Katedra za razvoj informacijskih sustava
Programsko inženjerstvo, akademska godina: 2012/13

Laboratorijske vježbe 10

Rad sa bazom podataka kroz ADO.NET Entity Framework

Sažetak

Na današnjim vježbama ćemo obraditi koncepte ORM alata, i to kroz Entity Framework, koji postepeno postaje glavna .NET tehnologija za rad sa podacima. Kao i ostali ORM alati omogućava apstrahiranje baze podataka, na način da korisnik radi sa konceptualnim modelom. Korisnik više ne mora sam preslikavati strukturu baze podataka u klase aplikacije, nego to za njega radi ORM. Na taj način se generira velik dio infrastrukturalnog kôda koji je prije morao pisati sam programer, te se ubrzava proces izrade aplikacija.

Ključne riječi: ADO.NET, ORM, Entity Framework

Sadržaj

1. Uvod

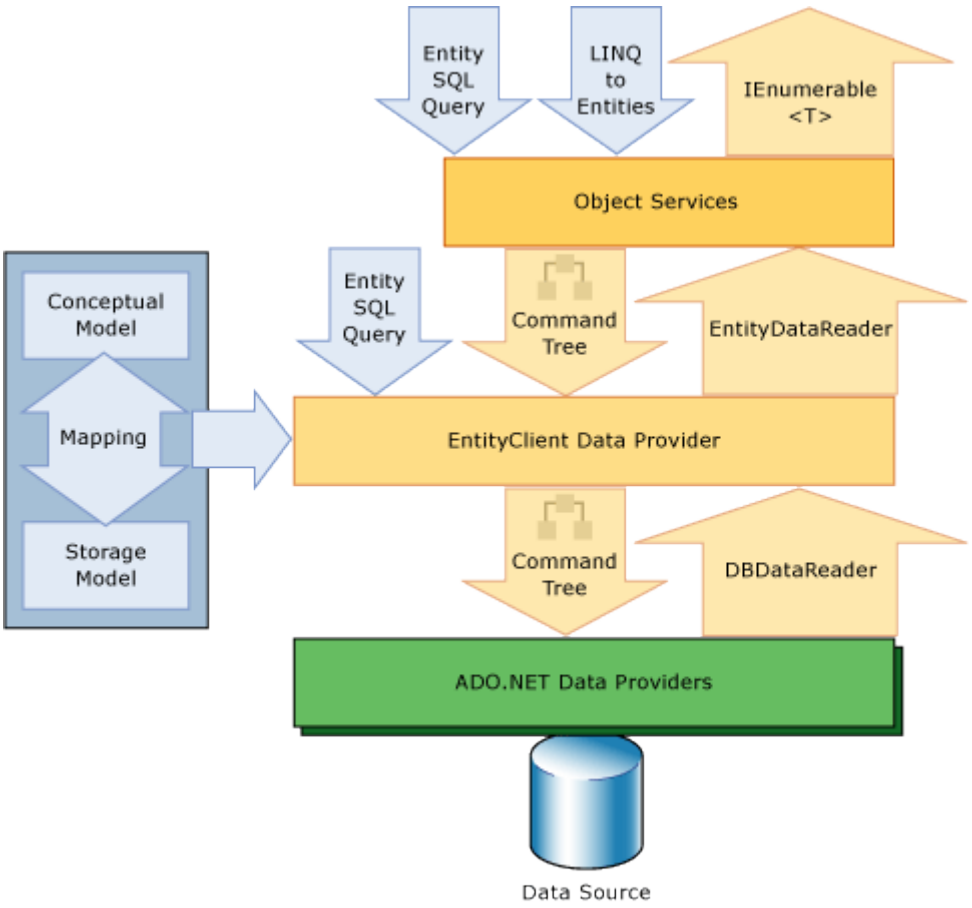
Podaci se u relacijskim bazama podataka nalaze u obliku relacija sa zapisima, strukturiranim i prilagođenim tako da zadovoljavaju pravila oblikovanja baza podataka (normalizacija). Ta pravila osiguravaju bazu podataka od pojave redundancije, nekonzistentnosti podataka, različitih anomalija, te omogućuju brzo pretraživanje. Međutim, istovremeno uzrokuju da podaci nisu uvijek spremljeni na najprirodniji način za razumijevanje. S druge strane, većina današnjih aplikacija, koje koriste podatke iz relacijskih baza podataka, su napisane u objektno-orijentiranih programskim jezicima. Suprotno od relacijskih baza podataka, objektni pristup pokušava prikazati entitete na prirodan način kao objekte definirane njihovim svojstvima i ponašanjima. Iako struktura klase vrlo često odgovara strukturi tablice u relacijskoj bazi podataka, njihove strukture nisu nužno jednake. Ta različitost uzrokuje problem pretvorbe podataka iz jednog pristupa u drugi, u literaturi poznat pod imenom objektno-relacijska neusklađenost (eng. *object-relational impedance mismatch*).

2. ADO.NET Entity Framework

Jedan od ORM (eng. Object-Relational Mapping) alata je i Entity Framework ,razvijen od strane Microsofta 2008. godine u okviru .NET Framework-a 3.5. Iako je u početku bio dočekan sa izvjesnom dozom skepticizma i nije bio najbolje prihvaćen, zahvaljujući brojnim poboljšanjima je sada prerastao u glavnu .NET tehnologiju za pristup bazi podataka, što je potvrdio i sam Microsoft.

Arhitekturalno Entity Framework se i dalje temelji na ADO.NET tehnologiji, konkretnije DataReader i Command objektima, ali omogućava korisniku da radi s podacima na višoj razini apstrakcije. Zapravo, glavna korist Entity Framework-a je da oslobađa programera brige o strukturi baze podataka, na način da se sada radi sa konceptualnim modelom podataka koji reflektira poslovne objekte naše aplikacije. Nije više potrebno pisati i izvršavati SQL upite za dohvaćanje podataka iz baze i unos podataka u bazu. Sada Entity Framework omogućava da se sa podacima radi u obliku specifičnih domenskih objekata, bez da se moramo brinuti o tome u kojoj su konkretno tablici i atributu pohranjeni podaci.

