

Elektrotehnički fakultet u Sarajevu

Predmet: Objektno orijentisana analiza i dizajn

Profesorica: prof. dr dipl. ing. Dženana Đonko

Mentorica: Jasmina Bajramović

Dokumentacija projektnog zadatka

“Av-Mau Azil grada Sarajeva”

Tim: CodeX

Ajna Zatrić, Edin Avdić, Nadir Avdagić

Akademska godina 2017/2018.

Opis teme  
Sistem je namijenjen za rad (poslovanje) azila za nezbrinute životinje. Azil pruža privremeno sklonište životinjama (psi i mačke), omogućuje njihovo udomljavanje, veterinarske preglede i dresuru. Osnovna svrha informacionog sistema je da pruži efikasno i jednostavno obavljanje gore navedenih zahtjeva koji se postavljaju pred savremeno sklonište za životinje.  
Problem prekomjernog broja pasa i maca lutalica je poznat široj javnosti u Sarajevu. Neuređenost i decentralizovanost procesa prihvatanja životinja u azilima kod kojih se evidencije vrše manuelno doprinose greškama na svakodnevnom nivou, i otežavaju i usporavaju proces pružanja pomoći i zaštite životinja, a svakako i umanjuju učinak i trud uposlenih.  
  
Predloženi sistem pruža korisniku mogućnost jednostavnog i efikasnog smještanja nezbrinutih životinja u azil, te udomljavanje istih iz azila. Aplikacija će također pružati pogodnosti poput: jedinstvena evidencija o uposlenicima, zatim o udomiteljima, i naposlijetku psima i macama. Sistematizacija podataka predviđena aplikacijom bi, također, omogućila bržu i jednostavniju saradnju uposlenika (vozača, veterinara, higijeničara, upravitelja, udomitelja...).  
Razlog kupovine/unajmljivanja sistema je činjenica da u Sarajevu ne postoji sličan informacioni sistem koji bi olakšao djelovanje u ovoj sferi, te posljedično sveo mogućnost grešaka na minimum. Sistem će pružiti lakši vid saradnje sa korisnicima i udomiteljima, i u konačnici, omogućiti bolji i humaniji suživot stanovnika i četveronožnih stanovnika u gradu Sarajevu.

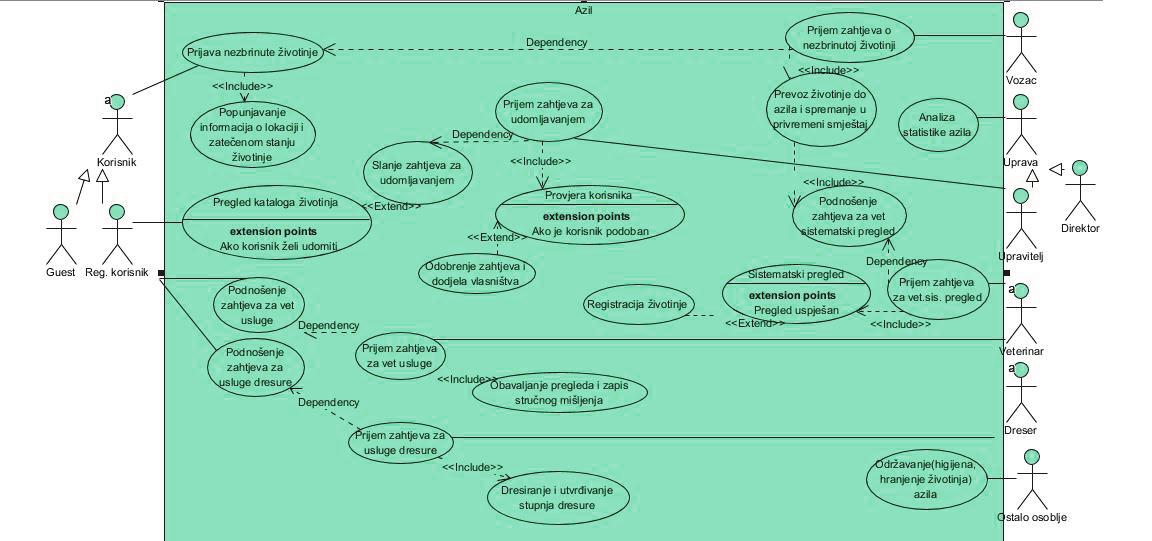
Program projektnih zadataka na predmetu OOAD 2017/2018:

