Objektno oblikovanje

Rad za laboratorij profila

|  |  |
| --- | --- |
| ČLANOVI TIMA | „Lovro i prijatelji“ |
| ČLANOVI GRUPE | KLARA BANIĆ, LOVRO RAČKI, LEON SULJIĆ |
| SAžETAK SEMINARA | APLIKACIJA ZA TRAŽENJE RECEPATA |
| GIT REPOZITORIJ | <HTTPS://GITHUB.COM/RACKILOVRO/SEMINAROBJEKTNO> |

sadržaj

[Opis domene 3](#_Toc63003546)

[NHibernate vs. Entity Framework 4](#_Toc63003547)

[Usporedba razvojnih okvira 4](#_Toc63003548)

[Podržane platforme 4](#_Toc63003549)

[Arhitektura 4](#_Toc63003550)

[Podržane baze podataka 4](#_Toc63003551)

[Konfiguracije i mapiranja 4](#_Toc63003552)

[Nasljeđivanje tablica 5](#_Toc63003553)

[Generiranje primarnih ključeva 5](#_Toc63003554)

[Upiti i izmjene nad podacima 5](#_Toc63003555)

[Transakcije 5](#_Toc63003556)

[Kolekcije 5](#_Toc63003557)

[O/R mapiranje 8](#_Toc63003558)

[Uvod 8](#_Toc63003559)

[Tijek objektno relacijskog mapiranja 8](#_Toc63003560)

[Database-first pristup 8](#_Toc63003561)

[Prvi problemi 8](#_Toc63003562)

[Primjer mapiranja 12](#_Toc63003563)

# Opis domene

Model domene je sličan rasporedu relacija u bazi podataka za aplikaciju. Ovo je posljedica korištenja *Entity Frameworka* u izradi O/R sloja.