2.1. Arhitektura Entity Framework-a



Slika 1 Arhitektura Entity Framework-a

Na slici iznad možemo vidjeti pojednostavljenu arhitekturu Entity Framework-a. Sastoji se od sljedećih elemenata:

1. **Entitetni podatkovni model** (eng. Entity Data Model - EDM) – srž Entity Framework-a, definira poveznicu između konceptualnog modela nad kojim programer radi, i baze podataka. Sastoji se od 3 XML datoteke (ovisno o verziji), od kojih svaka ima posebnu funkciju:
   a. Konceptualni model (eng. Conceptual Model) – opisuje domenske klase i veze između njih. Definira se uz pomoć CSDL jezika (eng. Conceptual Schema Definition Language). Sadrži popis svih entiteta i veza između njih, te detaljan opis strukture svakog od entiteta.
   b. Model pohrane (eng. Storage Model) – predstavlja ekvivalent konceptualnom modelu, ali s razlikom da opisuje strukturu baze podataka, njenih tablica, pogleda, pohranjenih procedura, ključeva i veza. Definira se uz pomoć SSDL jezika (eng. Store Schema Definition Language).
   c. Model preslikavanja (eng. Mapping Model) – služi za neutraliziranje razlika između konceptualnog modela i modela pohrane. On specificira pravila

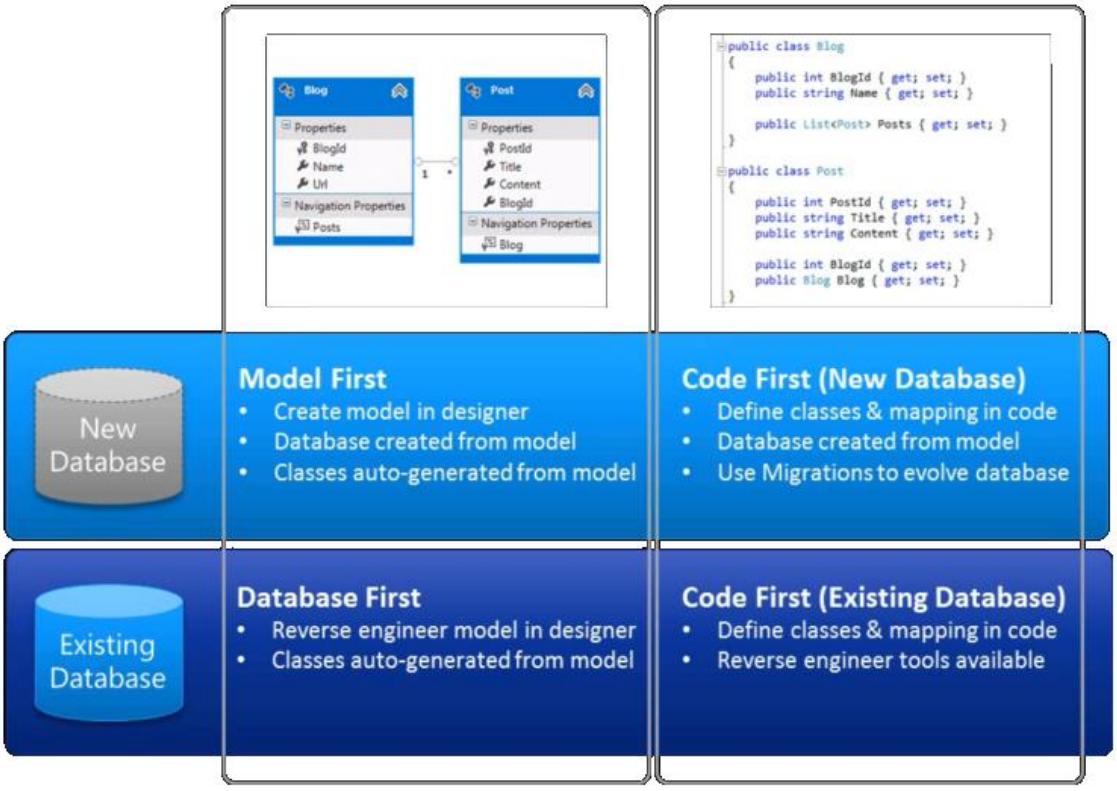
preslikavanja iz jednog modela u drugi, i obrnuto, uz pomoć MSL jezika (eng. Mapping Specification Language).

2. **Objektne usluge** (eng. Object Services) – sloj zadužen za upravljanje objektima u Entity Framework-u. Na temelju proslijeđenog upita (Entity SQL ili LINQ to SQL) i konceptualnog modela, sloj objektnih usluga kreira komandno stablo (eng. Command Tree) koje prosljeđuje sloju entitetnog klijenta (eng. Entity Client). S druge strane, kada od entitetnog klijenta zaprimi podatke u formatu definiranom konceptualnim modelom, uz pomoć tih podataka i konceptualnog modela rekonstruira domenske objekte.
3. **Entitetni klijent** (eng. Entity Client Data Provider) – sloj zadužen za komunikaciju za bazom podataka. Ne komunicira izravno sa bazom podataka, nego preko ADO.NET pružatelja podataka. Koristi entitetni podatkovni model (sloj mapiranja i sloj pohrane) kako bi komandno stablo dobiveno od sloja objektnih usluga pretvorio u SQL naredbe. Također, kada od ADO.NET sloja dobije podatke u tabličnom obliku DataReader-a, koristi konceptualni model kako bi podatke oblikovao da budu razumljivi sloju objektnih usluga.
4. **LINQ to Entities** – je LINQ dijalekt i glavni upitni jezik u Entity Framework-u. Omogućava pisanje tipiziranih upita nad konceptualnim modelom, koji vraćaju gotove objekte. Iako LINQ to Entities radi isključivo nad objektima, i dalje se on mora prevesti u standardni SQL kako bi se mogao izvršiti nad bazom.
5. **Entity SQL** – koristio se prije nego je nastao LINQ to Entities jezik, međutim koristi se u određenim slučajevima i danas. Vrlo je složen, ali ima i neke prednosti u odnosu na L2E, kao što su: zadaje se u tekstualnom obliku, pa ga je lakše dinamički generirati, sadrži više funkcije, može se izvršavati izravno u sloju entitetnog klijenta.

2.2. Tokovi rada u Entity Framework-u

Entity Framework trenutno nudi 4 načina rada, nazvana tokovi rada (eng. Workflows). Da bi odlučili koji od njih koristiti moramo odgovoriti na sljedeća pitanja:

1. Da li radimo sa postojećom bazom podataka ili baza podataka još ne postoji?
2. Da li objektni model želimo kreirati uz pomoć grafičkog alata – dizajnera, ili pisanjem kôda.



Slika 2 Tokovi rada u Entity Framework-u

Na slici iznad možemo vidjeti koji tok rada bi bio prikladan u ovisnosti o odgovorima na postavljena pitanja, te je moguće odabrati neki od sljedećih pristupa:

1. **Prvo model** (eng. Model First) – prikladan kada nemamo postojeću bazu podataka nego treba kreirati novu, i kada objektni model želimo kreirati uz pomoć grafičkog alata. U dizajneru prvo kreiramo objektni model, a onda se na temelju njega generira baza podataka, i kôd za klasa.
2. **Prvo baza podataka** (eng. Database First) – prikladan kada već imamo izgrađenu bazu podataka, iz koje tada u dizajneru reverznim inženjeringom generiramo model, a zatim iz modela generiramo kôd za klase.
3. **Prvo kôd – nova baza** (eng. Code First – New Database) – prikladan kada nemamo postojeću bazu podataka nego treba kreirati novu, i kada objektni

model želimo kreirati pisanjem koda. Tada objektni model kreiramo u kôdu pisanjem klasa i definiranjem pravila mapiranja, te iz tako kreiranog modela generiramo bazu podataka.