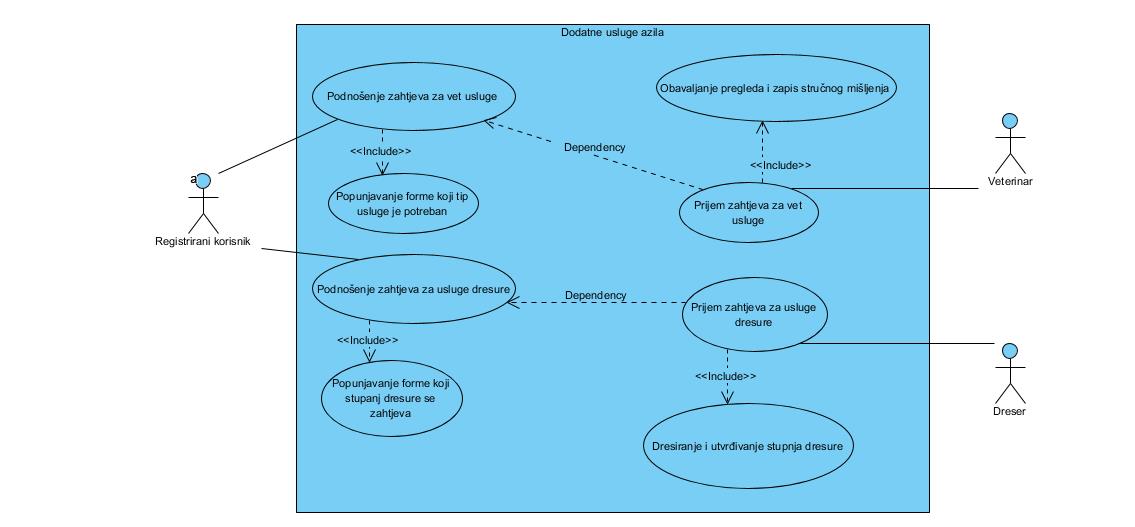
PZ1 - Prijedlog teme . PZ2 - Use case dijagram i scenariji . PZ3 Mokapovi i dijagrami aktivnosti . PZ4 - Korisnički interfejs aplikacije (u XAMLu za admina) i dijagram klasa . P5 - Re-dizajnirati dijagram klasa tako da prati MVVM pattern i popraviti uočene prekšene SOLID principe tako da budu ispoštovani . PZ6 - Rad sa bazom i MVVM u kodu . PZ7 - Kreirati odgovarajuće dijagrame sekvenci i komunikacije. PZ8 - Dizajn paterni PZ9 - Za projekat kreirati dijagrame komponenti, paketa i raspoređivanja. PZ10 - Prezentacija projekta i videi

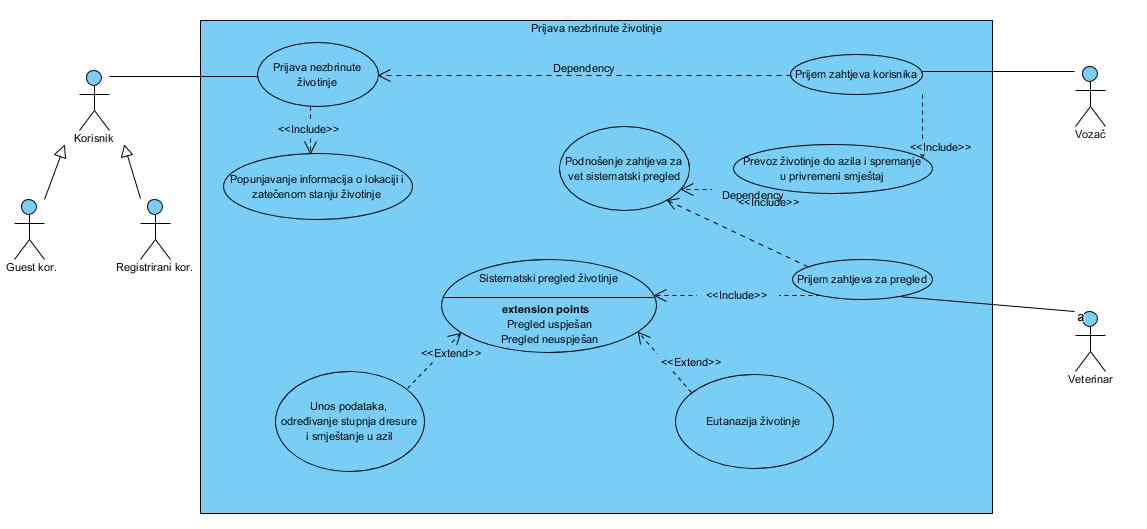
Prijedlog teme

Detaljan opis i prijedlog teme - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

