SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Seminar

VetMan

Marko Posavec

Luka Šopar

Zagreb, siječanj, 2019.

Sadržaj

[1. Uvod 3](#_Toc143879)

[1.1. Design story 3](#_Toc143880)

[2. Specifikacija zahtjeva 4](#_Toc143881)

[2.1. Desktop aplikacija 4](#_Toc143882)

[2.1.1. User story 4](#_Toc143883)

[2.1.2. Use cases 5](#_Toc143884)

[2.2. Web aplikacija 5](#_Toc143885)

[2.2.1. User story 5](#_Toc143886)

[2.2.2. Use cases 5](#_Toc143887)

[3. Opis modela 7](#_Toc143888)

[4. Opis implementacije perzistencije 10](#_Toc143889)

[4.1. Opis mapiranja najvažnijih razreda 13](#_Toc143890)

[5. Opis desktop aplikacije 16](#_Toc143891)

[4.1. Isječci iz aplikacije 18](#_Toc143892)

[6. Opis web aplikacije 20](#_Toc143893)

[5.1. Isječci iz aplikacije 22](#_Toc143894)

# Uvod

Sustav koji smo izgradili služi kao alat za veterinare koji ubrzava i olakšava rad te omogućava praćenje istog. Sustav će, osim veterinarima, pomoći i vlasnicima životinja u njihovoj brizi za svoje životinje.

Sustav će biti ostvaren kroz dva sučelja; desktop aplikaciju i web sučelje. Desktop aplikacija će biti namijenjena samim veterinarima. Posao veterinara je pomaganje i liječenje životinja. Mnogo vremena odlazi na papirologiju i birokraciju. Sustav će pokušati pomoći veterinaru da što učinkovitije obavi sve stvari koje je potrebno zabilježiti te omogućiti veterinaru da više vremena posveti pacijentu. Također će nuditi i pomoć u obliku informacija o postojećim lijekovima i bolestima kako bi veterinar bio informiran o svemu.

Web sučelje će biti namijenjeno vlasnicima životinja, koji često nemaju sve informacije ili dovoljno znanje o postupcima koje je njihova životinja prošla. Također će moći pročitati i detalje o bolestima i lijekovima ili postupcima što će im pomoći u daljnjem informiranju.

## Design story

Sustav je zamišljen kao potpora svim veterinarima koji žele nabaviti proizvod, a za održavanje podataka zaduženi su autori aplikacije u suradnji sa stručnjacima.

Veterinar pregledava životinje koje mogu biti raznih vrsta. Životinja kod raznih veterinara može biti različitih vrsta (npr. kod jednog vrste pas, a kod drugog labrador). Vrstu životinja koje pregledava, bira veterinar.

Nad životinjama se obavljaju postupci. Postupke koje obavlja također bira i uređuje veterinar. Lijekovi i bolesti (terapija i dijagnoza) pridružuju se postupcima. Veterinar izdaje račune te naplaćuje postupke i prodane lijekove. Veterinar može prodavati lijekove u raznim pakiranjima (ovisno kakve ima na raspolaganju). Lijekovi mogu interagirati međusobno te je potrebno voditi računa o tome.

Popis bolesti i lijekova koji se nalazi u aplikaciji uređuje agencija zadužena za održavanje proizvoda.

# Specifikacija zahtjeva

Sustav sadrži dva sučelja, koja su namijenjena različitim korisnicima pa će u specifikaciji biti promatrana zasebno.

## Desktop aplikacija

### User story

Program pomaže veterinaru u praćenju svoga rada i stanja svojih pacijenata kroz vrijeme. Veterinar se može prijaviti u sustav.

Veterinar svoje podatke za prijavu dobiva prilikom nabave programskog proizvoda.

Prilikom prvog dolaska bilokojem veterinaru koji koristi sustav, vlasnik životinje će dobiti podatke potrebne za praćenje evidencije svojih ljubimaca, ako ih već nema.

Veterinar ne mora biti stručnjak za sve životinje koje postoje te on sam dodaje vrste životinja kojima je sposoban pružiti odgovarajuću skrb. Životinju prilikom prvog posjeta veterinar registrira u sustav, ako ona već nije registrirana preko web sučelja.

Vlasnik može imati više registriranih životinja.

Životinja može biti različitih vrsta kod različitih veterinara (npr. jedan veterinar je specijaliziran za domaće životinje pa je životinja vrste pas, dok je drugi stručnjak za pse te je životinja vrste labrador).

Nad životinjom veterinar obavlja određeni postupak (npr. pregled ili neki zahvat). Veterinar sam određuje koje usluge postupaka nudi te koja je cijena usluge. Uz postupak može ustanoviti dijagnozu i pripisati terapiju. Terapiju bira s popisa lijekova koji je ponuđen u aplikaciji, a koji uređuje nadležno tijelo za lijekove (u ovom slučaju, autori aplikacije).

Svaki veterinar u svojoj ordinaciji ima lijekove. Lijekove koje ima može registrirati u aplikaciji. Te lijekove može prodavati po cijeni koju odredi.

Svoju dijagnozu nad pacijentom, veterinar može zapisati tijekom postupka. Dijagnoze odabire s popisa poznatih bolesti koje također uređuje nadležno tijelo.

Svakoj bolesti pridruženi su lijekovi koji mogu pomoći pacijentu te se preko lijeka vidi bolesti koje olakšava te obratno, za svaku bolest može se vidjeti koji lijekovi pomažu.

Lijekovi mogu imati neželjene posljedice ako se kombiniraju, tako da je potrebno voditi računa o interakciji između lijekova i upozoriti veterinara tijekom izvođenja postupka.

Veterinar svoje usluge naplaćuje te je dužan izdati račun. Stavke računa mogu biti postupci koje je obavljao te lijekovi koje prodaje. Račun se može preuzeti u PDF obliku.

Veterinar u preko imena vlasnika može pristupiti povijesti postupaka obavljenih nad životinjom koju vlasnik posjeduje.

Veterinar može pregledati popis obavljenih postupaka te generirati i dnevna izvješća obavljenog posla za neki dan te ih preuzeti u PDF obliku.

### Popis use caseova

1. Prijava veterinara u sustav
2. Registracija novih vlasnika
3. Registracija životinja
4. Pretraživanje vlasnika
5. Dodavanje vrsti životinja kojima veterinar pruža uslugu
6. Brisanje vrste životinja
7. Pridruživanje vrste životinja registriranoj životinji
8. Pregled postojećih lijekova
9. Pregled detalja o lijekovima
10. Dodavanje lijekova u lijekove za prodaju
11. Pregled postojećih bolesti
12. Pregled detalja o bolesti
13. Pretraživanje lijekova po imenu
14. Pretraživanje bolesti po imenu
15. Pregled prošlih postupaka
16. Obavljanje postupka
17. Dodavanje postupaka koje veterinar izvodi
18. Brisanje postupaka koje veterinar izvodi
19. Izdavanje računa
20. Preuzimanje PDF izvješća za neki dan
21. Preuzimanje PDF računa

## Web aplikacija

### User story

Web sučelje je, za razliku od desktop sučelja koje je namijenjeno veterinarima, namijenjeno vlasnicima životinja kako bi bili informirani o stanju njihove životinje.

Vlasnici se preko web sučelja mogu registrirati, ako podatke nisu dobili od veterinara. Također mogu i registrirati i svoje životinje.

Vlasnici mogu mijenjati lozinku na web sučelju.