4. **Prvo kôd – postojeća baza** (eng. Code First – Existing Database) – prikladan kada već imamo izgrađenu bazu podataka, a model želimo kreirati u kôdu pisanjem klasa i definiranjem pravila mapiranja.

Bottom-up i Top-down dizajn

Izbor gore navedenih pristupa ovisi i o osobnim preferencijama i navikama. Neki preferiraju tzv. bottom-up pristup koji najveću važnost pridaje podacima, odnosno bazi podataka. Zato taj pristup uvijek kreće od definiranja baze podataka. Nasuprot tome, top-down pristup prednost daje procesima kojima se bavi aplikacija, pa zbog toga zagovara prvo kreiranje modela, a tek onda podataka.

3. Kreiranje aplikacije pristupom „Prvo baza podataka“

Na vježbama ćemo obraditi pristup koji pretpostavlja postojanje gotove baze podataka, na temelju koje ćemo generirati objektni model (eng. Database First). Ukoliko vas zanimaju ostali pristupi možete pogledati dodatnu literaturu navedenu na kraju vježbi.

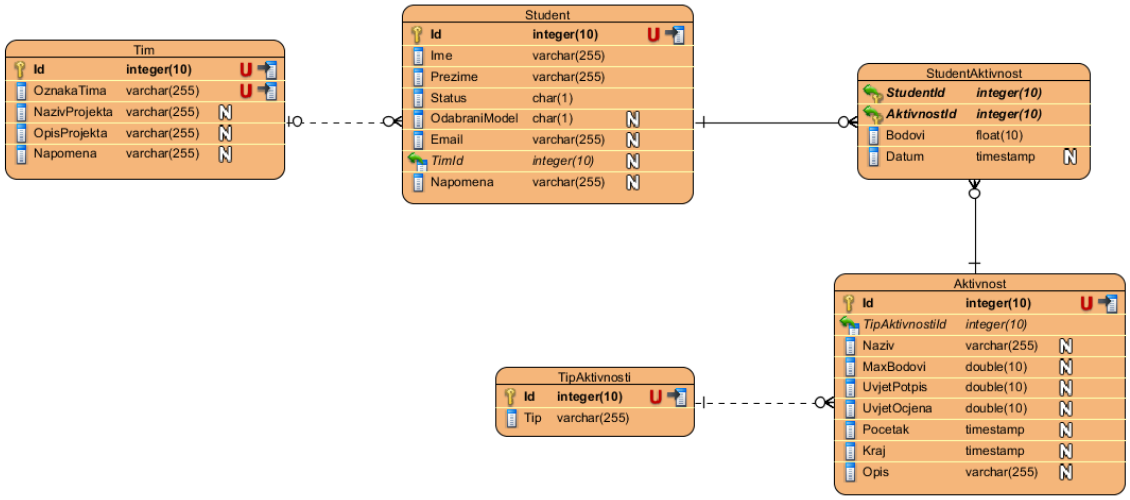
Praktičan rad sa Entity Framework-om ćemo pokazati na sljedećoj aplikaciji:

Aplikacija omogućava vođenje evidencije o studentima na kolegiju programsko inženjerstvo, njihovom timu, odabranom projektu, te aktivnostima koje izvršavaju. Funkcionalnosti koje aplikacija mora imati implementirane su:

1. *Dodavanje, izmjena i brisanje timova, dodjela studenata u timove.*
2. *Dodavanje, izmjena i brisanje studenata, te praćenje aktivnosti studenata.*
3. *Dodavanje, izmjena i brisanje aktivnosti.*
4. *Dodavanje, izmjena i brisanje tipova aktivnosti.*

Sve informacije će se pohranjivati u SQL Server Compact bazu podataka navedenu na ERA dijagramu. S obzirom da se želimo fokusirati na izradu pristupa bazi podataka, sa GitHub sustava ćemo preuzeti projekt (https://github.com/PI2013FOI/Lab_3_1_EvidencijaStudenata.git) koji sadrži gotovu bazu podataka, i definirano korisničko sučelje aplikacije.

Dio aplikacije ćemo napraviti na vježbama, a ostatak aplikacije možete uraditi doma za vježbu.

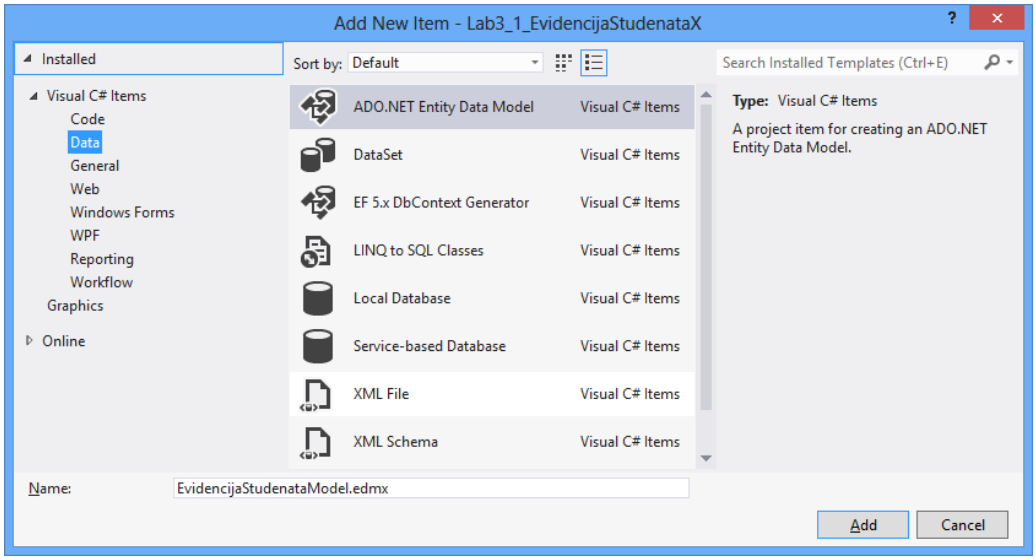


Slika 3 ERA model baze podataka

3.1. Spajanje na bazu i generiranje modela

S obzirom da naša aplikacija ima već izgrađeno grafičko sučelje možemo odmah preći na sljedeći korak, a to je spajanje na bazu podataka i generiranje entitetnog podatkovnog modela. To možemo napraviti na sljedeći način:

1. U Solution Explorer-u kliknite desnom tipkom miša na projekt, te odaberite **Add -> New Item**.
2. U listi elemenata odaberite **ADO. NET Entity Data Model**, pod **Name** upišite „*EvidencijaStudenataModel*“, te kliknite **Add**.