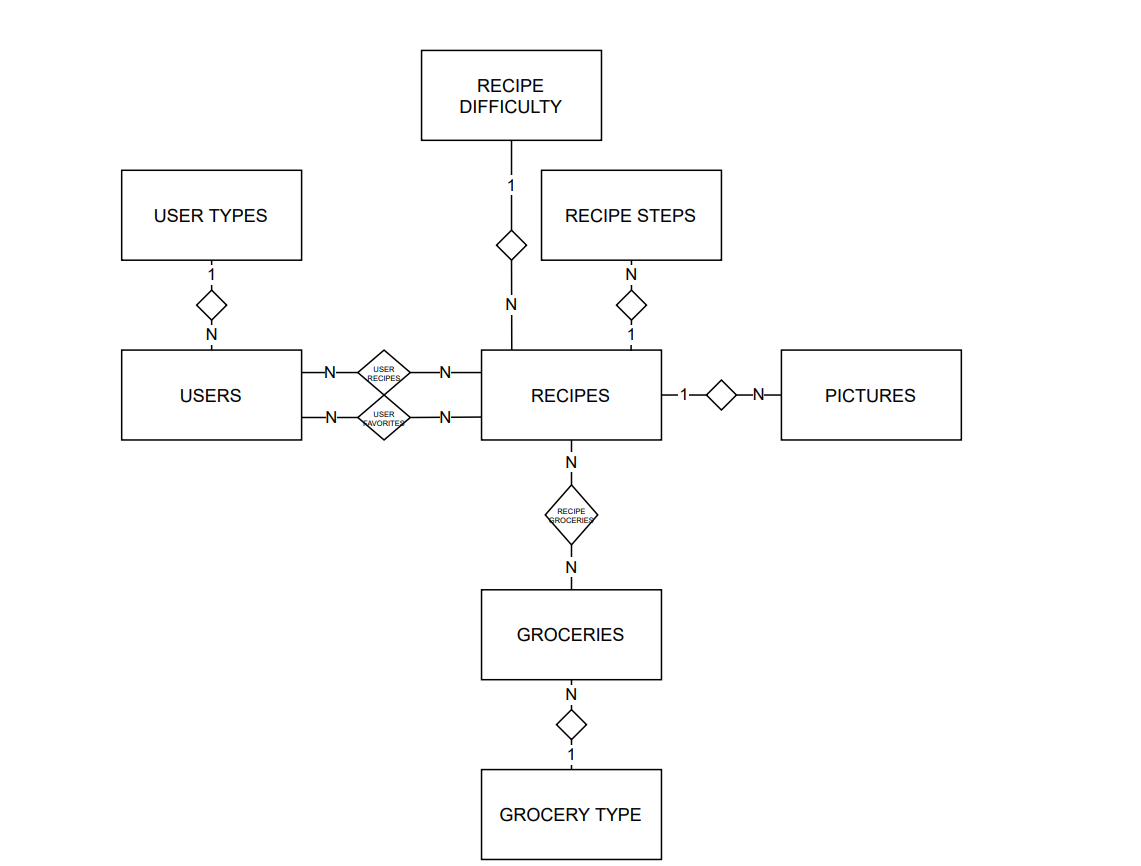


Figure 1 Relacijski dijagram modela domene

Model je prikazan dijagramom klasa i očigledno predstavlja *Model* komponentu u *MVC* i *MVVM* arhitekturnim obrascima. Najvažnije klase su *Recipe* i *User*preko kojih se odrađuju skoro sve funkcionalnosti na aplikacijskom sloju.

