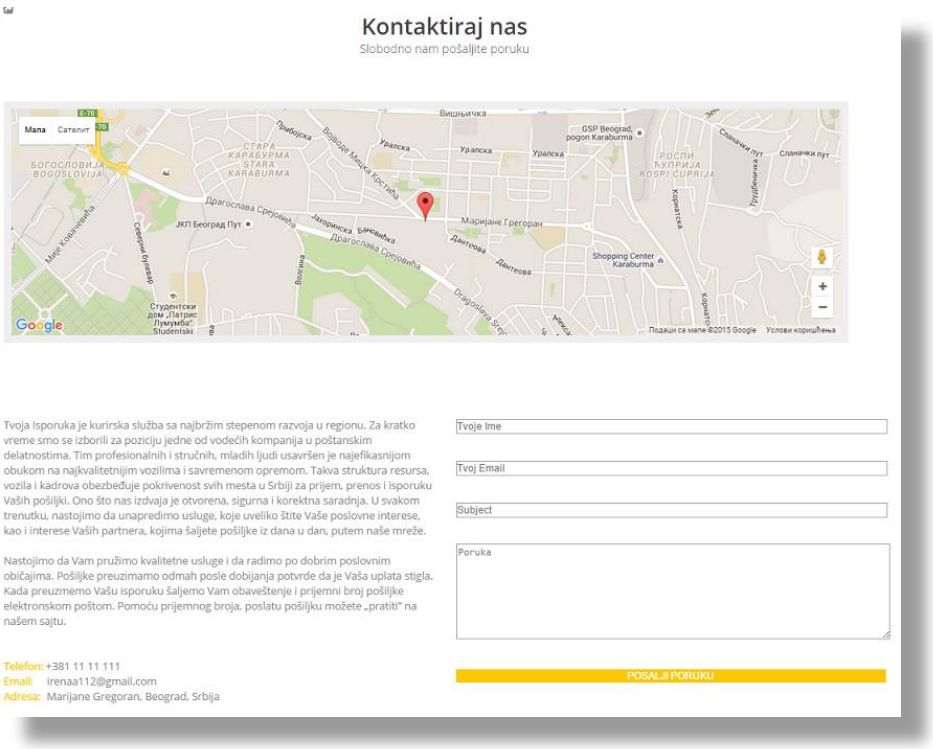
- User – u ovoj tabeli se nalaze svi korisnici aplikacije
- Paket – čuva informacije o svim paketima
- Lokacija – sadrži informacije o svim lokacijama
- LokacijePaketa – sadrži vezu između konkretnog paketa I odgovarajuće lokacije.

## 10.2 Funkcionalnost aplikacije

Kada korisnik nije ulogovan iz linije menija mogu se odabrati sledeći tabovi: O Nama, Naš Tim i Kontakt.