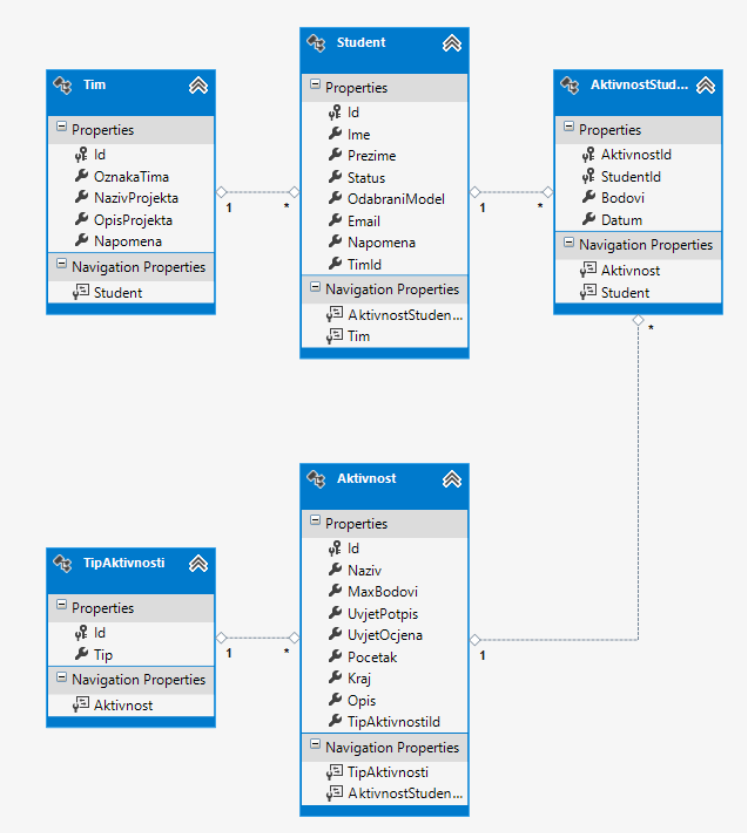
3. Nakon toga nas Visual Studio pita što novokreirani model treba sadržavati. Imamo dvije ponuđene opcije: generiranje iz postojeće baze ili prazan model. S

obzirom da mi imamo gotovu bazu generirati ćemo model iz nje, i to tako da odaberemo opciju **Generate from database**.

4. Otvara nam se prozor u kojem trebamo odabrati konekciju na bazu podataka. Odaberite **New Connection**.
5. U prozoru za novu konekciju postavimo vrijednost **Data Source-a** na **Microsoft SQL Server Compact** (gumb **Change**).
6. Pod **Database** pronađite bazu pod nazivom *EvidencijaStudenata.sdf* (gumb **Browse**).
7. Testirajte konekciju na bazu sa opcijom **Test Connection**, i ako je sve uredu kliknite **OK**.
8. Kliknite **Next** kako bi spremili **ConnectionString**.
9. Stavite kvačicu na opciju Tables kako bi uključili sve tablice u model. Također opcija **Include foreign key columns in model** treba biti uključena. Kliknite **Finish**.

3.2. Konceptualni model

Visual Studio nam je sada generirao entitetni model na temelju baze podataka. Konceptualni model izgleda ovako:



Slika 4 Konceptualni model

Možemo primijetiti da je prikazani konceptualni model zapravo jako sličan ERA modelu naše baze. Ipak, u određenim slučajevima razlike će postojati. S obzirom da konceptualni model prikazuje generirane klase, tu možemo iskoristiti koncepte objektno-orijentiranog programiranja, kao što je nasljeđivanje. U bazi podataka nije moguće definirati nasljeđivanje između tablica. Osim toga, u bazi podataka nužno moramo razbiti vezu više-više, dok u konceptualnom modelu ne.

Ako pogledamo svojstva klasa u konceptualnom modelu, možemo vidjeti da postoje dvije vrste: **skalarna svojstva** i **navigacijska svojstva**. Skalarna svojstva preslikavaju uobičajene atribute tablice iz baze, kao što su atributi tipa integer, float, char. S druge strane, navigacijska svojstva preslikavaju veze između entiteta.

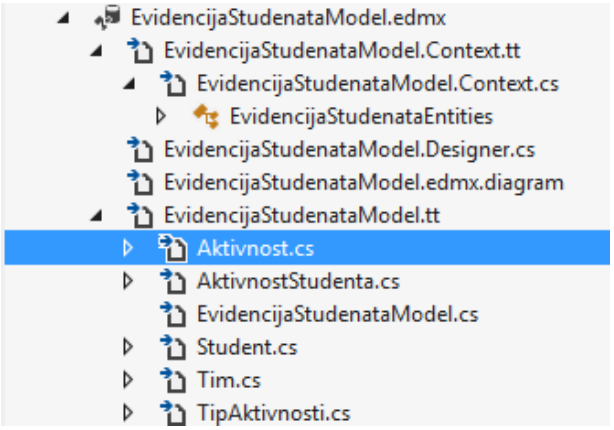
Da bi razumjeli navigacijska svojstva moramo promotriti tipove veza između klasa na konceptualnom modelu. Postoje 3 vrste veza:

1. Veza 1:1 – implementira se na način da svaka klasa sadrži navigacijsko svojstvo koje zapravo predstavlja referencu na objekt klase s kojom je povezana.
2. Veza 1:N – implementira se na način da klasa na strani 1 sadrži navigacijsko svojstvo koje predstavlja kolekciju objekata klase s kojom je povezana, a klasa na strani N sadrži referencu na jedan objekt klase s kojom je povezana.
3. Veza N:M – implementira se na način da obje klase sadrže navigacijsko svojstvo kolekciju objekata one druge klase.

Konceptualni model na slici 4. sadrži samo veze tipa 1:N. Npr. klasa Tim je povezana vezom 1:N sa klasom Student. To znači da klasa Tim sadrži navigacijsko svojstvo Student koje je kolekcija objekata tipa Student. S druge strane klasa Student sadrži navigacijsko svojstvo Tim, koje predstavlja referencu na objekt tipa Tim.

3.3. Entitetne klase i kontekst

Klase koje možemo vidjeti na konceptualnom modelu je Visual Studio i generirao (Aktivnost, AktivnostStudenata, Student, Model). Možemo ih pronaći u **Solution Exploreru** u okviru **edmx** skupine datoteka. Osim njih na tom mjestu možemo pronaći i klasu konteksta *EvidencijaStudenataEntities*.