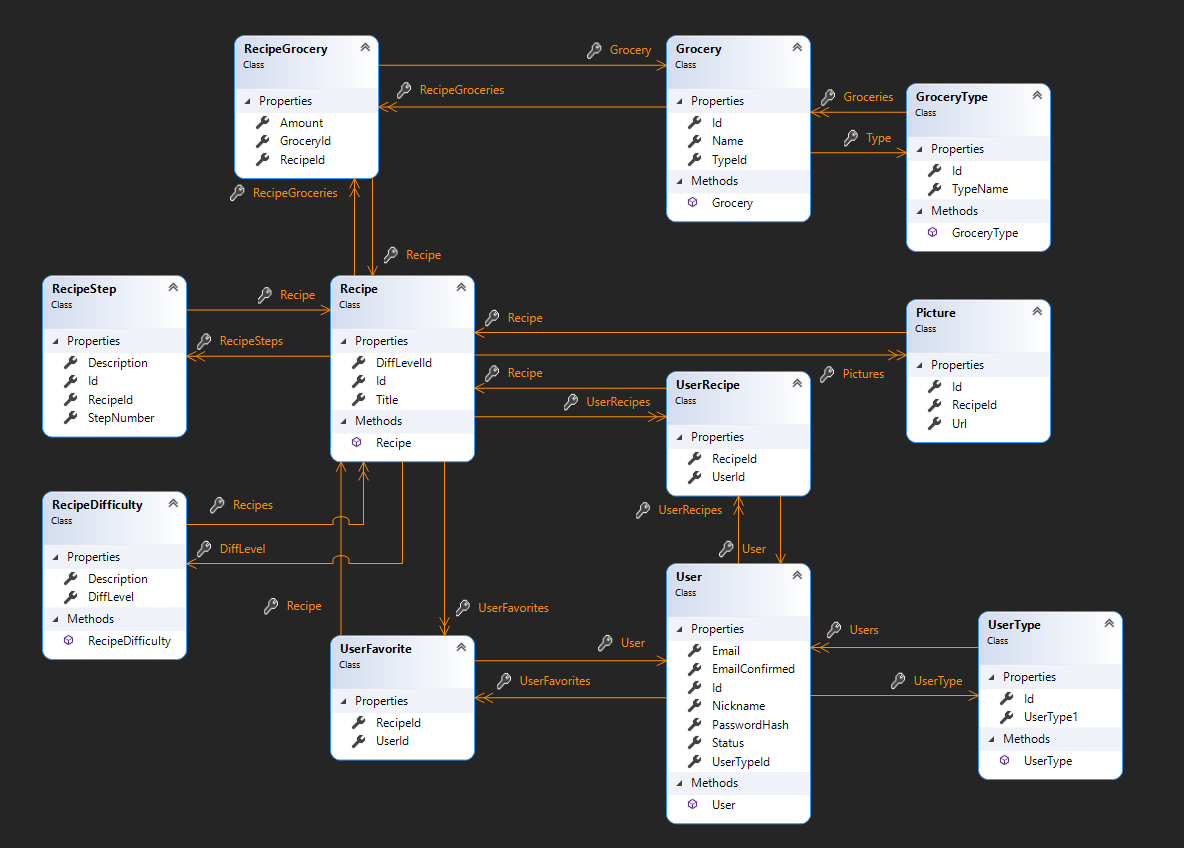


Figure 2 Dijagram klasa

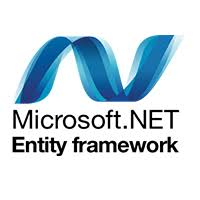
Svaka klasa sadrži potrebna polja koja su mogu lako preslikati u n-torku koja se unosi u prikladnu relaciju u bazi podataka. Uz polja*, Entity Framework* je dodao referencu na klasu koja sadrži strani ključ početne klase. Ovime je omogućeno jednostavno dohvaćanje podataka u složenim operacijama gdje je potrebno spajati instance klasa po određenim ključevima.

Može se primijetiti, iz modela, kako je realizirana jedna od najvažnijih funkcionalnosti aplikacije. Naime, klasa Recept sadrži kolekcije instanci klasa *RecipeStep*, *RecipeGrocery* i reference na instance klasa *Picture*, *RecipeDifficulty* i *UserRecipe*. Povezivanje instanci ovakvih povezanih klasa je uz ovakav model trivijalno i *Entity Framework* nakon spremanja promjena u kontekstu povezuje ove objekte na razini baze podataka.

Svakako je prednost ovakvog modela njegova jednostavnost dodavanja i dohvaćanja podataka. Međutim, poslovna logika (koje u ovoj aplikaciji nema puno) je prebačena u zaseban sloj aplikacije. Konkretno, u slučaju *RecepTour*-a, je ona u *Controller* sloju.

# NHibernate vs. Entity Framework

 *NHibernate* je razvojni okvir za objektno relacijsko mapiranje u *Microsoft .NET* aplikacijama. Objektno relacijsko mapiranje odnosi se na prevođenje strukture podataka iz tradicionalnih baza podataka u objektno orijentirani model aplikacijskog koda. *Nhibernate* je besplatan alat otvorenog koda i nastao je iz *Hibernatea*, razvojnog okvira za objektno relacijsko mapiranje u programskom jeziku *Java*.

*Entity Framework* je sličan *NHibernate* razvojnom okviru i isto služi za objektno relacijsko mapiranje, no namijenjen je za uporabu unutar *.Net* razvojnog okvira te je bolje podržan unutar *Microsoftovog* stoga tehnologija.

## Usporedba razvojnih okvira

*NHibernate* jedan je od najstarijih i najcjenjenijih razvojni okvira za objektno relacijsko mapiranje, jako podržan u zajednici koja ga ujedno i održava, dok je *Entity Framework* konkurentan i noviji proizvod kojeg podržava *Microsoft*.  
*NHibernate* nudi mnogo više mogućnosti kod svakog svojstva koje je važno za neki razvojni okvir za objektno relacijsko mapiranje. *NHibernate* podupire više baza podataka, više mogućnosti objektno relacijskih mapiranja, više načina generiranja primarnih ključeva itd.  
Iako *NHibernate* nudi daleko više mogućnosti, *Entity Framework* je jako lako za uključiti u projekt i još lakše za koristiti te je bolje integriran u *Microsoftov* stog tehnologija.

### Podržane platforme

*NHibernate* podržava *.Net* 4, i više verzije, te *.Net Core*. Trenutno nema planova za podržavanje izvora podataka koji nisu relacijski, npr. danas sve popularniji *NoSql* sustavi baza podataka.