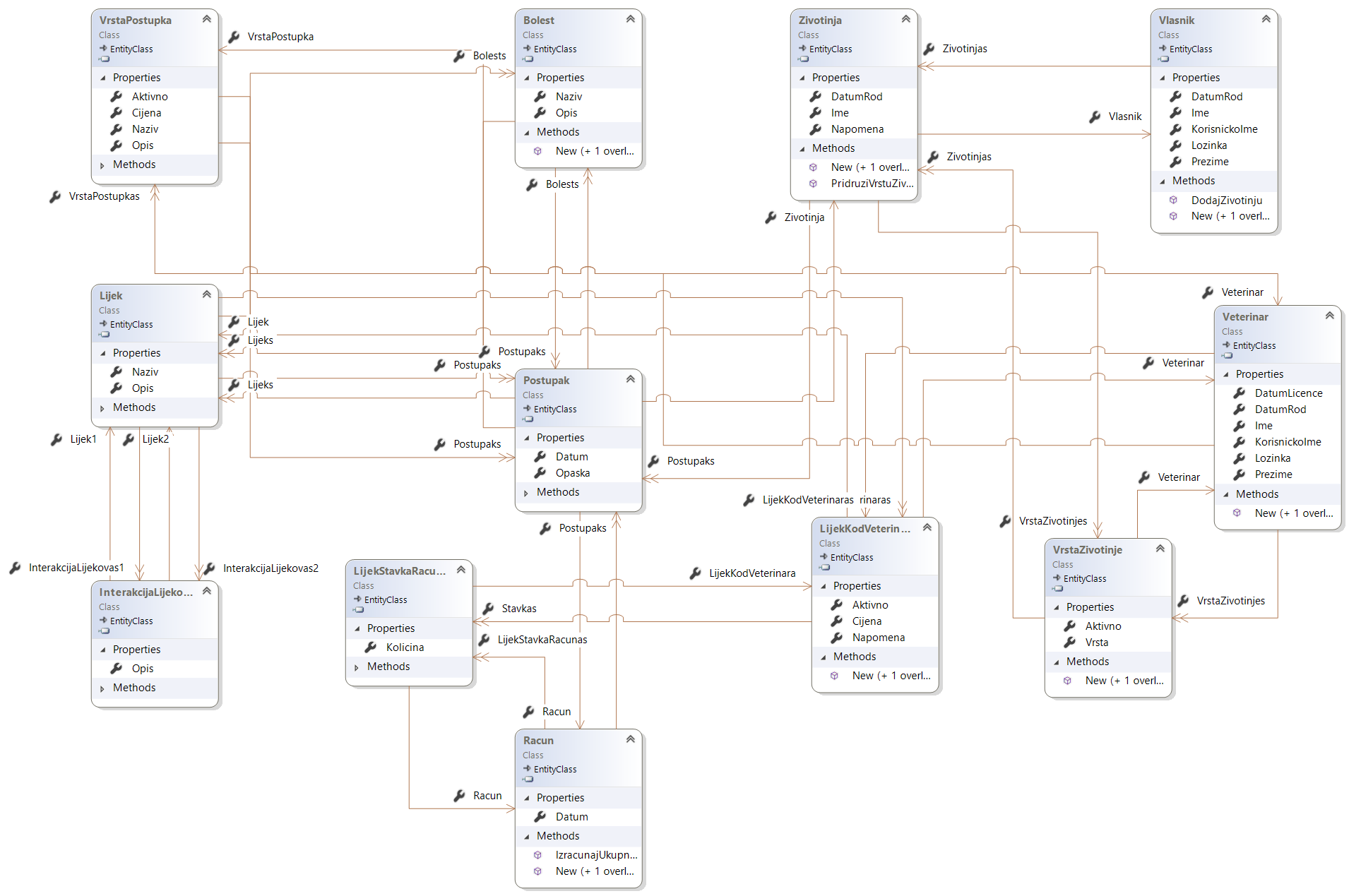
Nad svojim već registriranim životinjama, vlasnici mogu pregledavati povijest obavljenih postupaka nad životinjom s pripadnom terapijom i dijagnozama obavljenih kod različitih veterinara

Vlasnici mogu vidjeti osnovne podatke o veterinaru kod kojeg je obavljen postupak.

### Use cases

1. Registracija vlasnika
2. Promjena lozinke vlasnika
3. Prijava vlasnika
4. Registracija životinje
5. Pregled obavljenih postupaka
6. Detalji o lijeku
7. Detalji o bolesti

# Opis modela



Slika Dijagram razreda modela

U našem modelu svi razredi sa gornjeg dijagrama razreda su entiteti. Svaki od razreda koji smo koristili za modeliranje nije važan samo zbog svojih atributa nego i zbog identiteta. Tako npr. vrsta postupka bi bila *Value* objekt kada ne bi pripadala nekom veterinaru nego bi bila samo vrijednost kao niz znakova. No, potrebno je ostvariti ograničavanje vrijednosti koje se mogu koristiti kao vrste postupaka, a vrste određuju sami korisnici te je bilo potrebno da ih označimo kao entiteti. Takva situacija ponavlja se s raznim vrstama u modelu.

Razlikujemo klase:

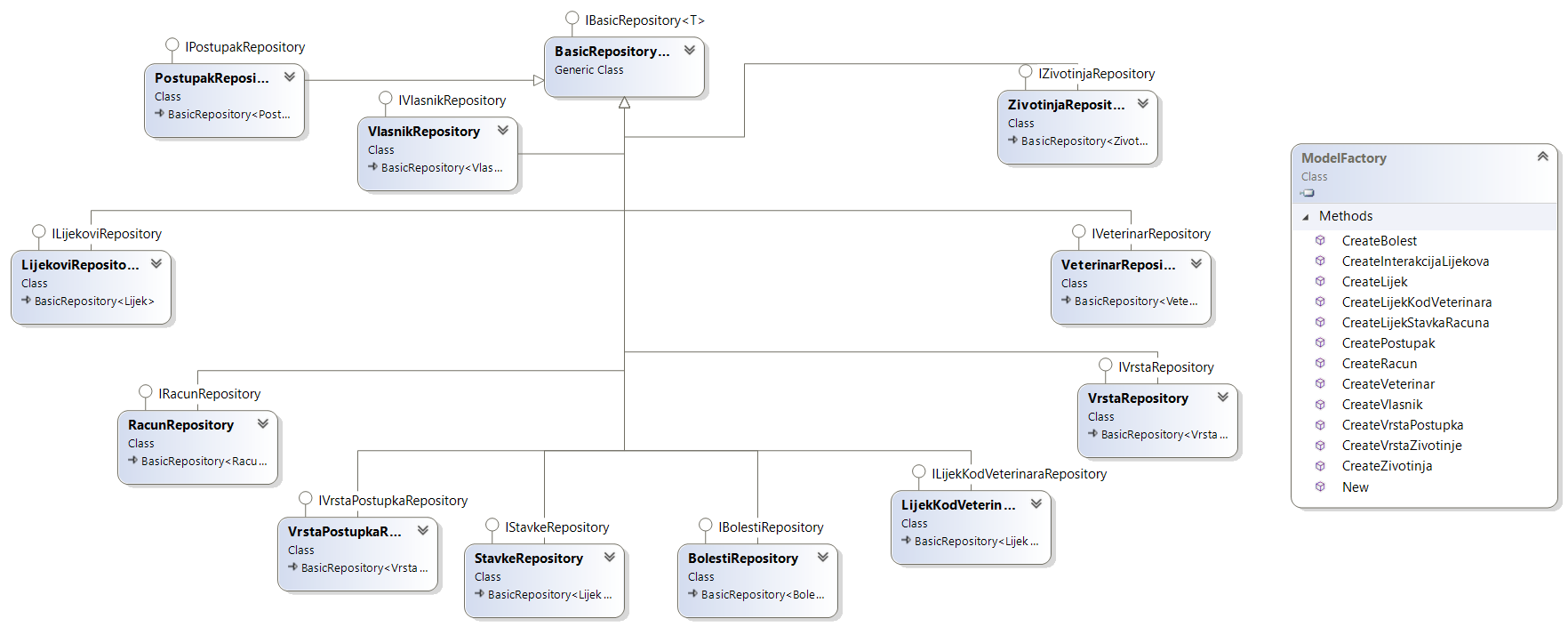
* Veterinar
  + Veterinar koji se prijavljuje i sadrži svoju vlastite kolekciju vrsta životinja, lijekova koje prodaje te postupaka koje nudi.
* Vlasnik
  + Vlasnik koji se prijavljuje i ima svoje životinje

\*Odlučili smo da veterinar i vlasnik neće nasljeđivati neku apstraktnu klasu Čovjek jer u našem modelu želimo razlikovati dvije vrste korisnika za koje imamo dva sučelja – veterinare i vlasnike te nam ne treba polimorfizam.