Use case dijagram i scenariji





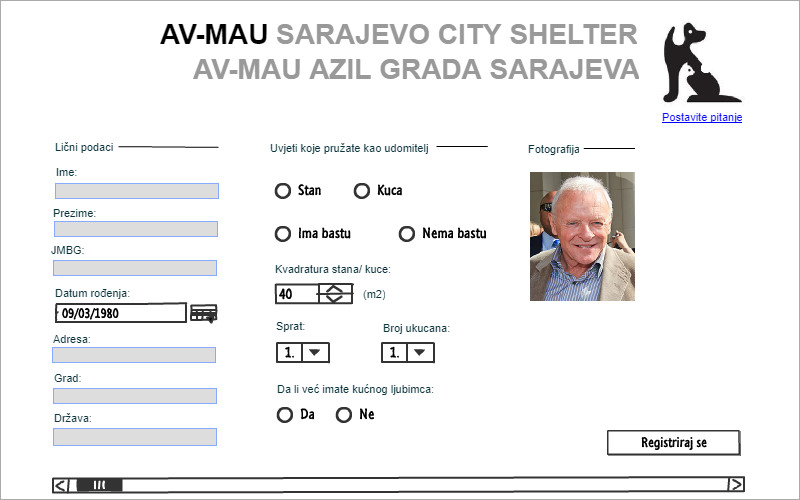
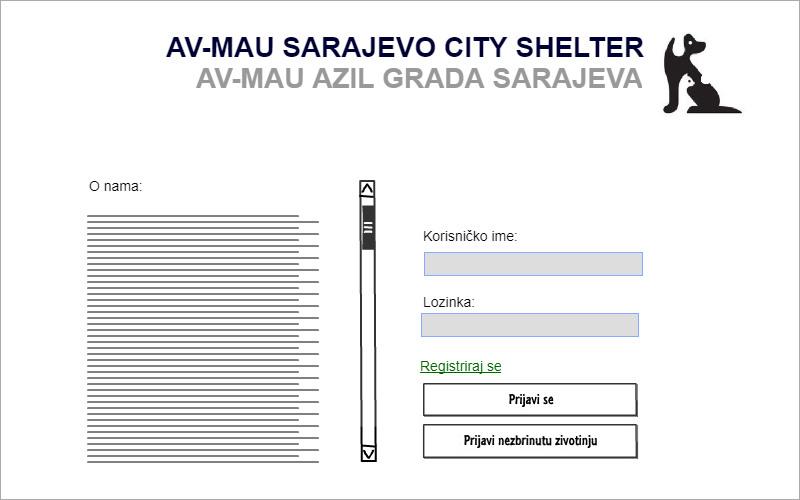


Sve Use case dijagrame projekta i scenarije - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

Radili: Edin Avdić i Nadir Avdagić

Prototipovi formi - mokapovi rađeni u Lumsy-ju

Sve forme osmišljene su i urađene su u vidu mokapova unaprijed, prije pristupanja programiranju