Slika 5 Datoteke EDM-a

Na slici ispod možete vidjeti primjer generirane klase *Student*. S obzirom da je klasa generirana na temelju tablice iz baze

podataka, u njoj su generirana skalarna svojstva koja preslikavaju strukturu tablice, te navigacijska svojstva koja implementiraju vezu sa drugim tablicama. Osim konstruktora nikakve druge metode i logika nisu generirane. Ovakve klase, koje sadrže samo atribute i svojstva, se često nazivaju *entitetnim klasama*.

Ipak primijetite da je klasa definirana kao parcijalna klasa (eng. partial), što ostavlja mogućnost definiranje logike klase, ali i ostalih dodatnih elemenata klase u drugoj datoteci. To je ujedno i jedna od velikih prednosti ORM pristupa u odnosu na npr. nepovezani način rada sa bazom podataka (DataSet).

```
public partial class Student
{
    public Student()
    {
        this.AktivnostStudenta = new HashSet<AktivnostStudenta>();
    }

    public int Id { get; set; }
    public string Ime { get; set; }
    public string Prezime { get; set; }
    public string Status { get; set; }
    public string OdabraniModel { get; set; }
    public string Email { get; set; }
    public string Napomena { get; set; }
    public Nullable<int> TimId { get; set; }

    public virtual ICollection<AktivnostStudenta> AktivnostStudenta { get; set; }
    public virtual Tim Tim { get; set; }
}
```

Slika 6 Generirana klasa Student

Jedna od najbitnijih generiranih klasa je i klasa *EvidencijaStudenataEntities* koju nazivamo kontekst. Kontekstna klasa nasljeđuje klasu **DbContext**, i predstavlja primarni način za dohvaćanje podataka iz baze (u obliku objekata), te spremanje napravljenih promjena u bazu. Kao što je vidljivo iz slike ispod, kontekstna klasa sadrži DbSet kolekcije objekata za svaku tablicu iz baze podataka, preko kojih zapravo radimo sa podacima. Ako želimo npr. dohvatiti jednog ili više studenata, izmijeniti, obrisati ili dodati studenta, tada to nećemo raditi izravno nad tablicom Studenti u bazi podataka, nego sa DbSet kolekcijom Studenti. Detalje komunikacije sa bazom ostavljamo Entity Framework-u. Također, kroz konstruktor klase je proslijeđen naziv konekcije na bazu na koju se kontekst spaja (podaci o konekciju su dostupni kroz App.config datoteku u Solution Exploreru).

```
public partial class EvidencijaStudenataEntities : DbContext
{
    public EvidencijaStudenataEntities()
        : base("name=EvidencijaStudenataEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public DbSet<Aktivnost> Aktivnost { get; set; }
    public DbSet<AktivnostStudenta> AktivnostStudenta { get; set; }
    public DbSet<Student> Student { get; set; }
    public DbSet<Tim> Tim { get; set; }
    public DbSet<TipAktivnosti> TipAktivnosti { get; set; }
}
```

Slika 7 Kontekstna klasa

3.4. Čitanje podataka uz pomoć EF-a

S obzirom da već imamo definirano grafičko sučelje, i da smo se uz pomoć Entity Framework-a spojili na bazu podataka, možemo pročitati podatke i prikazati ih na odgovarajućim formama. Započeti ćemo sa prikazom studenata po timovima na formi FrmPopisStudenata, i to kroz dvije DataGridView kontrole.

Da bi DataGridView kontrole znale kakvi će podaci biti smješteni u njih postaviti ćemo im izvor podataka (eng. Data Source) na sljedeći način:

1. Selektirajmo DataGridView *dgvTimovi*, te kliknimo na trokutić koji se pojavio u gornjem desnom kutu kontrole.
2. U izborniku otvorimo opciju *Chose Data Source* te odaberimo opciju *Add Project Data Source*.
3. Odaberimo Object kao izvor podataka i kliknimo *Next*.
4. Pronađite klasu Tim, označite je i kliknite *Finish*.

Primijetite kako je Visual Studio automatski generirao BindingSource objekt te kako je svako svojstvo klase Tim sada prikazano u obliku stupca u DataGridView kontroli. Ipak, ne trebaju nam svi stupci. Naime, klasa tim sadrži navigacijsko svojstvo Studenti, koje zapravo predstavlja kolekciju studenata koji se nalaze u timu. DataGridView kontrola ne zna kako da u jednoj ćeliji prikaže kolekciju objekata, te bi u ovom slučaju prilikom prikaza DataGridView-a dogodila iznimka. Zbog toga ćemo stupac Studenti sakriti preko opcije *Edit Columns* (svojstvo Visible = False).

Na isti način potrebno je definirati izvor podataka za DataGridView kontrolu za prikaz studenata (dgvStudenti), te sakriti stupce *TimId, AktivnostiStudenta, Tim*.

Kada smo definirali strukturu prikaza u DataGridView kontrolama možemo izraditi metode koje će zapravo ažurirati prikaz timova i studenata. Metode ćemo dodati u formi FrmPopisStudenata, te ćemo ih za početak pozvati u Load događaju forme. Također kako bi se studenti prikazivali ovisno o selektiranom timu, metodu PrikaziStudente ćemo pozvati i prilikom promjene selektiranog retka u *dgvTimovi*.

Metode za prikaz studenata po timovima

```
/// <summary>
/// Dohvaća listu svih timova iz kolekcije timova u kontekstu, te ih prikazuje
/// u DataGridView-u.
/// </summary>
private void PrikaziTimove()
{
     BindingList<Tim> listaTimova = null;
     using (var db = new EvidencijaStudenataEntities())
     {
          listaTimova = new BindingList<Tim>(db.Tim.ToList());
     }
     timBindingSource.DataSource = listaTimova;

}

/// <summary>
/// Dohvaća listu studenata proslijeđenog tima te ih prikazuje u DataGridView-u.
/// </summary>
/// <param name="tim">Tim čije studente želimo prikazati.</param>
private void PrikaziStudente(Tim tim)
{
     BindingList<Student> listaStudenata = null;
     using (var db = new EvidencijaStudenataEntities())
     {
          db.Tim.Attach(tim);
          listaStudenata = new
          BindingList<Student>(tim.Student.ToList<Student>());
     }
            studentBindingSource.DataSource = listaStudenata;
}

/// <summary>
/// Rukuje događajem pokretanja forme.
/// </summary>
private void FrmPopisStudenata_Load(object sender, EventArgs e)
{
     PrikaziTimove();
     PrikaziStudente(timBindingSource.Current as Tim);
}

/// <summary>
/// Rukuje događajem promjene selektiranog retka u tablici.
/// </summary>
private void dgvTimovi_SelectionChanged(object sender, EventArgs e)
{
     Tim selektiraniTim = timBindingSource.Current as Tim;
```

```
      if (selektiraniTim != null)
      {
            PrikaziStudente(selektiraniTim);
      }
}
```