*Entity Framework* podržava .*Net Core* i svaku platformu koja ga podržava. Budući da je razvijen na *provider* modelu, u teoriji podržava bilo koji izvor podataka.

### Arhitektura

*NHibernate* se uključuje u projekt kroz *Nuget* kao jedan *DLL*, bez dodatnih ovisnosti.

*Entity Framework* raspodijeljen je kroz nekoliko različitih *DLLa,* što može dovesti do problema, kao što je objašnjeno u nastavku.

Za razliku od *NHibernatea*, *Entity Framework* sva potreba mapiranja, informacije o postavkama i *APIje,* stvara, pohranjuje i prosljeđuje u *DBContext* konstruktu.

### Podržane baze podataka

*NHibernate* ukratko podržava većinu postojećih relacijskih baza podataka, dok *Entity Framework* podržava samo *SQL Server, SQLite* i *in-memory* bazu podataka. Valja napomenuti da se podrška za *PostgreSQL* može ostvariti kroz određene programske knjižice.

### Konfiguracije i mapiranja

*Entity Framework* koristi *code-based* konfiguraciju ili *attribute-based* mapiranje, a ugrađene konvencije nije moguće mijenjati. *NHibernate* podržava sve navedene konfiguracije mapiranja i podržava korisničko definirane konvencije.

### Nasljeđivanje tablica

*NHibernate* podržava tri glavna obrasca nasljeđivanja tablica:

* *Table Per Class Hierarchy / Single Table Inheritance*
* *Table Per Class / Class Table Inheritance*
* *Table Per Concrete Class / Concrete Table Inheritance*

*Entity Framework* podržava samo *Table Per Class Hierarchy / Single Table Inheritance.* Postoje zahtjevi za uvođenjem ostalih tipova nasljeđivanja, no nema najave kada bi mogli biti uvedeni.

### Generiranje primarnih ključeva

*Entity Framework* nudi samo tri načina generiranja primarnih ključeva, što se pokazalo kao velik nedostatak u našem razvojnom procesu.  
*Nhibernate* podržava vrlo velik broj načina automatskog generiranja ključeva:

* *Identities in SQL Server/Sybase, or auto-increment values in other databases*
* *Sequences in any database that supports them*
* *The database-independent High-Low algorithm*
* *High-Low based on a sequence*
* *A generator based on the current time*
* *Several flavors of GUIDs (Globally Unique Identifiers)*
* *Manually assigned*

### Upiti i izmjene nad podacima

*Entity Framework* podržava samo *LINQ* i *SQL*.  
Puno više mogućnosti nudi *NHibernate*:

* *LINQ*
* *Criteria API (a Query Object pattern implementation)*
* *QueryOver (Criteria with LINQ)*
* *HQL (object-oriented SQL)*
* *SQL*

Osim veće podrške upitnim jezicima, *NHibernate* omogućava korisnicima da pozivaju pohranjene procedure ili da korisnici zaredaju nekoliko upita koji će se zatim izvršiti ili čak ubacivanje podataka u bazu podataka po *batchevima*.

### Transakcije

Oba razvojna okvira podržavaju osnovne funkcionalnosti transakcija, no kada je riječ o naprednijim funkcionalnostima *NHibernate* nudi veće mogućnosti, no takve funkcionalnosti nam nisu bile potrebne u našim jednostavnim *web* i *desktop* aplikacijama.

### Kolekcije

*Entity Framework* podržava samo jednostavne liste što znači da je samo moguće ostvariti *one-to-many* vezu među razredima, dok *NHibernate* podržava puno više tipova kolekcija, što se najbolje vidi kod uspoređivanja implementacije *many-to-many* veze sa ili bez *NHibernatea*.