* VrstaŽivotinje
  + Razred koji predstavlja vrstu životinje kod veterinara
* Zivotinja
  + Razred koji predstavlja životinju
* Bolest, Lijek
  + Razredi koji označavaju jednostavne koncepte iz domene – bolesti i lijekove
* LijekKodVeterinara
  + Razred koji predstavlja lijek koji veterinar prodaje (npr. lijek 1 kao mast po cijeni od 30kn i lijek 1 kao kapsule po cijeni od 50 kn).
* InterakcijaLijekova
  + Razred koji predstavlja interakciju dva lijeka
* Postupak
  + Razred koji predstavlja postupak – ima dijagnozu i terapiju te vrstu postupka
* VrstaPostupka
  + Razred koji predstavlja vrstu postupka koja se obavlja (svaki veterinar ima svoje postupke)
* Racun
  + Razred koji predstavlja račun za naplaćene postupke i lijekove
* LijekStavkaRacuna
  + Razred koji predstavlja količinu kupljenog lijeka koje veterinar nudi na jednom računu.

Radi preglednosti, agregati nisu nacrtani na gornjem dijagramu nego ovdje nabrojani:

* Agregat oko Lijek (korijen): Lijek, LijekKodVeterinara, LijekStavkaRacuna, InterakcijaLijekova, Postupak, Bolest, Racun
* Agregat oko Bolest(korijen): Bolest, Lijek, Postupak
* Agregat oko Zivotinja(korijen): Zivotinja, Vlasnik, VrstaZivotinje, Postupak, Racun
* Agregat oko Veterinar(korijen): Veterinar, VrstaPostupka, VrstaZivotinje, Zivotinja, LijekKodVeterinara, Postupak
* Agregat oko Postupak(korijen): Postupak, Zivotinja, Racun, Lijek, Bolest



Slika Repozitoriji i tvornica

Repozitoriji i tvornica nisu stali na prethodni dijagram pa su prikazani na zasebnom. Svi repozitoriji nasljeđuju osnovni repozitorij, a sva njihova vlastita sučelja nasljeđuju osnovno sučelje.

Tvornica nudi metode za kreaciju svakog razreda iz modela, umjesto jednog razreda za svaki entitet, koristili smo jednu veliku tvornicu.

# Opis implementacije perzistencije

U projektu smo koristili NHibernate za objektno/relacijsko mapiranje. Objektno/relacijsko mapiranje je tehnika koja se koristi za pretvaranje podataka između nekompatibilnih sustava koji koriste objekte te sustava koje koriste relacije.

U oblikovanju, krenuli smo od objektnog modela podataka. Kako bi osigurali da podaci ostanu sačuvani u SQLite bazi podataka, koristili smo preslikavanje u Fluent NHibernate-u. Fluent NHibernate omogućuje definiranje mapiranja u programskom jeziku C#, umjesto XML-a.

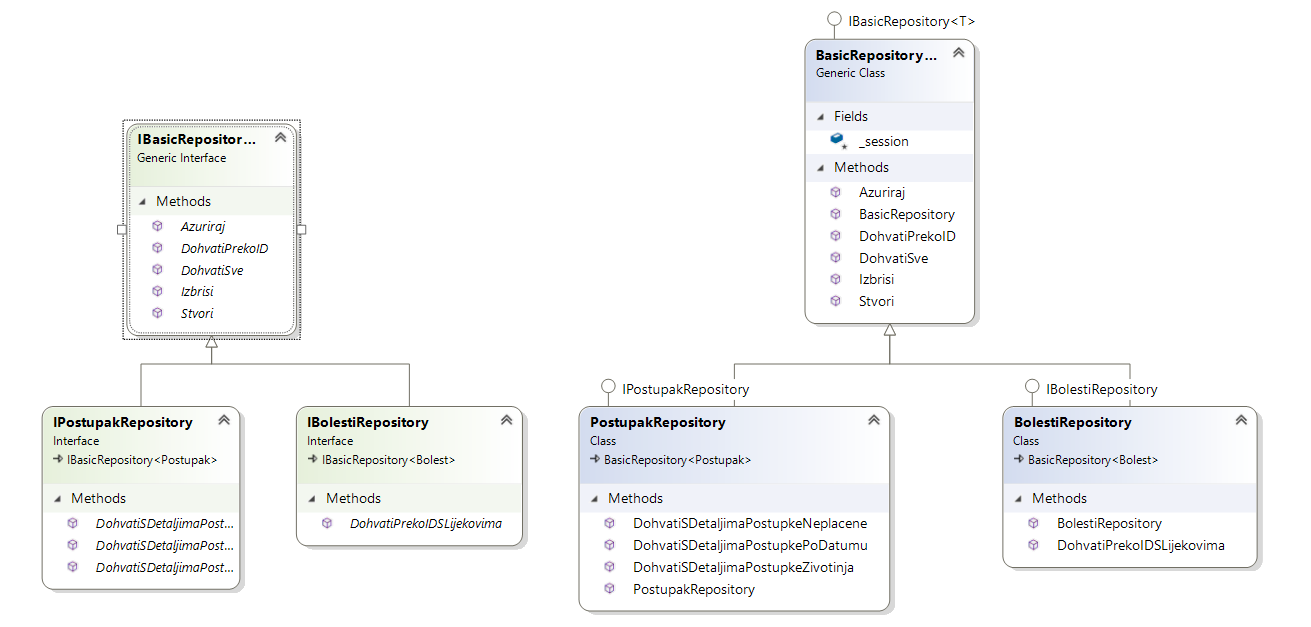
Takva mapiranja se prevode skupa s ostatkom koda te isključuje eventualne pogreške u pisanju ili tipiziranju.

Potrebno je konfigurirati NHibernate na početku rada aplikacije kako bi se znao osloniti na korišteno mapiranje.

U našem projektu, po pokretanju aplikacije (desktop ili web) pozivamo metodu OpenSessionFactory u razredu NHibernateService. U toj metodi NHibernate-u konfiguriramo lokaciju i vrstu baze podataka te mu označavamo gdje se nalaze potrebni razredi za preslikavanje. Ta metoda na kraju vraća tvornicu sjednica (ISessionFactory) koja se koristi za stvaranje novih sjednica. Ta tvornica je singleton objekt.

Za komunikaciju s bazom koristimo obrazac Repozitorij. U našem projektu postoji osnovno generičko sučelje IBasicRepository. Ono je parametrizirano svakim Entity objektom u našem modelu. To sučelje nudi osnovne metode za komunikaciju s bazom podataka (ili nekim drugim izvorom podataka. Postoji i konkretna generička implementacija sučelja.

Da bi se proširile funkcionalnosti osim ovih osnovnih, za svaki razred u modelu postoji posebno sučelje i konkretan razred implementacije, koji nasljeđuju osnovno generičko sučelje.



Slika Repozitoriji u projektu

Slika 1 prikazuje primjer izdvojena dva konkretna repozitorija iz našeg modela. Svaki implementira pripadno sučelje te nasljeđuju osnovni repozitorij.

Razdvajanje u sučelja osim olakšavanja nadogradnje, omogućuje i testiranje preko mocking obrazaca.