Forme: Početna stranica i registracija korisnika



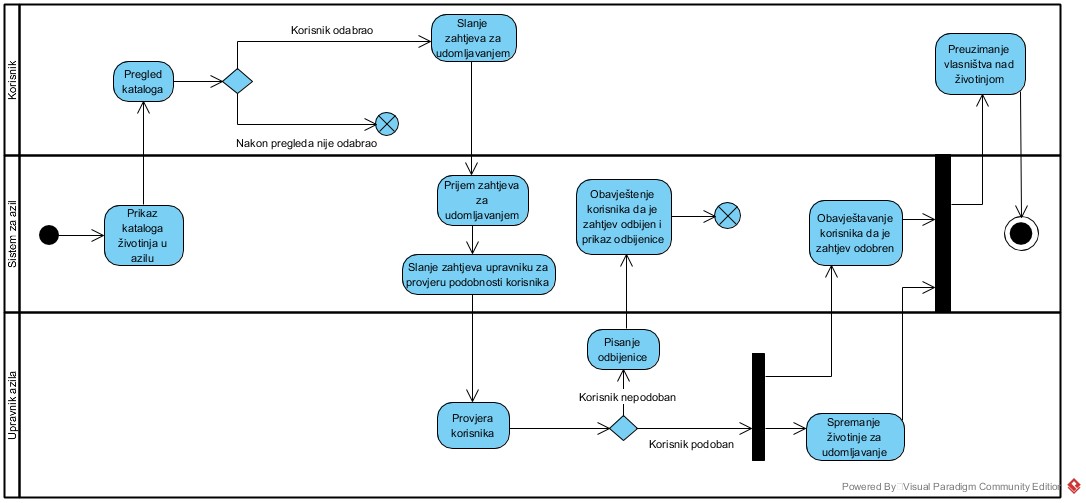
Pretraga PetBook-a filterom

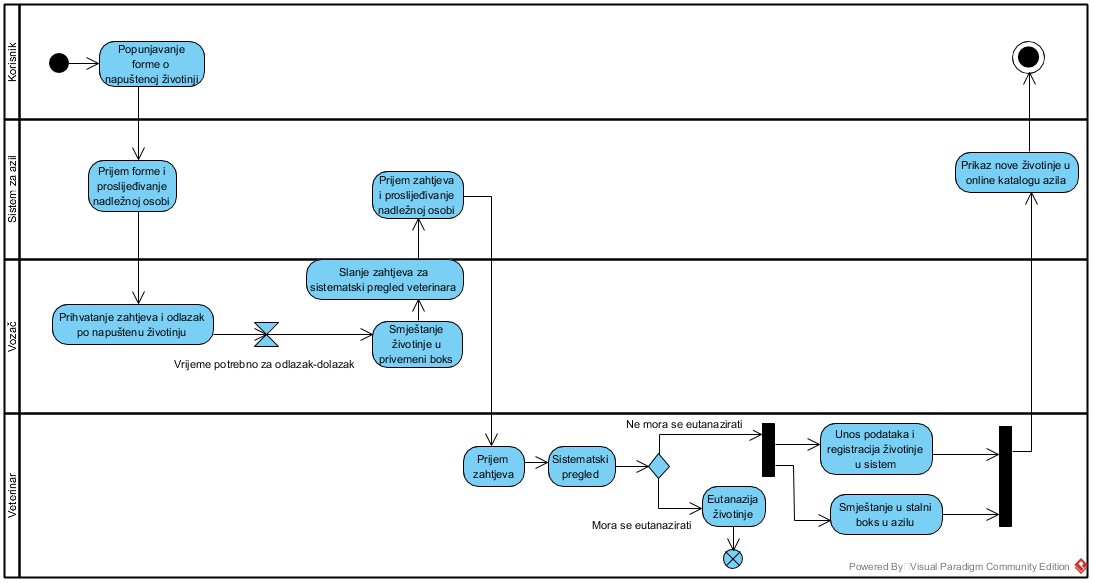


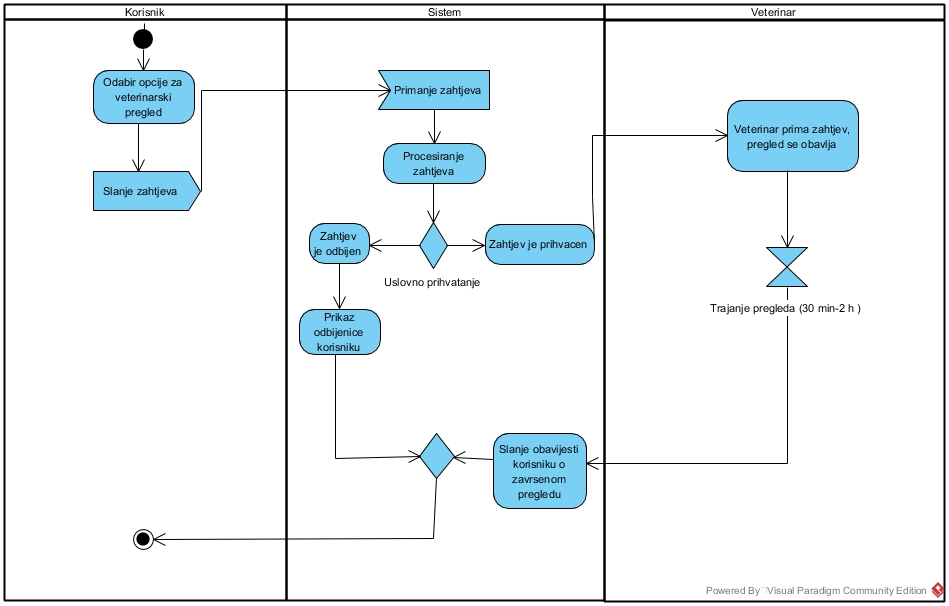
Prototipove svih formi - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/Prototip)