|  |  |  |
| --- | --- | --- |
|  | **Entity Framework** | **NHibernate** |
| Podržane platforme | .NET Core | .NET 4, .NET Core |
| Arhitektura | Nekoliko DLL | Jedan DLL |
| Podržane baze podataka | SQL Server (including Azure)  SQLite  In-memory | SQL Server (including Azure)  SQL Server CE  Oracle Database (several editions)  Ingres  PostgreSQL  MySQL  DB2  Sybase  Informix  SQLite  Firebird  Any using ODBC (Open Database Connectivity), or OLE (Object Linking and Embedding) DB |
| Konfiguracija i mapiranje | fluent (code-based) configuration i fluent or attribute-based mapiranje | XML i fluent configuration i mapiranja |
| Nasljeđivanje tablica | Table Per Class Hierarchy / Single Table Inheritance | Table Per Class Hierarchy / Single Table Inheritance  Table Per Class / Class Table Inheritance  Table Per Concrete Class / Concrete Table Inheritance |
| Generiranje primarnih ključeva | IDENTITY auto-generated keys on SQL Server and SQLite  Sequences on SQL Server 2012+  Manually assigned values | Identities in SQL Server/Sybase, or auto-increment values in other databases  Sequences in any database that supports them  The database-independent High-Low algorithm  High-Low based on a sequence  A generator based on the current time  Several flavors of GUIDs (Globally Unique Identifiers)  Manually assigned |
| Upiti i izmjene nad podacima | LINQ  SQL | LINQ  Criteria API (a Query Object pattern implementation)  QueryOver (Criteria with LINQ)  HQL (object-oriented SQL)  SQL |
| Transakcije | ROWVERSION/TIMESTAMP | ROWVERSION/TIMESTAMP  ORA\_ROWSCN  TransactionScope |
| kolekcije | lists | Bags of entities: unordered collections with possible duplication  Sets of entities: no duplication, maybe ordered  Lists: collections indexed by a number  Maps: collections indexed by any column or possibly other entity  Arrays of primitive types  Collections of primitive or value types (components in NHibernate) |
| Lazy i explicit loading | Explicit | Lazy i explicit |
| Pretvorbe tipova | - | Any type |
| Filteri | Global at entity level | Global at entity i collection level |
| Interception | DiagnosticSource | Multiple interceptors |
| Caching | First-level | First and second level |
| Validacija | Data Annotations validation | NHibernate Validator |
| Migracije i generiranje baza podataka | migrations API | automatic ili upon request |
| Roadmap | perspektivno | Ne baš perspektivno |

# O/R mapiranje

## Uvod

Za naš projektni rad odlučili smo se za izbor tehnologija koji nije najbolji i najoptimalniji, no s ovim tehnologijama smo najbolje upoznati pa smo ih odlučili koristiti, makar ne surađuju najbolje zajedno.

 Naša baza podataka nalazi se na *Heroku* serveru. *Heroku* je bio polazišna točka koja je uvjetovala izbor ostalih tehnologija koje smo kasnije koristili. Odlučili smo se za *hostanje* baze podataka na serveru, umjesto *in-memory* rješenja kako bismo lakše mogli ostvariti raspodijeljeni razvoj programske podrške.

 Budući da smo bazu odlučili *hostati* na *Heroku* serveru, izbor *SQL* platforme nam je uvelike smanjen, jer *Heroku* najbolje integrira potporu za *PostgreSQL*, za kojeg smo se odlučili, jer je dobro podržan na *Heroku* i jer ga najbolje poznajemo kroz predmete *Baze podataka* i *Napredni modeli i baze podataka*. Ovo se isto pokazalo kao nespretna odluka jer *Entity Framework* službeno ne podržava *PostgreSQL*, nego se potpora za njega ostvaruje kroz određene *Nuget* pakete.

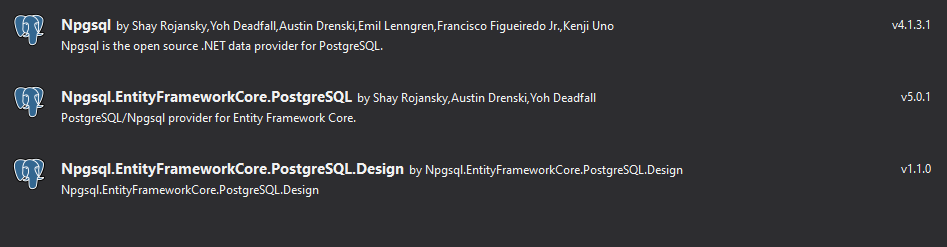
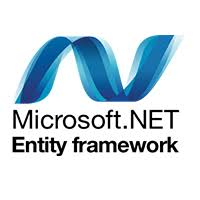


Figure 3 Nuget paketi za rad sa pgsql i Entity Frameworkom

*Npsql* je potreban za usklađivanje *Entity Frameworka* i *postgreSQLa*.