Primijetite kako za pristup podacima koristimo instancu kontekstne klase (*EvidencijaStudenataEntities*), koju smo kreirali unutar **using** bloka kako ne bi morali brinuti o oslobađanju resursa. Za dohvaćanje svih timova jednostavno koristimo DbSet kolekciju Timovi koja se nalazi unutar konteksta. S druge strane s obzirom da želimo prikazati samo studente trenutno označenog tima, kolekciju tih studenata možemo dobiti preko navigacijskog svojstva Studenti koje sadrži klasa Tim. Naime, navigacijsko svojstvo Studenti klase Tim zapravo predstavlja kolekciju studenata koji se nalaze u tom timu.

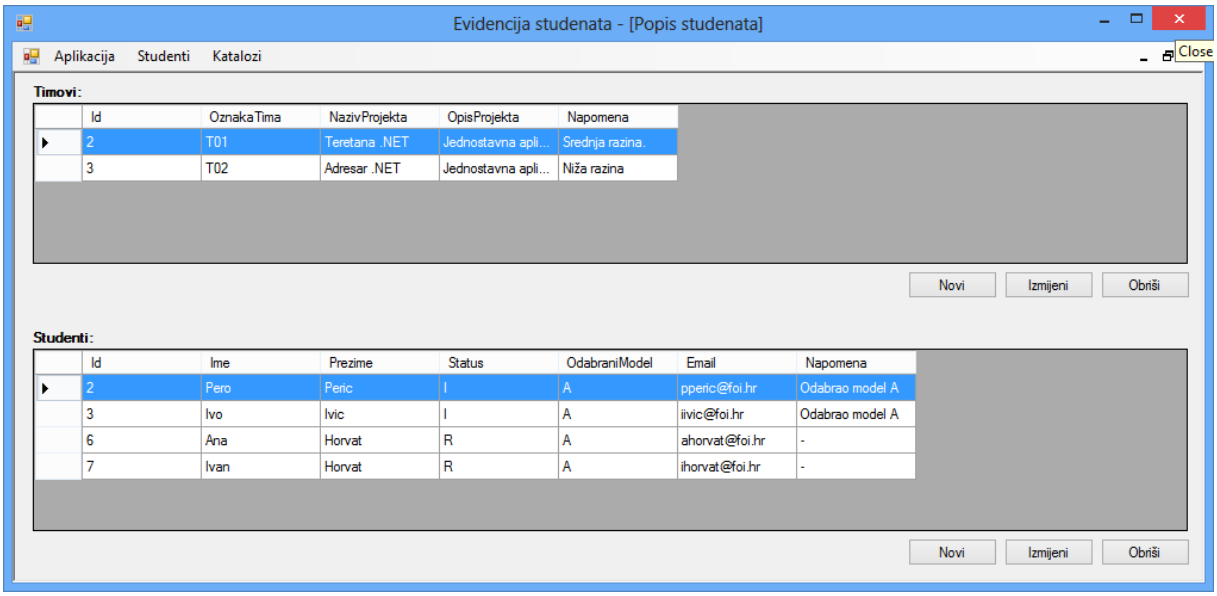
Povezani (eng. Connected) i nepovezani (eng. Disconnected) scenarij

Jedna od stvari na koju treba obratiti pažnju prilikom rada sa Entity Framework-om je postojanje dva scenarija rada sa entitetnim objektima. Povezani način rada podrazumijeva da entitetne objekte koristimo unutar konteksta u kojem smo ih i dohvatili. S druge strane, nepovezani scenarij podrazumijeva da unutar jednog konteksta koristimo entitetni objekt koji smo kreirali u nekom drugom kontekstu i proslijedili trenutnom kontekstu. U nepovezanom scenariju kontekst ne može prepoznati proslijeđeni objekt, te ga je potrebno eksplicitno „registrirati" uz pomož naredbe Attach. Primjer povezanog i nepovezang scenarija su sljedeći:

```
private void PovezaniScenarij()
{
      using (var db = new EvidencijaStudenataEntities4())
      {
            //Tim dohvaćamo preko trenutnog konteksta
            Tim prviTim = (from t in db.Timovi
                           where t.Id == 1
                           select t).First<Tim>();
            BindingList<Student> lista = new
            BindingList<Student>(prviTim.Studenti.ToList<Student>());
            studentBindingSource.DataSource = lista;
      }
}

private void NepovezaniScenarij(Tim prviTim)
{
      using (var db = new EvidencijaStudenataEntities4())
      {
      //Proslijedili smo Tim trenutnom kontekstu i zato ga je potrebno "registrirati".
            db.Timovi.Attach(prviTim);
            BindingList<Student> lista = new
            BindingList<Student>(prviTim.Studenti.ToList<Student>());
            studentBindingSource.DataSource = lista;
      }
}
```

Ako sada pokrenemo projekt, i otvorimo formu sa popisom studenata, rezultat bi trebao biti otprilike sljedeći:



Slika 8 Popis studenata po timovima

3.5. Kreiranje novog entitetnog objekta

Na primjeru dodavanja novog tima i novog studenta ćemo pokazati kako se dodaju novi entitetni objekti u kontekst, te kako se iz konteksta spremaju u bazu podataka. Na formi FrmNoviTim moguće je upisati podatke za novi tim, te klikom na gumb *Uredu* kreirati novi objekt i spremiti ga u bazu.

Kreiranje novog tima

```
/// <summary>
/// Rukuje događajem klika na gumb Uredu.
/// </summary>
private void btnUredu_Click(object sender, EventArgs e)
{
     using(var db = new EvidencijaStudenataEntities())
     {
          //Kreiramo novi objekt klase Tim i popunjavamo ga podacima sa forme.
          Tim tim = new Tim
          {
               OznakaTima = txtOznakaTima.Text,
               NazivProjekta = txtNazivProjekta.Text,
               OpisProjekta = txtOpisProjekta.Text,
               Napomena = txtNapomena.Text
          };
          db.Tim.Add(tim);  //Dodajemo tim u odgovarajuću kolekciju u kontekstu.
          db.SaveChanges();    //Spremamo napravljene promjene u bazu podataka.
     }
     Close();
}
```