Radila: Ajna Zatrić

Dijagrami aktivnosti

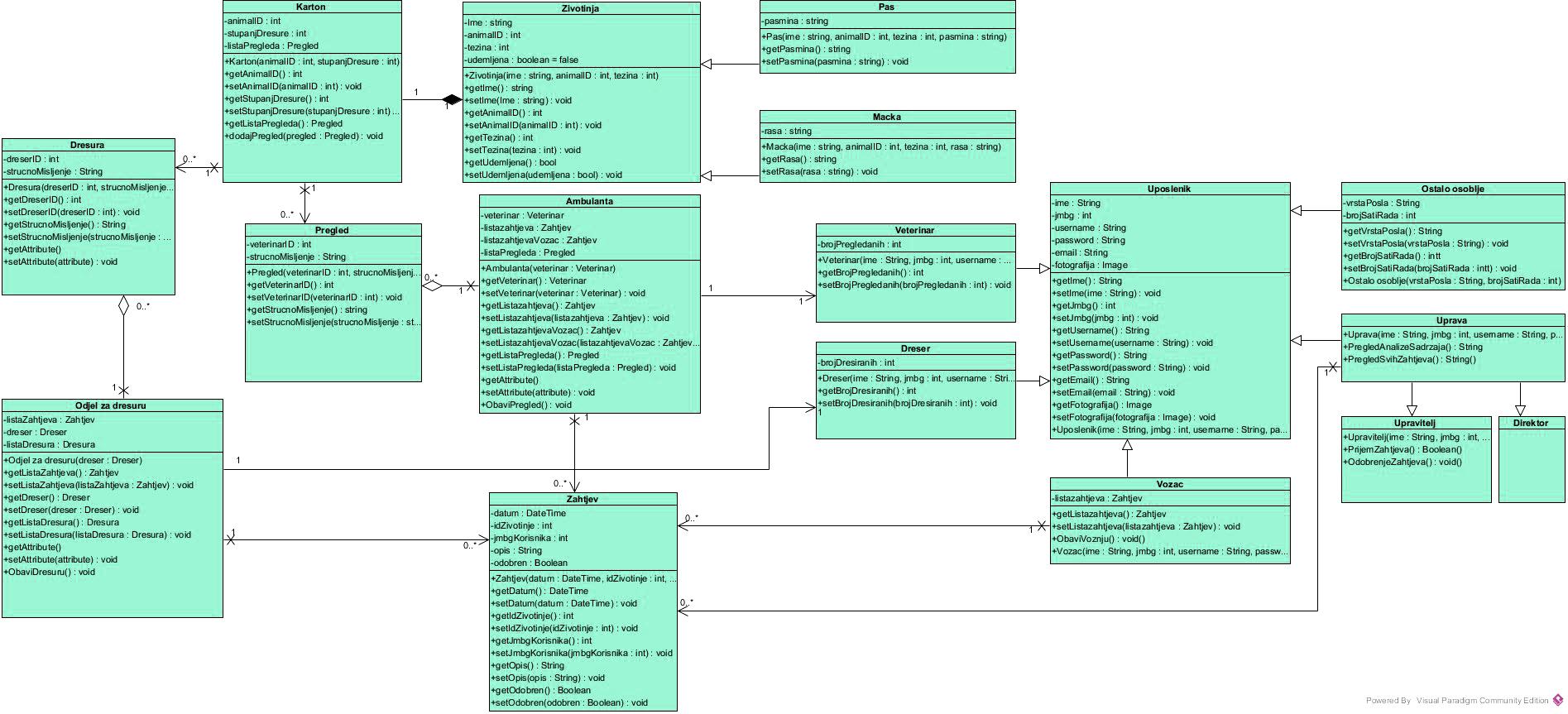




 Sve dijagrame aktivnosti - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/DijagramiAktivnosti).