NHibernate razlikuje tri stanja objekta:

* Transient
  + Objekt koji nema svoju reprezentaciju u bazi podataka
* Persistent
  + Objekt koji ima svoju reprezentaciju u bazi podataka i čije se promjene prate unutar sjednice
* Detached
  + Objekt koji ima svoju reprezentaciju u bazi podataka, ali se promjene ne prate

Da bi ovi konkretni repozitoriji komunicirali s bazom podataka pomoću NHibernate-a potreban je objekt sjednice (ISession objekt). Svaki repozitorij ima sjednicu koja je zadužena za komunikaciju s bazom podataka i održavanje perzistencije. Objekti mogu biti perzistentni samo u okviru sjednice te su povezani s točno jednom sjednicom. Nakon zatvaranja sjednice oni postaju odvojeni (*detached*). Sjednice u svakoj metodi repozitorija otvaraju i na kraju zatvaraju transakciju (ITransaction objekt) koji predstavlja skup događaja koji se odvija u cijelosti ili se uopće ne odvija (*rollback*). Osim perzistentnih i odvojenih, objekti mogu biti i tranzijentni – objekti koji još nisu povezani s redom u tablici i nisu perzistentni.

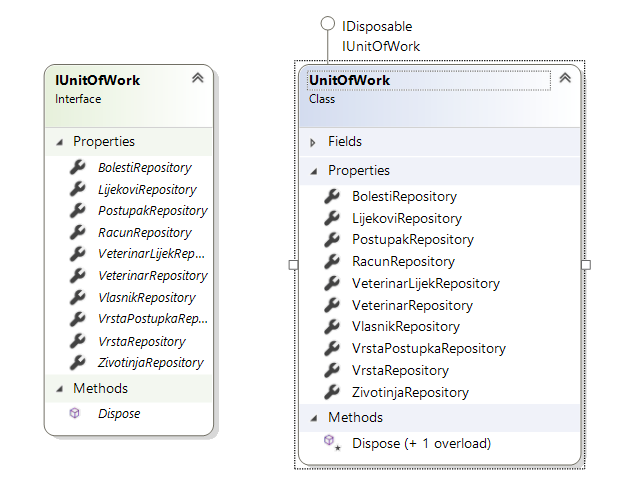
Često nam je potrebno raditi s više različitih repozitorija (npr. repozitoriji za bolesti i postupke) te smo koristili obrazac *Unit of Work.* Taj se obrazac koristi da se osigura da prilikom korištenja više različitih repozitorija oni dijele zajednički objekt sjednice.

Tako svaki naš prezenter i upravljač, prilikom stvaranja dobiva svoj *Unit of Work* objekt koji po potrebi stvara željeni repozitorij te svi stvoreni repozitoriji unutar istog *Unit of Work* objekta dijele istu sjednicu.

*Unit of Work* objekt u prezenteru se zatvara nakon zatvaranja glavne forme prezentera (u desktop aplikaciji), odnosno upravljača (u web aplikaciji).

Taj objekt bi se mogao koristiti i za definiranje nekih vlastitih akcija komunikacije s bazom podataka koje ne postoje implementirane u repozitorijima (npr. odjednom dodavati postupak i veterinara).

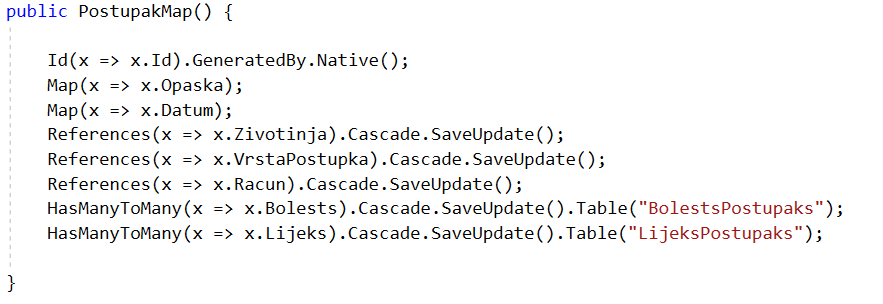
Slika 2 prikazuje strukturu obrasca, koji sadrži sve repozitorije te pohranjuje jedan ISession objekt.



Slika Unit of Work obrazac

Svaki put kad je potrebno komunicirati s bazom podataka, podaci se preslikavaju u objekte ili u relacije. Da bi se ispravno preslikali, potrebno je definirati preslikavanja.

Mapiranja se definiraju u razredima koji nasljeđuju razrede ClassMap<T>, gdje je T razred koji se preslikava.



Slika Primjer mapiranja postupka

Slika 3 prikazuje mapiranje postupka na kojem će biti objašnjeni pojmovi preslikavanja.

Svaki preslikani objekt mora imati svoj Id. U primjeru za preslikavanje postupka, definirano je da Id automatski generira baza podataka.

Pojedinačna preslikavanja nativnih objekata u C#-u, definiraju se preko ključne riječi Map (*integeri*, *stringovi* itd.). U primjeru postupka to su opaska (tipa *string*) i datum (tipa *DateTime*).

Ako naš razred sadrži referencu na neki drugi razred, u relacijskom modelu to se preslikava u vezu preko stranog ključa u relacijskoj bazi podataka te je to potrebno dodatno naglasiti ključnom riječi *References*. U relacijskoj tablici, naš objekt će za svaki objekt koji referenicira dobiti stupac koji će predstavljati primarni ključ u tablici referenciranog objekta (tzv. strani ključ). Druga strana te veze (npr. životinja iz primjera s postupkom) u relacijskoj tablici nema nikakve dodatne stupce. To je veza *One-to-many*.

Veza *Many-to-man*y je veza u kojoj jedan objekt referencira kolekciju drugih objekata, a ti objekti sadrže kolekciju objekata prvog tipa (npr. na postupku se može ustanoviti više bolesti, a bolest može biti ustanovljena na više postupaka). Takve veze potrebno je naglasiti ključnom riječju *HasManyToMany*. Takva veza u relacijskom modelu traži dodatnu tablicu koja sadrži parove u kojima je jedan stupac ključ prvog razreda, a drugi stupac ključ drugog razreda. Takvo preslikavanje potrebno je zbog normalizacije baze podataka i potrebno je navesti ime te dodatne tablice.

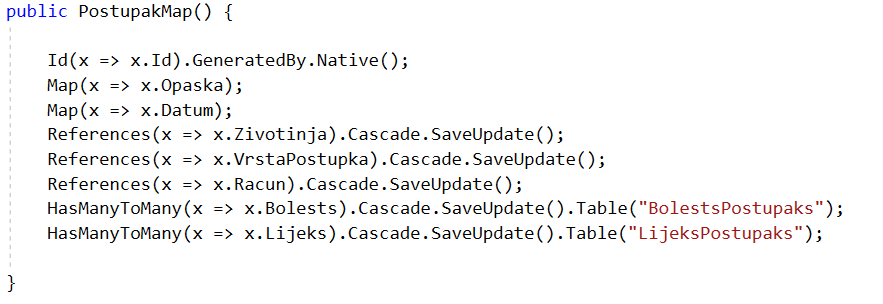
Kod objekata koje sadrže druge objekte koji se također nalaze u bazi podataka, potrebno je definirati što se događa nakon promjene podataka u prvom objektu – kaskadne promjene.

NHibernate nudi više opcija

* None – ne rade se nikakve kaskadne promjene
* SaveUpdate – prilikom spremanja ili ažuriranja objekta provjerava se je li potrebno spremiti ili ažurirati objekt iz veze
* Delete – prilikom brisanja objekta, brišu se i svi objekti iz veze
* Delete-orphan – prilikom brisanja objekta, brišu se i svi objekti iz veze. Dodatno, kada je objekt maknut iz veze, provjerava se pripada li drugoj vezi. Ako ne pripada, briše se.
* All – uključuje sve, osim delete-orphan
* All-delete-orphan – uključuje sve

## Opis mapiranja najvažnijih razreda

Razred Postupak



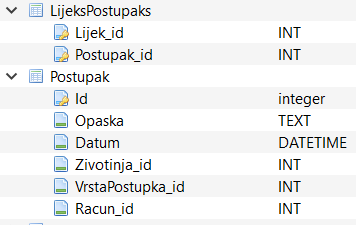
Slika Primjer preslikavanja postupka

Id se generira u bazi podataka.

Opaska i datum se preslikavaju u nativne tipove baze podataka koja se koristi (*string* se preslikava u TEXT, a *DateTime* se preslikavaju u DATETIME u SQLite bazi podataka) te će se preslikati u dva zasebna stupca (stupac Opaska i stupac Datum).