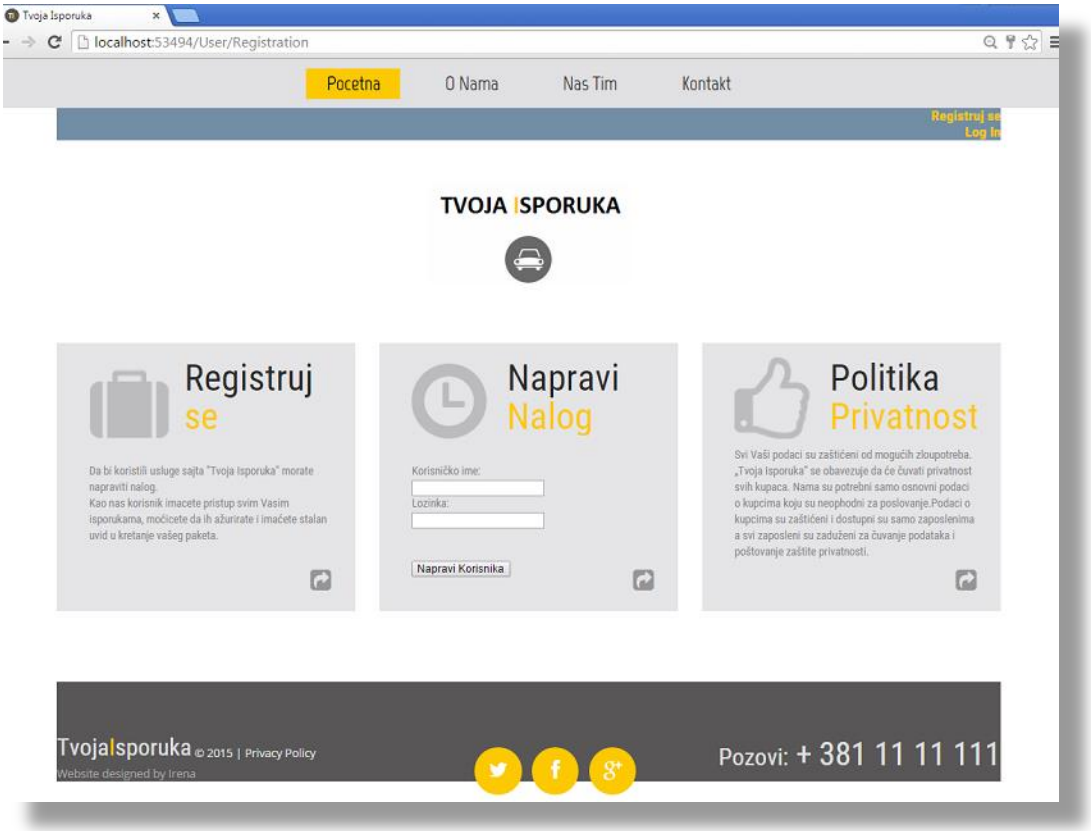



Kada se pređe mišom preko slike člana tima, pojavi se žuta poyadina na kojoj se može videti ime zaposlenog, njegova funkcija u timu i linkovi ka društvenim mrežama koje su vezani za tog zaposlenog.