 Naposljetku je u projekt uključen i *Entity Framework* kako bi povezao sve navedene tehnologije i omogućio objektno relacijsko mapiranje.

## Tijek objektno relacijskog mapiranja

Zbog samog izbora tehnologija koje u načelu nisu namijenjene za rad jedno s drugim, najveći problem bio je nesklad *PostgreSQLa i EntityFrameworka*, ne postoji velik izvor informacija na internetu koji bi nas usmjerio u razrješavanju nastalih problema.

### Database-first pristup

Budući da smo kao grupa bolji arhitekti baza podataka, nego inženjeri programskih sustava namijenjenih za *web*, odlučili smo se za *database-first* pristup. Najprije smo definirali model baze podataka te izveli potrebne *SQL* skripte direktno na *Heroku* server. Nakon postavljanja i punjenja baze podataka reprezentativnim podacima, sljedeći korak je povezati *Entity Framework* sa *Heroku* serverom i provesti objektno relacijsko mapiranje.

Prvi problemi

*Entity Framework* jednostavan je za koristiti jer nudi dobru i razvijenu podršku kroz dobro osmišljeno korisničko sučelje…ako koristite neku od službeno podržanih podržanih *SQL* standarda. Nažalost, odlučili smo se za *PostgreSQL* i time si otežali posao. Morali smo se odreći korisničkog sučelja i daljnji razvoj odvijao se kroz terminal.  
Prvi problem na koji smo naišli je *connection string* potreban za povezivanje *Entity Frameworka* i baze podataka na serveru. *Heroku connection stringove* zapisuje u jednom formatu, a *Entity Framework*, odnosno *Npgsql*, očekuje ih u drugom formatu.

Figure 4 Heroku connection string

Figure 5 Heroku connection string

postgres://mcdgxeedppzugi:56fd5710c4601eaa2c022a7a28332a5983e48c97af7cc3cc39b6b8a69da9b9b1@ec2-54-220-229-215.eu-west-1.compute.amazonaws.com:5432/d3jgof5caojkns

Kako bismo riješili taj problem pronašli smo parser koji pretvara *connection string* iz jednog formata u drugi.

Figure 6 Entity Framework connection string

postgres://mcdgxeedppzugi:56fd5710c4601eaa2c022a7a28332a5983e48c97af7cc3cc39b6b8a69da9b9b1@ec2-54-220-229-215.eu-west-1.compute.amazonaws.com:5432/d3jgof5caojkns

Kada smo napokon dobili pravi *connection string* mogli smo dalje nastaviti sa procesom objektno relacijskog mapiranja.