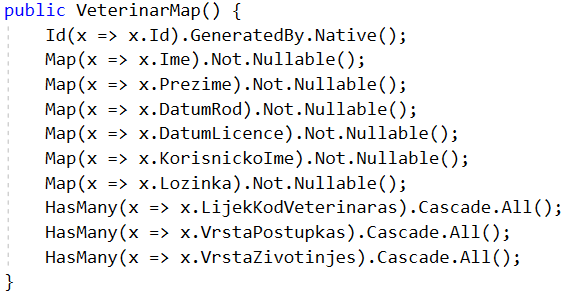
Postupak referencira životinju, vrstu postupka te račun. Prilikom promjene ili spremanja postupka provjerit će se kaskadno je li potrebno spremiti i promjene nad tim vezanim podacima. To će se preslikati u tri zasebna stupca svaki s pojedinačnim stranim ključem (Zivotinja\_id, VrstaPostupka\_id i Racun\_id).

Postupak ima vezu *Many-to-many* prema lijekovima i bolestima te je definirano da se prilikom promjene postupka provjeravaju kaskadno i bolesti i lijekovi. Ta veza spremljena je u novoj tablici BolestsPostupaks, odnosno LijeksPostupaks.



Slika Preslikani postupak

Razred veterinar



Slika Primjer preslikavanja veterinara

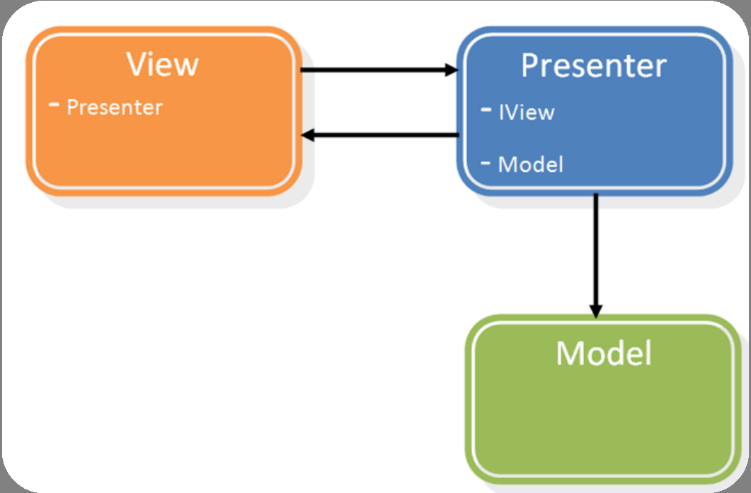
Id se generira u bazi podataka.

Ime, prezime, korisničko ime i lozinka se preslikavaju u nativni tip TEXT, dok se datum rođenja i datum licence preslikavaju u DATETIME. Ti elementi su označeni kao obavezni te ne smiju biti NULL što je označeno Not.Nullable.

Veterinar je također i druga strana *One-to-many* odnosima s lijekovima kod veterinara, vrstama postupaka i životinja koje on definira. Ovdje je kaskadni postupak označen kao All što znači da se, osim kod promjene, i kod brisanja obavlja kaskadno brisanje povezanih elemenata (nema smisla da vrsta postupka koju je veterinar obavljao ostaje, ako veterinara više nema – osim radi eventualne evidencije).

# Opis desktop aplikacije

Aplikacija prati oblikovni obrazac MVP (Model-View-Presenter) te je u skladu s tim podijeljena na te tri logičke cjeline.



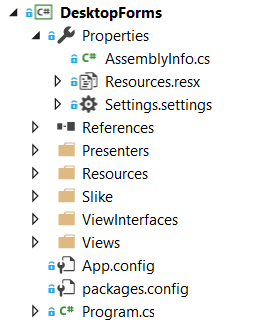
Slika MVP obrazac

Model je naš objektni model obrađen u poglavlju prije.

Pogled (prezentacijski sloj) sadrži forme koje korisniku prikazuju prozore aplikacije. Korisničke akcije (npr. pritisak na gumb) prosljeđuju se prezenteru na obradu. Pogled samo prikazuje podatke korisniku, dok ih prezenter obrađuje. Svaki put kada korisnik pokrene neku akciju, pogled pozove odgovarajuću metodu svog prezentera te mu prenese korisnikov unos, ako je on potreban.

Pogled implementira sučelje za komunikaciju s prezenterom (Slika 7 – IView). To sučelje sadrži podatke koje je potrebno prikazati te samog prezentera. Prezenter sadrži referencu na pogled preko sučelja, tako da ne ovisi o konkretnom pogledu.

Prezenter sadrži reference na repozitorije koji su mu potrebni za manipulaciju podatcima koje je dobio od pogleda te dohvaća i obrađuje podatke iz baze podataka i šalje ih u pogled kako bi se prikazali korisniku. Sam model nema referencu na niti jednog prezentera ili pogled, on je u potpunosti izoliran od ostatka aplikacije. Model i repozitoriji su globalno vidljivi u aplikaciji. Sami razredi modela sadrže metode koje definiraju ponašanje tog razreda te ih prezenter poziva kada je to potrebno.



Slika Prikaz rasporeda desktop aplikacije

Slika 8 prikazuje raspored MVP elemenata u projektu.

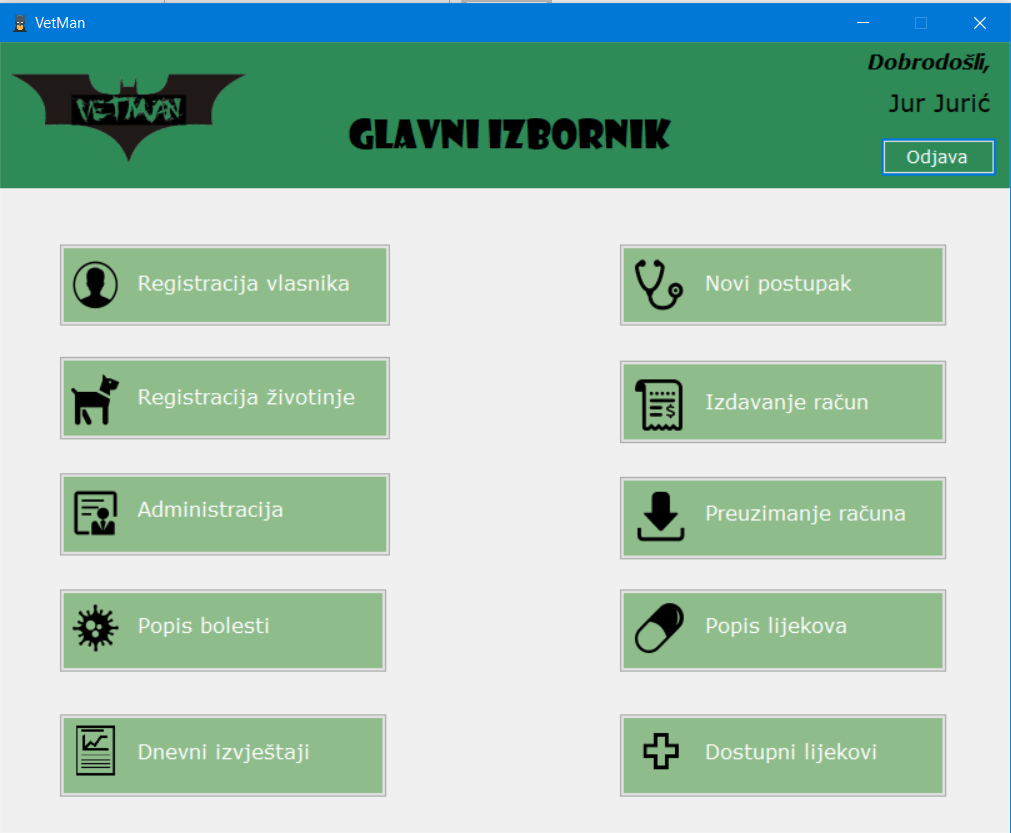
Također, u aplikaciji se koristi i obrazac Promatrač (*Observer)* koji omogućava automatsko osvježavanje podataka koji se prikazuju u nekoj formi, ako se podaci dodaju u drugoj.