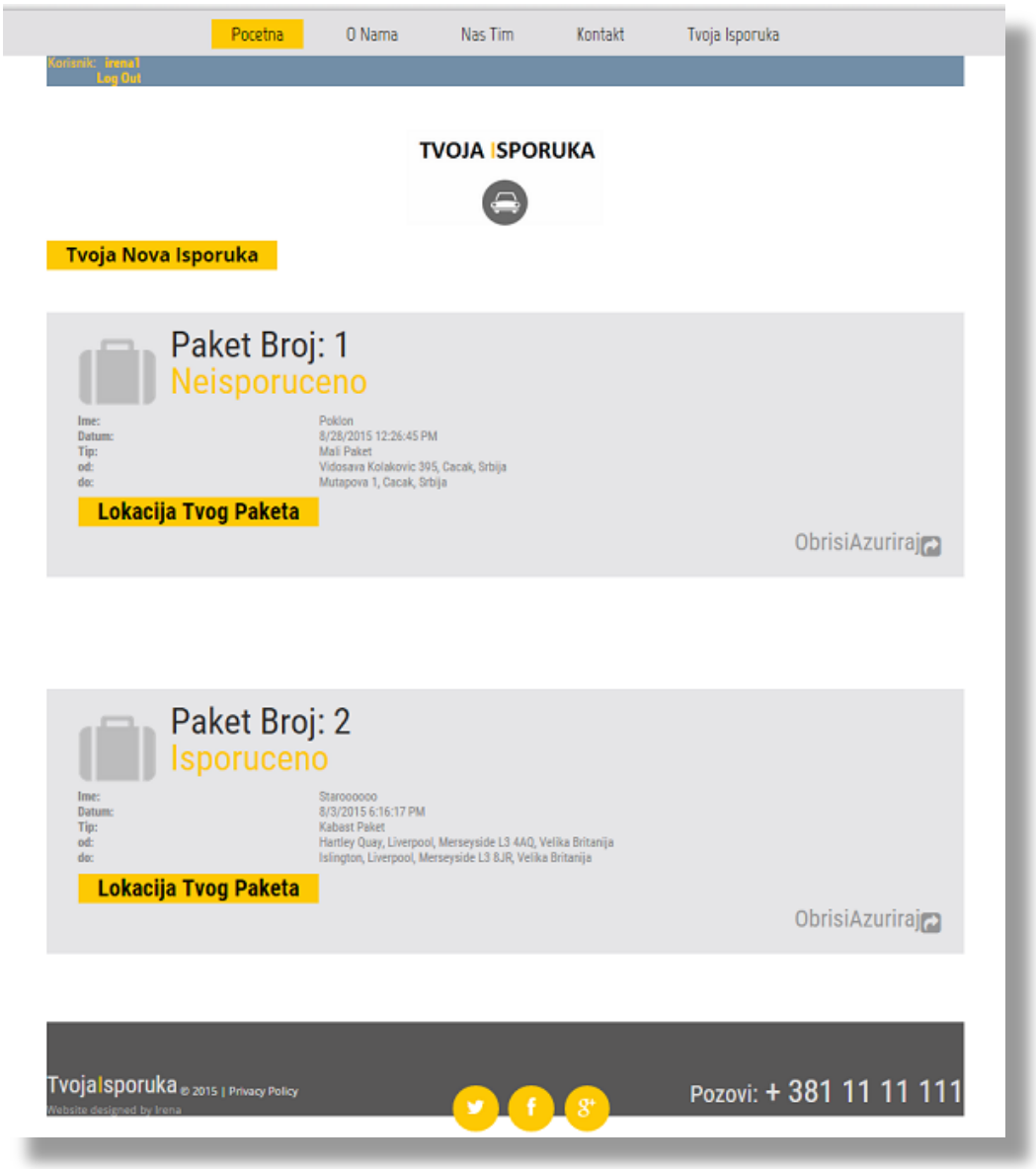



U delu Kontaktiraj nas postoji mogućnost slanja e-poruke. Potrebno je da posetilac sajta (koji ne mora da bude i korisnik) ostavi svoj e-adresu i napiše tekst poruke.

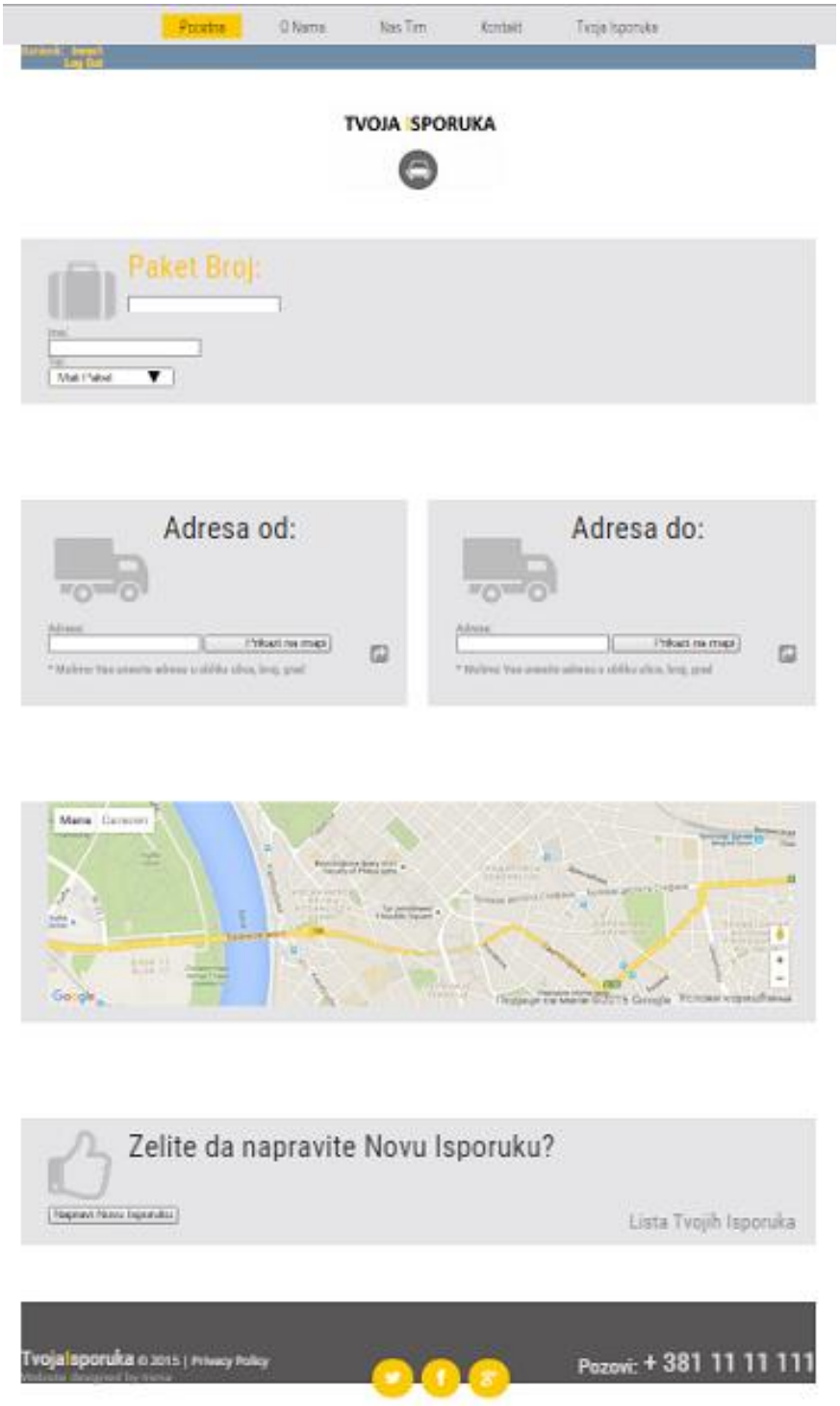
Da bi isporuka bila omogućena potrebno je da korisnik ima napravljen nalog. Registracija Korisnika se vrši klikom na opciju „Registruj se".