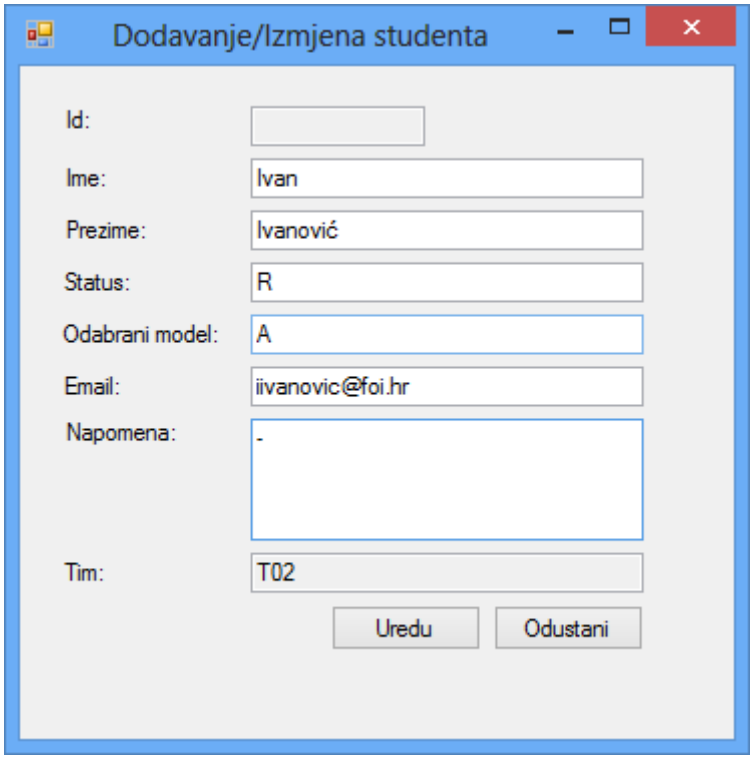
Obratite pažnju na dodavanje novog objekta u odgovarajuću kolekciju u kontekstu, te pozivanje metode konteksta **SaveChanges**, koja sve promjene koje su napravljene u kontekstu sprema u bazu podataka.

Da bi novokreirani tim bio prikazan u tablici timovi na formi FrmPopisStudenata potrebno je osvježiti prikaz ponovnim pozivanjem metode *PrikaziTimove*, i to u metodi *btnNoviTim_Click* forme FrmPopisStudenata:

Ažuriranje popisa timova nakon dodavanja novog tima

```
/// <summary>
/// Rukuje događajem klika na gumb za dodavanje novog tima.
/// </summary>
private void btnNoviTim_Click(object sender, EventArgs e)
{
      FrmNoviTim forma = new FrmNoviTim();
      forma.ShowDialog();
      PrikaziTimove();
}
```

Na sličan način možemo napraviti i dodavanje novog studenta na formi FrmNoviStudent. Klikom na gumb *Uredu* kreirati će se novi objekt klase Student, popuniti podacima sa forme, dodati u kolekciju Studenti u kontekst, te na kraju spremiti u bazu podataka. Da bi znali kojem timu pripada student, formi FrmNoviStudent ćemo proslijediti selektirani tim sa forme FrmPopisStudenata.



Slika 9 Dodavanje novog studenta

Kreiranje novog studenta

```
private Tim selektiraniTim;   //Atribut u koji ćemo pohraniti selektirani tim.

public FrmNoviStudent(Tim tim)  //Formi prosljeđujemo selektirani tim.
{
     InitializeComponent();
     selektiraniTim = tim;
}

private void btnUredu_Click(object sender, EventArgs e)
{
     using (var db = new EvidencijaStudenataEntities())
     {
          //S obzirom da objekt selektiraniTim nije kreiran u
          //tekućem kontekstu, moramo ga "registrirati".
          db.Tim.Attach(selektiraniTim);
          Student student = new Student       //Kreiramo novog studenta
          {
               Ime = txtIme.Text,
               Prezime = txtPrezime.Text,
               Status = txtStatus.Text,
               OdabraniModel = txtOdabraniModel.Text,
               Napomena = txtNapomena.Text,
               Email = txtEmail.Text,
               Tim = selektiraniTim
           };

           db.Student.Add(student);  //Dodajemo studenta u kolekciju.
           db.SaveChanges();          //Spremamo promjene u bazu podataka.
     }
     Close();
}
```

Da bi novokreirani student bio prikazan u popisu studenata potrebno je ažurirati tablicu studenti na formi FrmPopisStudenata. Također, prilikom pozivanja forme za dodavanje novog studenta je potrebno proslijediti selektirani tim. To sve ćemo napraviti u metodi btnNoviStudent_Click forme FrmPopisStudenata.

Ažuriranje popisa studenata nakon dodavanja novog studenta

```
/// <summary>
/// Rukuje događajem klika na gumb za dodavanje novog studenta.
/// </summary>
private void btnNoviStudent_Click(object sender, EventArgs e)
{
     FrmNoviStudent forma = new FrmNoviStudent(timBindingSource.Current as Tim);
     forma.ShowDialog();
     PrikaziStudente(timBindingSource.Current as Tim);
}
```

3.6. Izmjena postojećeg entitetnog objekta

Da bi pokazali način izmjene postojećeg entitetnog objekta u Entity Framework-u napraviti ćemo mogućnost izmjene postojećeg. Kada selektiramo tim iz popisa timova na formi FrmPopisStudenata i kliknemo na gumb izmijeni, otvoriti će nam se forma za izmjenu podataka o timu. Ovdje ćemo koristiti istu formu kao i za dodavanje timova (FrmNoviTim), ali dodatno prilagođenu za funkcionalnost izmjene tima.

Da bi forma FrmNoviTim znala koji tim želimo izmjeniti, potrebno je proslijediti tim selektiran na formi FrmPopisStudenata. Za to će nam biti potreban još jedan konstruktor i jedan atribut za formu FrmNoviTim. Kada se pokrene forma za izmjenu tima želimo popuniti TextBox kontrole na formi sa podacima prosljeđenog tima, pa ćemo doraditi metodu Load. Na kraju, klikom na gumb Uredu korisnik želi spremiti promjene na postojećem timu, te ćemo doraditi i metodu *btnUredu_Click*.