Postoje parametrizirana sučelja IObserver i ISubject. Konkretni promatrači su svi zainteresirani za promjene subjekta, dok su subjekti oni koji obavještavaju svoje promatrače kada dođe do promjene. Promatrači se mogu prijaviti i odjaviti sa subjekta.

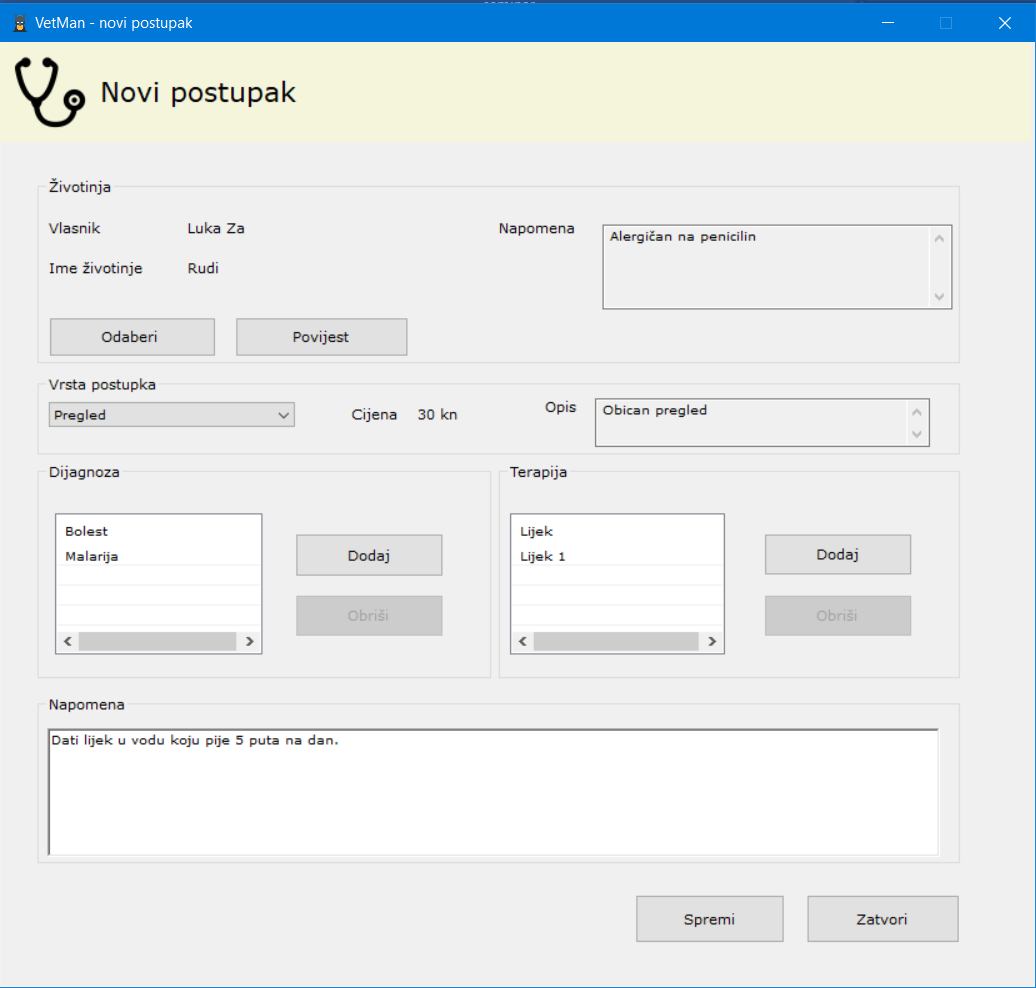
U našem slučaju subjekt je osnovni repozitorij na koji se prijavljuju promatrači te koji obavještava promatrače kada dođe do promjene u podacima kod dodavanja, ažuriranja ili brisanja.

Konkretni promatrači u našem slučaju su zainteresirane forme. Forme se prijavljuju na subjekt prije otvaranja prozora, a odjavljuju nakon zatvaranja prozora. Nakon što dođe do obavijesti od subjekta, forme prilagođavaju svoj prikaz podataka.