Figure 7 Pokretanje objektno relacijskog mapiranja

Nakon otvaranja *developer command prompta* potrebno je pozicionirati se u *root* direktorij *solutiona* i pokrenuti potrebnu *dbcontext scaffold* naredbu sa navedenim *connection strignom*.  
Ovdje dolazimo do drugog problema na koji smo naišli. Naime, za potrebe *scaffolda* sa baza podataka koje su *hostane* na *Heroku* potrebno je u *connection string* nadodati *SSL Mode* i *TrustServerCertificate* polja. Iako se čini jednostavno na ovaj korak smo izgubili određeno vrijeme, jer ovako specifičan izbor tehnologija nije previše dokumentiran niti popularan.  
Nakon izvršavanja u *Models* direktoriju *solutiona* generirani su svi potrebni razredi modela domene. Prilikom generiranja vidljive su neke negativne posljedice izbora navedenih tehnologija. *Many-to-many* veze nisu predstavljene neko složenom kolekcijom kao što bi bile u *NHibernateu*, nego kao *one-to-many* veze između polazne tablice, spojne tablice i naposljetku završne tablice. Osim toga automatski generirani *dbcontext* razred dobio je nerazumljivi naziv preuzet iz *connection stringa* (d3jgof5caojkns) što dodatno ukazuje na činjenicu da ove tehnologije nisu usklađene, no ovaj problem smo razrješili jednostavnom promjenom naziva razreda.  
Postupak provođenja objektno relacijskog mapiranja proveden je dvaput. Jednom za *desktop* aplikaciju, a jednom za *web* aplikaciju.   
Nakon generiranja modela domene, automatski smo generirali i sve potrebne bazne kontrolere koje smo kasnije nadograđivali po potrebi. U automatski generiranim baznim kontrolerima nalaze GET, POST i DELETE rute za sve potrebne modele.

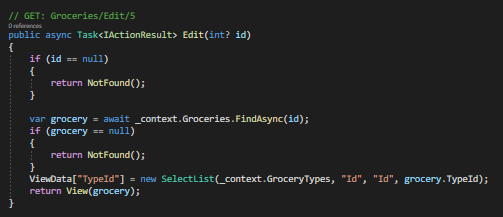


Figure 8 GET u kontroleru za namirnice

Kontroleri funkcioniraju na principu komuniciranja sa objektom *\_context* koji je tipa *db context*. *Db context* razred sadrži reference na sve tablice modela domene te se npr. svi zapisi tablice *Groceries* dobivaju sa *\_context.Groceries.*

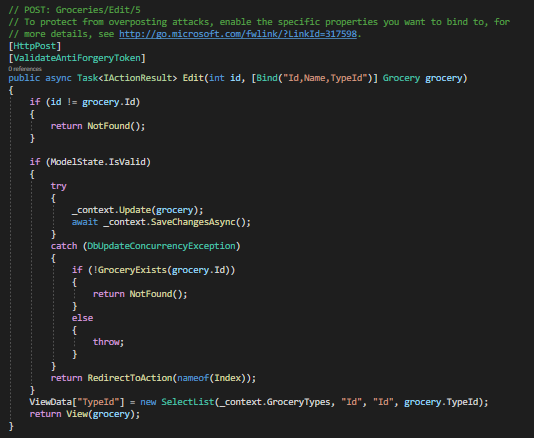


Figure 9 Update za namirnicu

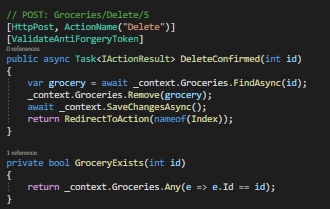


Figure 10 Delete za namirnicu

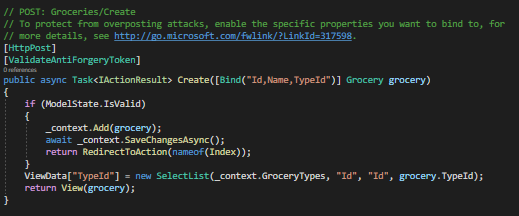


Figure 11 Create metoda za namirnice

Zadnji ozbiljniji problem s kojim smo se suočili vezan je uz generiranje primarnih ključeva.  
Naime, prilikom postavljanja baze na server, u bazu smo direktno ubacili nekoliko podataka za testiranje i razvoj. Kada smo napokon uspjeli pokrenuti *web* aplikaciju krenuli smo testirati automatski generirani *viewove* za pregled, dodavanje, ažuriranje i brisanje podataka. Zbog nekog čudnog razloga nismo mogli smo samo pregledavati podatke. Ispostavilo se da je ključna pogreška bilo dodavanje podataka izravno u bazu podataka, zbog kojeg je došlo do nesklada u komunikaciji prilikom generiranja vrijednosti primarnih ključeva. Brojač primarnih ključeva u bazi bio je na razini većoj od 1, dok je brojač primarnog ključa *Entity Frameworka* ostao na jedan te je uporno pokušavao u bazu podatka dodati zapis sa primarnim ključem 1, što je izazivalo pogreške. Problem je razriješen brisanjem svih zapisa iz baze podataka te smo se u budućnosti ograničili na dodavanje zapisa isključivo kroz *web* aplikaciju. Ovo je samo još jedan primjer nesklada između korištenih tehnologija.