Dorada forme FrmNoviTim za funkcionalnost izmjene podataka o timu

```
private Tim timZaIzmjenu;

/// <summary>
/// Konstruktor forme koji koristimo za izmjenu postojećeg tima.
/// </summary>
/// <param name="tim">Tim koji treba izmijeniti.</param>
public FrmNoviTim(Tim tim)
{
     InitializeComponent();
     timZaIzmjenu = tim;
}

/// <summary>
/// Rukuje događajem Load pokretanja forme.
/// </summary>
private void FrmNoviTim_Load(object sender, EventArgs e)
{
     txtOznakaTima.Focus();
     if (timZaIzmjenu != null)
     {
          txtId.Text = timZaIzmjenu.Id.ToString();
          txtOznakaTima.Text = timZaIzmjenu.OznakaTima;
          txtNazivProjekta.Text = timZaIzmjenu.NazivProjekta;
          txtOpisProjekta.Text = timZaIzmjenu.OpisProjekta;
          txtNapomena.Text = timZaIzmjenu.Napomena;
      }
}

/// <summary>
/// Rukuje događajem klika na gumb Uredu.
/// </summary>
private void btnUredu_Click(object sender, EventArgs e)
{
     using (var db = new EvidencijaStudenataEntities())
     {
          if (timZaIzmjenu == null)
          {
```

```
                //Kreiramo novi objekt klase Tim i popunjavamo ga
                //podacima sa forme.
                Tim tim = new Tim
                {
                     OznakaTima = txtOznakaTima.Text,
                     NazivProjekta = txtNazivProjekta.Text,
                     OpisProjekta = txtOpisProjekta.Text,
                     Napomena = txtNapomena.Text
                };
                db.Tim.Add(tim);  //Dodajemo tim u odgovarajuću kolekciju
                db.SaveChanges();   //Spremamo napravljene promjene u bazu podataka.
           }
           else   //Mijenjamo postojeći tim
           {
                db.Tim.Attach(timZaIzmjenu);     //registriramo prosljeđeni tim.

                timZaIzmjenu.OznakaTima = txtOznakaTima.Text;
                timZaIzmjenu.NazivProjekta = txtNazivProjekta.Text;
                timZaIzmjenu.OpisProjekta = txtOpisProjekta.Text;
                timZaIzmjenu.Napomena = txtNapomena.Text;

                db.SaveChanges();   //Spremamo promjene u bazu.
           }
      }
      Close();
}
```

Osim dorade forme FrmNoviTim potrebno je dodati metodu za rukovanje klikom na gumb Izmijeni na formi FrmPopisStudenata. Unutar te metode ćemo dohvatiti selektirani tim, te pozvati formu za izmjenu tog tima.

Pozivanje forme za izmjenu tima

```
/// <summary>
/// Rukuje događajem klika na gumb Izmijeni.
/// </summary>
private void btnIzmijeniTim_Click(object sender, EventArgs e)
{
     Tim selektiraniTim = timBindingSource.Current as Tim;
     if (selektiraniTim != null)
     {
          FrmNoviTim forma = new FrmNoviTim(selektiraniTim);
          forma.ShowDialog();
          PrikaziTimove();
     }
}
```

3.7. Brisanje postojećeg entitetnog objekta

Brisanje postojećeg entitetnog objekta je jednostavna operacija, a svodi se na uklanjanje objekta iz konteksta, odnosno njegove kolekcije u kontekstu, te pozivanje metode *SaveChanges* za spremanje promjena u bazu. To ćemo pokazati na primjeru brisanja tima i studenta.

Da bi korisnik izbrisao tim mora ga selektirati u popisu timova na formi FrmPopisStudenata, te kliknuti na gumb Obriši. S obzirom da tim može sadržavati studente u kolekciji studenata, prije brisanja tima moramo provjeriti da li se u njemu nalaze studenti. Ukoliko tim ima studente obavijestiti ćemo korisnika da nije moguće obrisati tim.

Brisanje tima na formi FrmPopisStudenata

```
/// <summary>
/// Rukuje događajem klika na gumb za brisanje tima.
/// </summary>
private void btnObrisiTim_Click(object sender, EventArgs e)
{
     Tim selektiraniTim = timBindingSource.Current as Tim;
     if (selektiraniTim != null)
     {
          if (MessageBox.Show("Da li ste sigurni?", "Upozorenje!",
              MessageBoxButtons.YesNo) == System.Windows.Forms.DialogResult.Yes)
          {
               using (var db = new EvidencijaStudenataEntities())
               {
                    db.Tim.Attach(selektiraniTim);   //Registriramo tim.
                    //Provjeravamo da li tim sadrži studente.
                    if (selektiraniTim.Student.Count == 0)
                    {
                         db.Tim.Remove(selektiraniTim);   //Brišemo tim..
                         db.SaveChanges();    //Spremamo promjene u bazu.
                    }
                    else
                    {
                         MessageBox.Show("Nije moguće obrisati tim koji sadrži
                                          studente!");
                    }
               }
               PrikaziTimove();    //Ažuriramo popis timova.
          }
     }
}
```

4. Samostalni zadatak

1. Napravite mogućnost izmjene podataka za studenta, te mogućnost brisanja studenta.
2. Napravite ostale opcije aplikacije koje su definirane u korisničkim zahtjevima.

5. Pitanja i zadaci

1. Objasnite koncept ORM-a. Koje su njegove prednosti?
2. Što je to entitetni model podataka (eng. Entity Data Model – EDM), od čega se on sastoji?
3. Objasnite konceptualni model u EF-u. Od čega se sastoji, i gdje ga u projektu možemo naći?
4. Objasnite model pohrane u EF-u. Od čega se sastoji, i gdje ga u projektu možemo naći?
5. Objasnite model preslikavanja. Od čega se sastoji, i gdje ga u projektu možemo naći?
6. Koja je uloga sloja objektnih usluga u EF-u?
7. Koja je uloga sloja entitetnog klijenta u EF-u?
8. Objasnite ulogu ADO.NET tehnologije u EF-u.
9. Koji je glavni upitni jezik u Entity Framework-u?
10. Koji tokovi rada (eng. Workflows) postoje u Entity Framework-u, i kada ih koristimo?
11. Objasnite razliku između Top-Down i Bottom-Up pristupa.
12. Koje vrste svojstava sadrži klasa u Entity Framework-u?
13. Što je to kontekstna klasa i za što služi?
14. Objasnite povezani i nepovezani scenarij rada sa kontekstnom klasom.

6. Više informacija o temi

1. MSDN Get started with Entity Framework (EF)
   http://msdn.microsoft.com/en-us/data/ee712907
2. MSDN Data Development Technical Articles
   http://msdn.microsoft.com/en-us/data/aa937699
3. Entity Framework Tutorial
   http://entityframeworktutorial.net/
4. Entity Framework in WinForms
   http://www.codeproject.com/Articles/221931/Entity-Framework-in-WinForms
5. Youtube Entity Framework tutorial
   Part 1: http://www.youtube.com/watch?v=BS6IKdUd2V8
   Part 2: http://www.youtube.com/watch?v=5RgL5O28B58
   Part 3: http://www.youtube.com/watch?v=iQqzTr-1waE
   Part 4: http://www.youtube.com/watch?v=OTasIwcXNyE
   Part 5: http://www.youtube.com/watch?v=tAJCwY-sClc
   Part 6: http://www.youtube.com/watch?v=tAJCwY-sClc