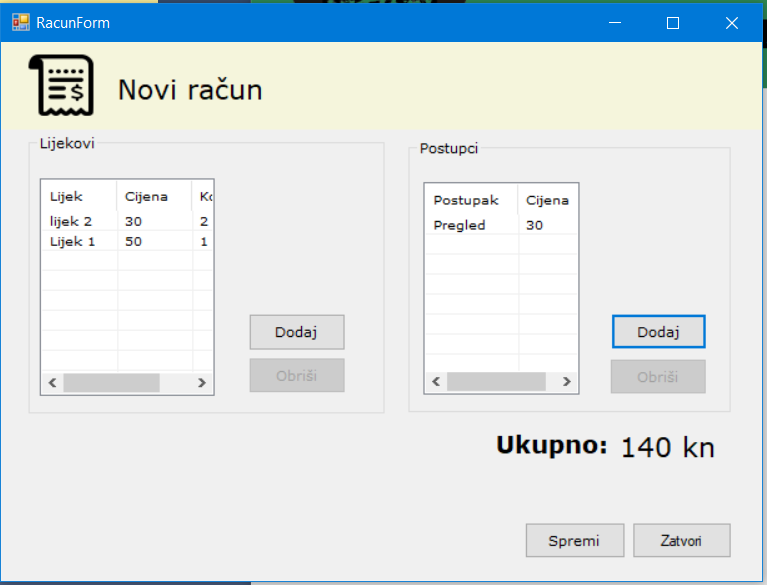
## 4.1. Isječci iz aplikacije



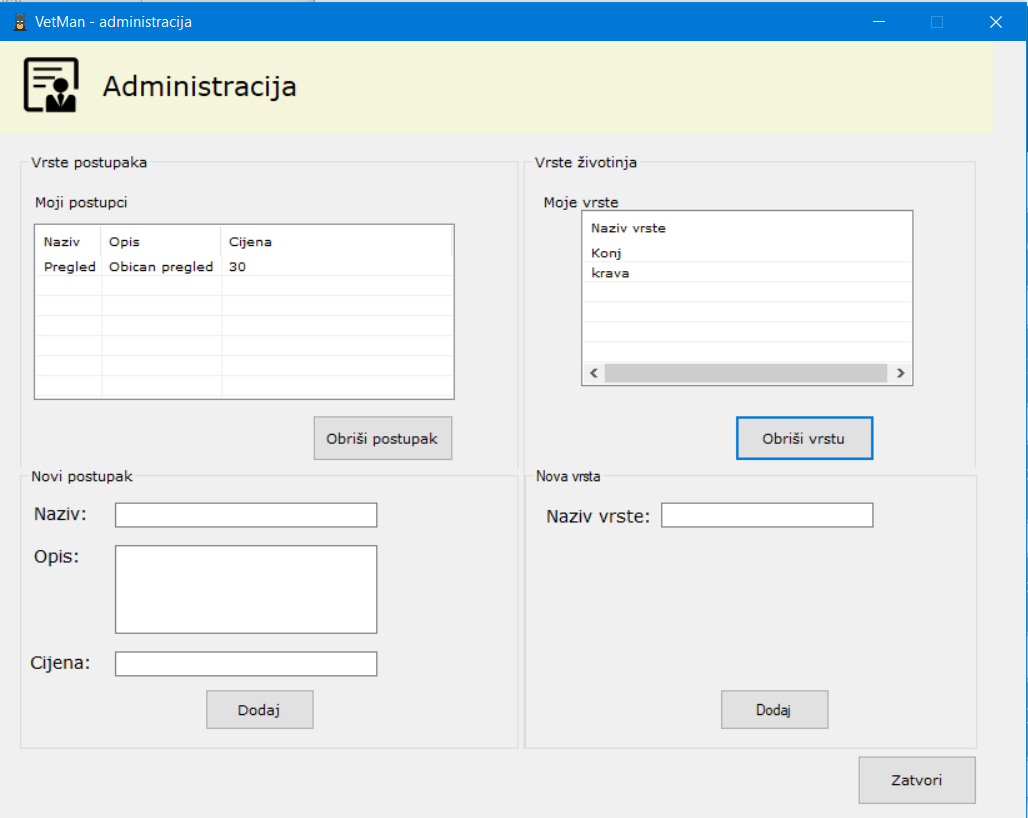
Slika Glavni izbornik



Slika Dodavanje postupka



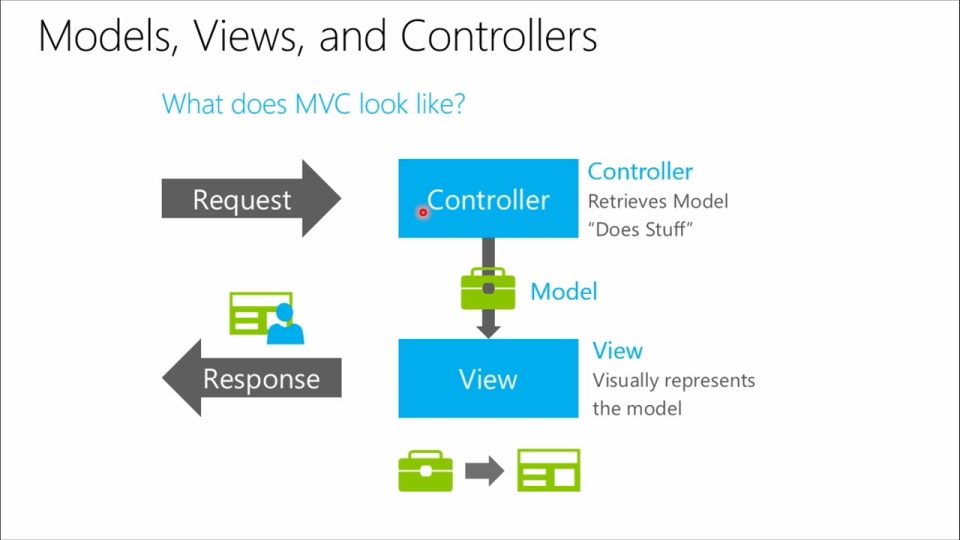
Slika Dodavanje računa



Slika Administracija

# Opis web aplikacije

Web aplikacija napravljena je pomoću ASP.NET Core tehnologije i MVC obrasca.



Slika MVC obrazac

Model je reprezentacija realnog problema i sadrži poslovnu logiku.

Pogledi definiraju raspored elemenata na korisničkom sučelju i interakciju s korisnikom. Pisani su u HTML-u s ugrađenim *Razor markupom*. Za uljepšavanje i dodavanje funkcionalnosti na korisničkoj strani koriste se CSS i Javascript.

Upravljači upravljaju komunikacijom i predstavljaju aplikacijsku logiku.

Korisnici svoje resurse (stanice na webu) zahtijevaju upisivanjem URL-a ili pritiskom na poveznicu. ASP.NET Core preslikava ulazni URL u pripadne upravljače i metode na upravljaču.

U našem slučaju postoje dva upravljača, HomeController i AccountController s pripadnim metodama. Kada korisnik zatraži i upiše URL: *localhost:1234/Home/Zivotinja/2*, to se preslikava u metodu Zivotinja u upravljaču Home s parametrom 2.

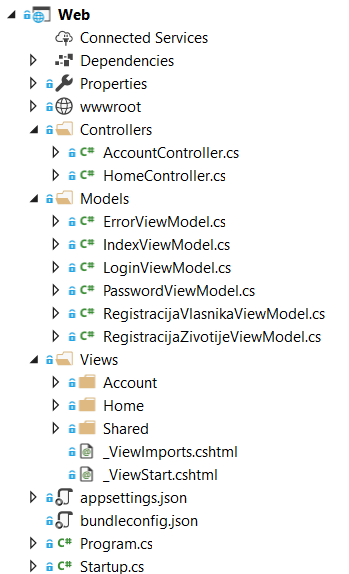
Upravljač, ovisno o zatraženoj akciji, provodi akcije nad modelom – čita, dodaje itd.

Upravljač nakon svakog zahtjeva šalje odgovor jer je komunikacija bazirana na HTTP protokolu. Odgovor je u našem slučaju uvijek pogled. Pogledu šalje podatke potrebne za prikaz preko modela pogleda (*View Model*). Model pogleda (model na Slika 9) sadrži podatke koje su važni za reprezentaciju stranice (npr. životinju čija stranica je zatražena i dr.).

Pomoću *Razor markup-*a, na temelju napisanog HTML predloška, pune se rezervirana mjesta za podatke podacima iz modela pogleda te se generira konačni HTML koji je spreman za prikaz.

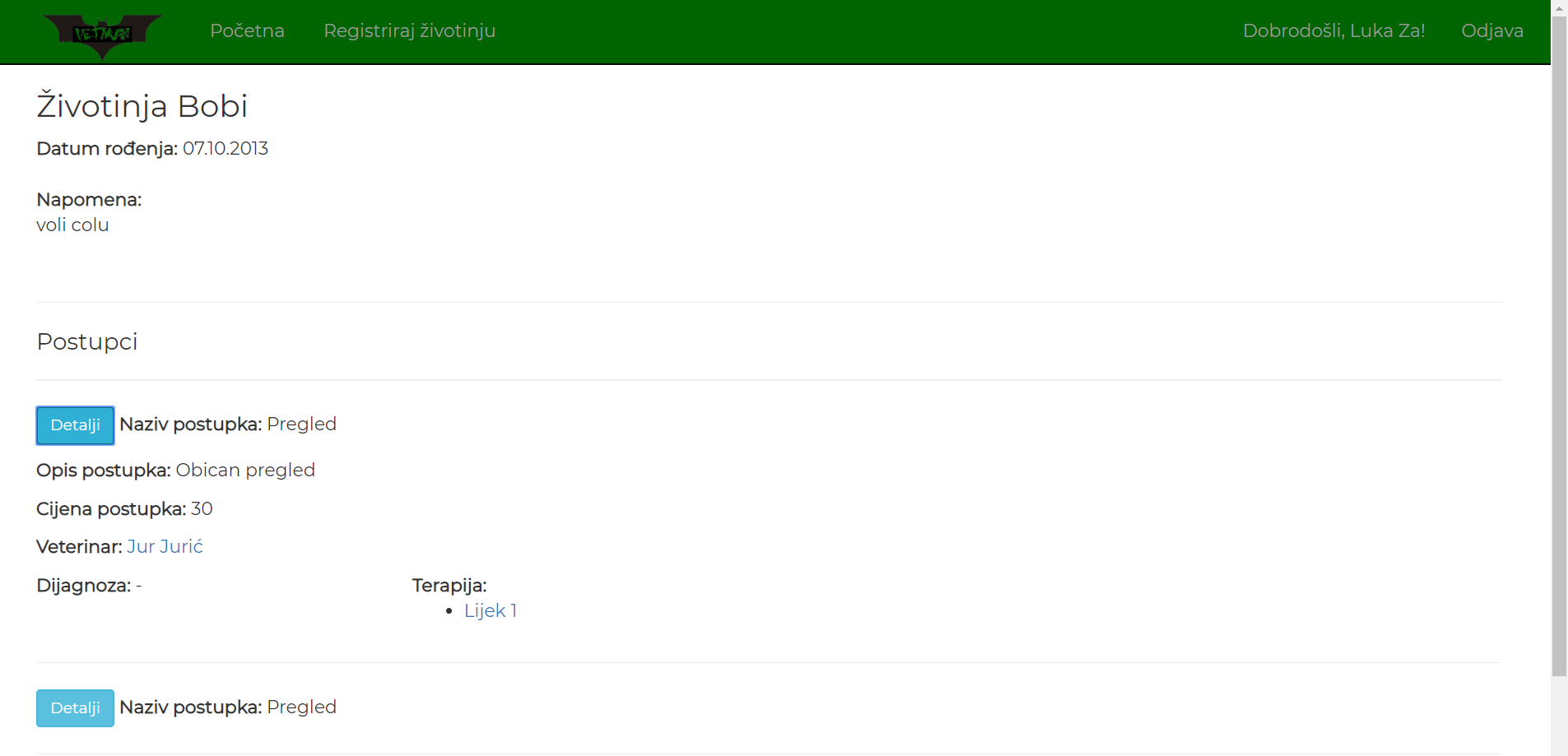
P ogledi su razvrstani u odgovarajuće mape koje nose ime upravljača, a nazvani su prema metodama koje taj pogled vraćaju. Npr. metoda Zivotinja u upravljaču HomeController šalje pogled ZivotinjaView u mapi Home.