## Primjer mapiranja

Tablica *Groceries* mapirana je u razred *Grocery.*

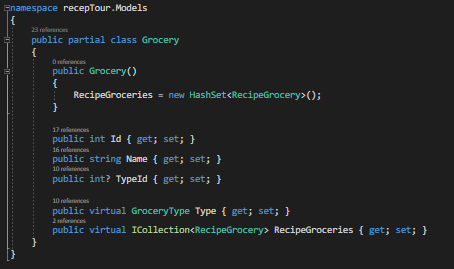
**

Figure 12 Primjer mapiranja tabice

Tablica *Groceries* *many-to-many* vezom povezana je sa tablicom *Recipes*.

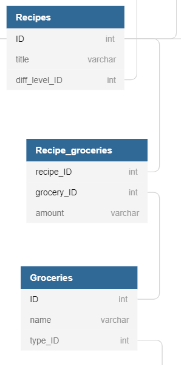


Figure 13 Tablice veze

Navedene tablice su mapirane u tri istoimena razreda gdje je veza između *Groceries* i *Recipe\_groceries* ostvarena pomoću kolekcije *RecipeGroceries*.

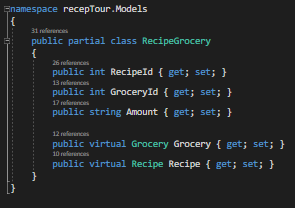


Figure 14 Mapiranje spojne tablice

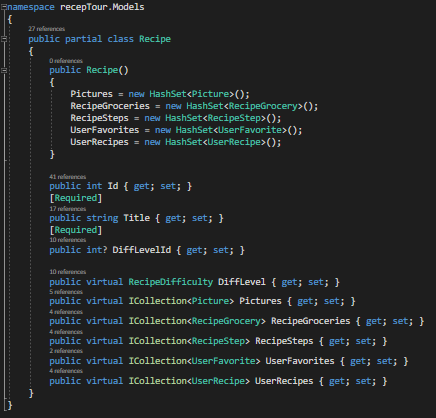


Figure 15 Mapiranje tablice Recipes

[Figure 1 Relacijski dijagram modela domene 3](#_Toc63024517)

[Figure 2 Dijagram klasa 4](#_Toc63024518)

[Figure 3 Nuget paketi za rad sa pgsql i Entity Frameworkom 9](#_Toc63024519)

[Figure 4 Heroku connection string 10](file:///C:\Users\racki\Documents\seminarObjektno\specifikacija\labProfila.docx#_Toc63024520)

[Figure 5 Heroku connection string 10](file:///C:\Users\racki\Documents\seminarObjektno\specifikacija\labProfila.docx#_Toc63024521)

[Figure 6 Entity Framework connection string 10](file:///C:\Users\racki\Documents\seminarObjektno\specifikacija\labProfila.docx#_Toc63024522)

[Figure 7 Pokretanje objektno relacijskog mapiranja 10](#_Toc63024523)

[Figure 8 GET u kontroleru za namirnice 11](#_Toc63024524)

[Figure 9 Update za namirnicu 11](#_Toc63024525)

[Figure 10 Delete za namirnicu 12](#_Toc63024526)

[Figure 11 Create metoda za namirnice 12](#_Toc63024527)

[Figure 12 Primjer mapiranja tabice 13](#_Toc63024528)

[Figure 13 Tablice veze 13](#_Toc63024529)

[Figure 14 Mapiranje spojne tablice 14](#_Toc63024530)

[Figure 15 Mapiranje tablice Recipes 14](#_Toc63024531)