Kada se korisnik registruje i uloguje moguće je da izvrši naruđbinu isporuke. Osnovni element aplikacije, a samim tim i baze podataka je paket koji se šalje. Korisnik unosi informacije o paketu, kao što su ime paketa, tip paketa, adresu sa koje lokacije treba da se pokupi paket i na koju lokaciju paket treba da se dostavi. Korisnik nakon kreiranja pakete može da prati kretanje svog paketa. Administrator aplikacije ima mogućnost da unosi nove lokacije po paketima u zavisnosti gde se taj paket nalazi i kad se paket isporuči status paketa će se promeniti sa „Neisporučen“ na „Isporučen“.

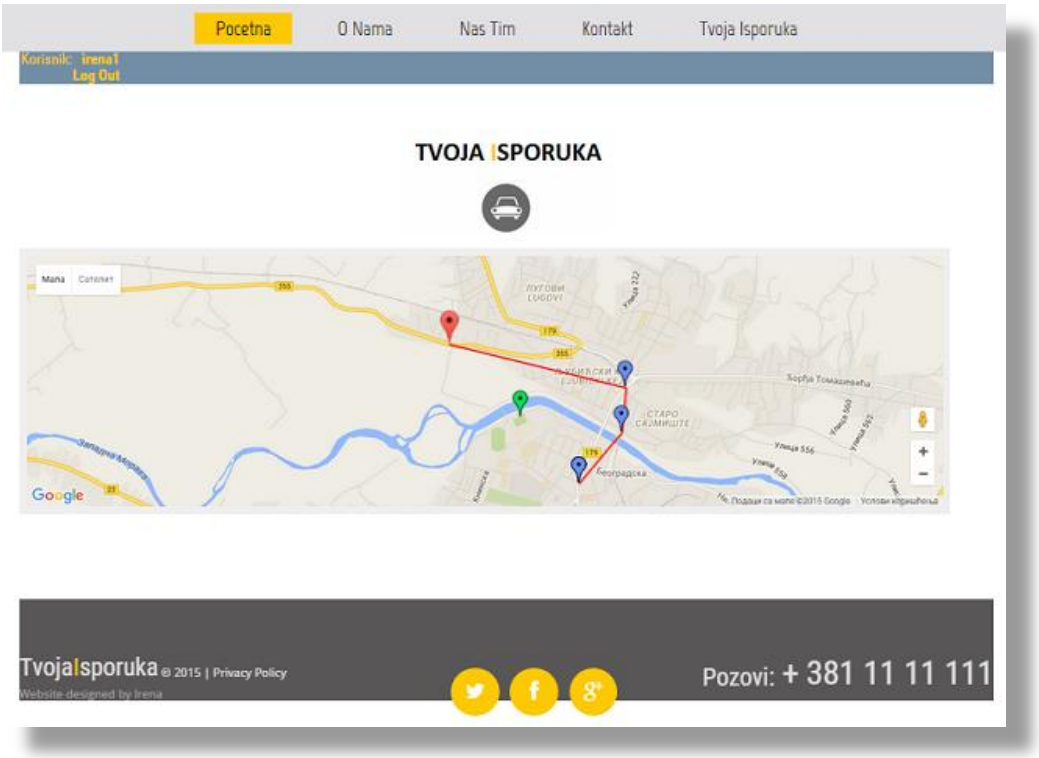


Aplikacija omogućava da se vrši unošenje novih paketa, ažuriranje i brisanje već postojećih paketa.