Također, pogledi mogu slati modele pogleda i upravljačima u svojim zahtjevima (npr. kod slanja formi).

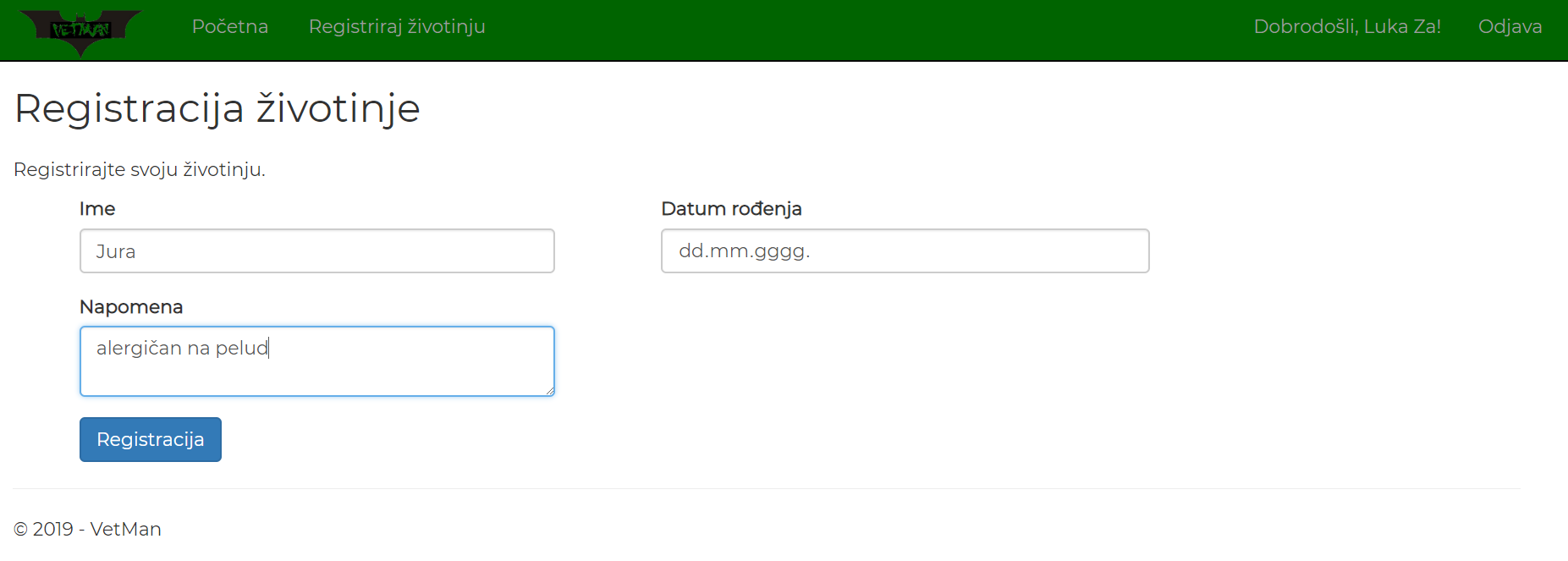


Slika Raspored MVC elemenata u projektu

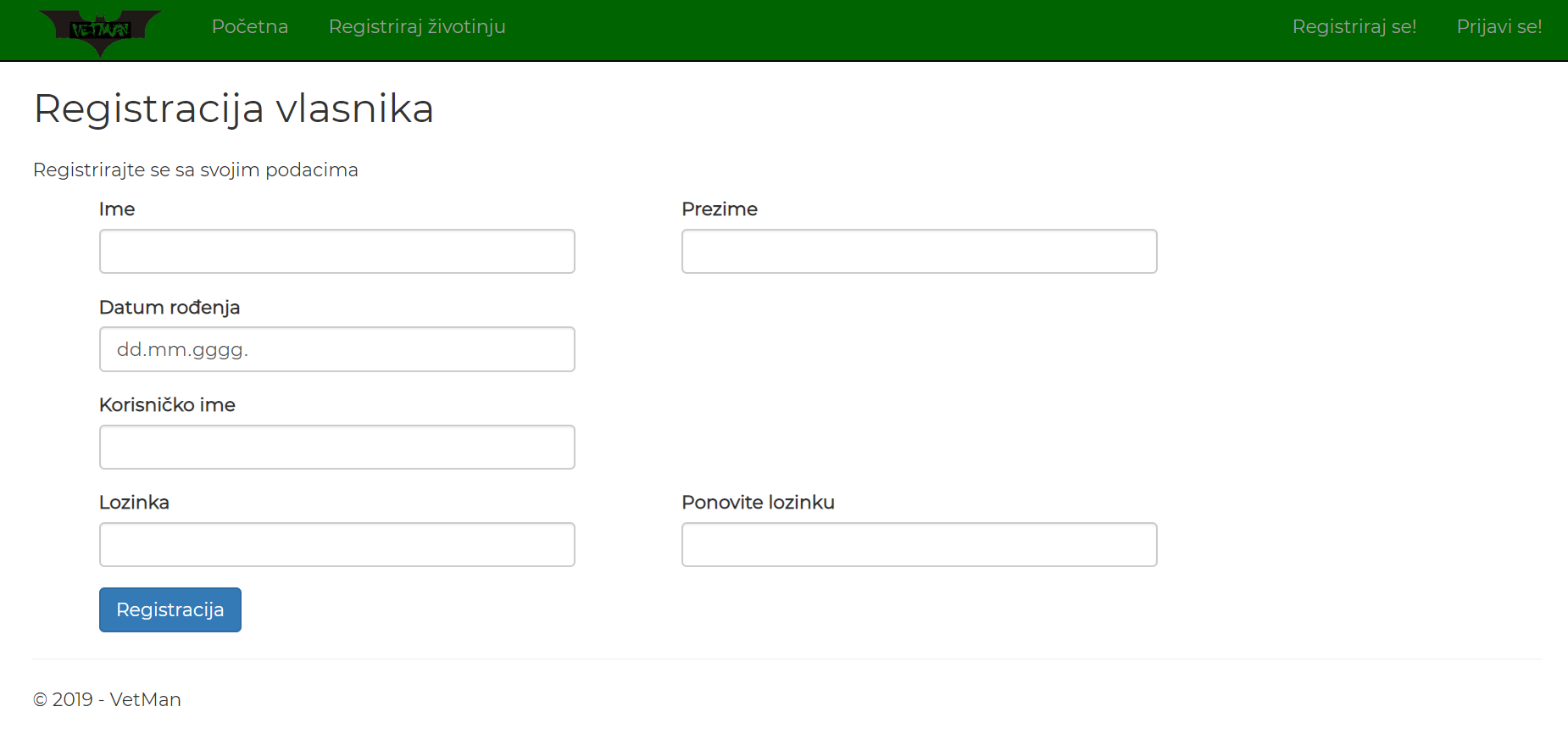
## 5.1. Isječci iz aplikacije



Slika Pregled postupaka nad životinjom



Slika Registracija životinje



Slika Registracija vlasnika