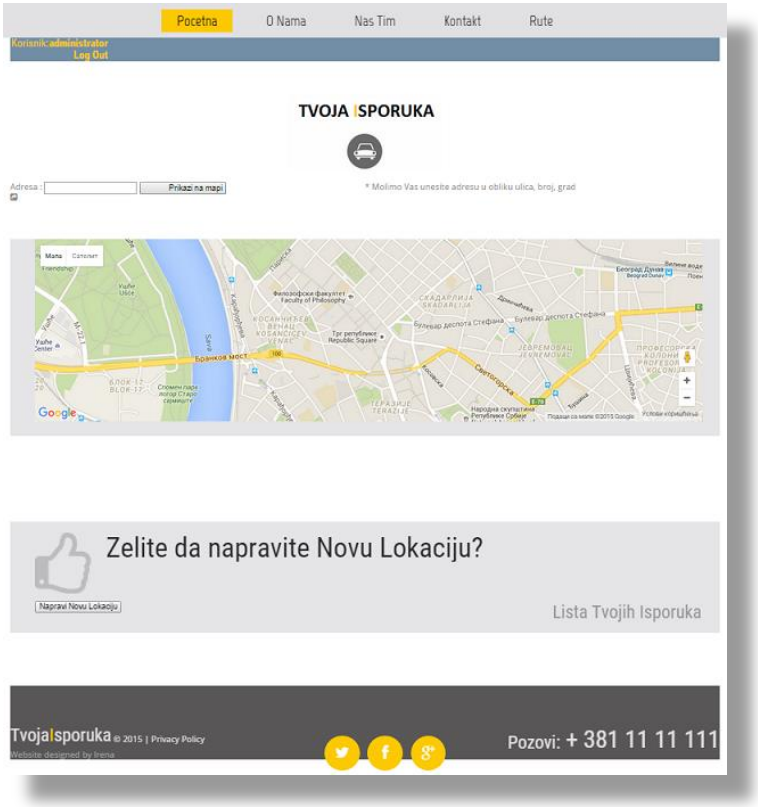



Na stranici na kojij se unosi novi paket postoji orikaza na mapi adresa koje se unose kao početna i krajnja adresa isporuke paketa.

Korisnik ima uvid u kretanje svog paketa na opciji „Lokacija Tvog Paketa“. Crvena oznaka lokacije označava početnu lokaciju paketa, plave tačke označavaju kretanje paketa dok je keajnja tačka kretanja paketa označena zelenom bojom.




Administrator ima mogućnost dodavanja lokacija za svaki od paketa.



# 11 Zaključak

*ADO.NET Entity Framework* je alat kompanije *Microsoft* za objektno relaciono mapiranje (*ORM*) koji omogućava programerima da rade sa relacionim bazama podataka pri čemu se podaci posmatraju kao objekte. *Entity Framework* omogućava pisanje upita koristeći *LINQ*. Nedostatak *Entity Framework* je to što kada dođe do promene šeme baze podataka, model neće biti automatski ažuriran. Jedna od prednosti *Entity Framework* je produktivnost. Uprkos svojim ograničenjima, dizajner integrisana u okruženju *Visual Studio* veoma pojednostavljuje proces mapiranja podataka iz baze podataka sa odgovarajućim objektima. Prednost *Entity Framework* je i ta što omogućava lako i brze sprovođenje *CRUD* operacija (kreiranje, čitanje, ažuriranje i brisanje podataka). *Entity Framework* je namenjen za razvoj poslovnih aplikacija koje mogu da podržavaju *MS SQL* baze podataka, kao i druge baze podataka kao što su *Oracle*, *DB2*, *MySql* i još mnogo toga.

## 12 Literatura


[1] Concepts of Database Management. s.l. : Course Technology, Pratt P., Adamski J.,Cengage Learning, 2008.

[2] Uvod u relacione baze podataka, Dr Gordana Pavlović Lažetić

[3] Baze podataka, Branislav Lazarević, Zoran Marjanović, Nenad Aničić

[4] Object-relation mapping, https://en.wikipedia.org/wiki/Object-relational_mapping

[5]http://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B

[6]http://www.entityframeworktutorial.net/EntityFramework5/entity-framework5-introduction.aspx

[7] https://msdn.microsoft.com/en-us/data/ff830362.aspx

[8] https://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf

[9] Julia Lerman, Programming Entity Framework, 2010

[10]http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

[11] Pro.ASP.NET.MVC.4.Framework, Adam Freeman, 2012

[12] https://msdn.microsoft.com/en-us/data/ff830362.aspx

[13] http://stackoverflow.com/

[14]http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

[15] https://msdn.microsoft.com/en-us/library/bsc2ak47(v=vs.110).aspx

[16] https://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf

[17] http://www.asp.net/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

[18] https://sh.wikipedia.org/wiki/Relacijska_baza_podataka

[19] Entity Framework, http://www.asp.net/entity-framework

[20]http://www.entityframeworktutorial.net/EntityFramework5/entity-framework5-introduction.aspx

[21] https://entityframework.codeplex.com/