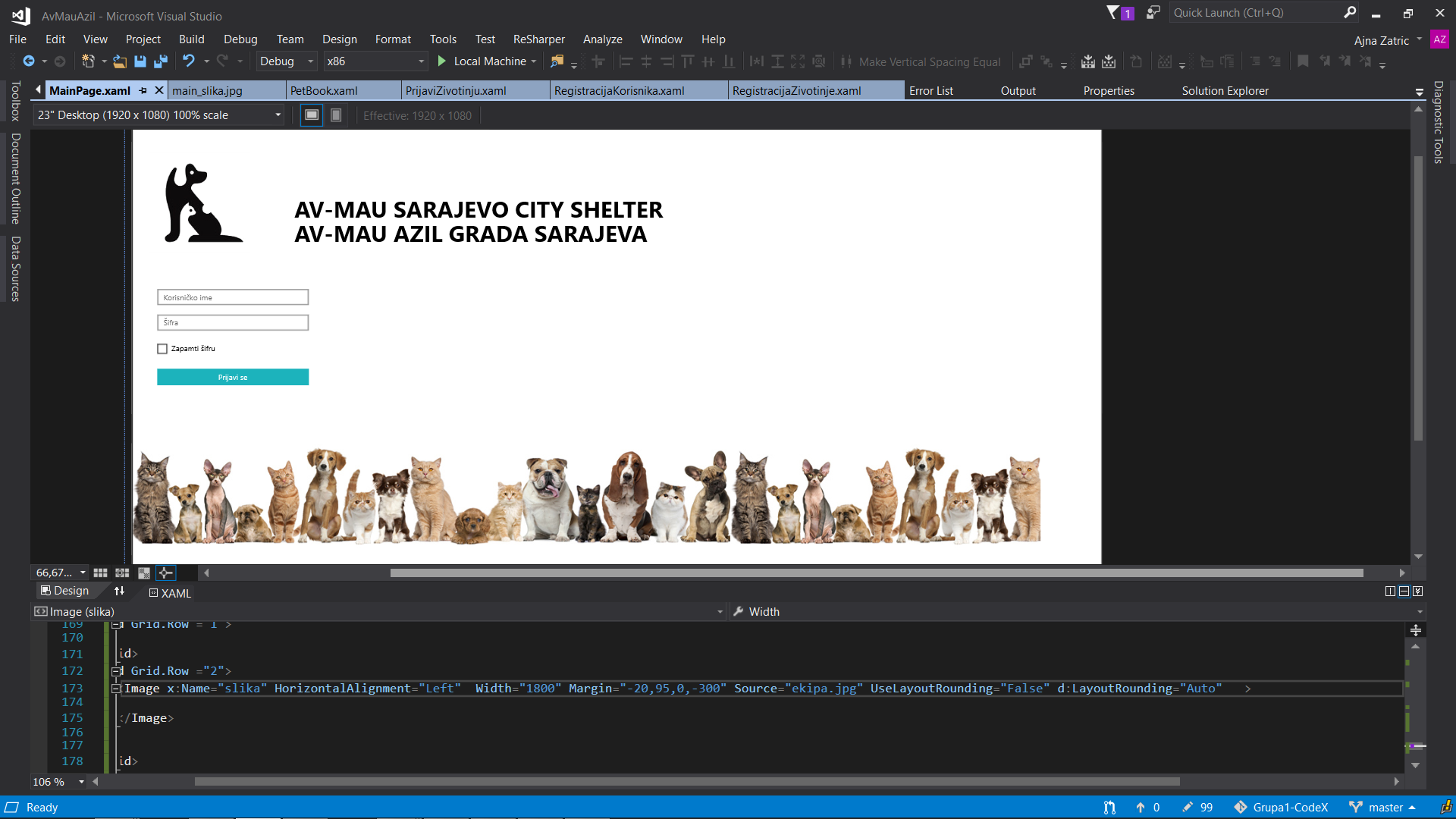
Radili: Nadir Avdagić i Edin Avdić

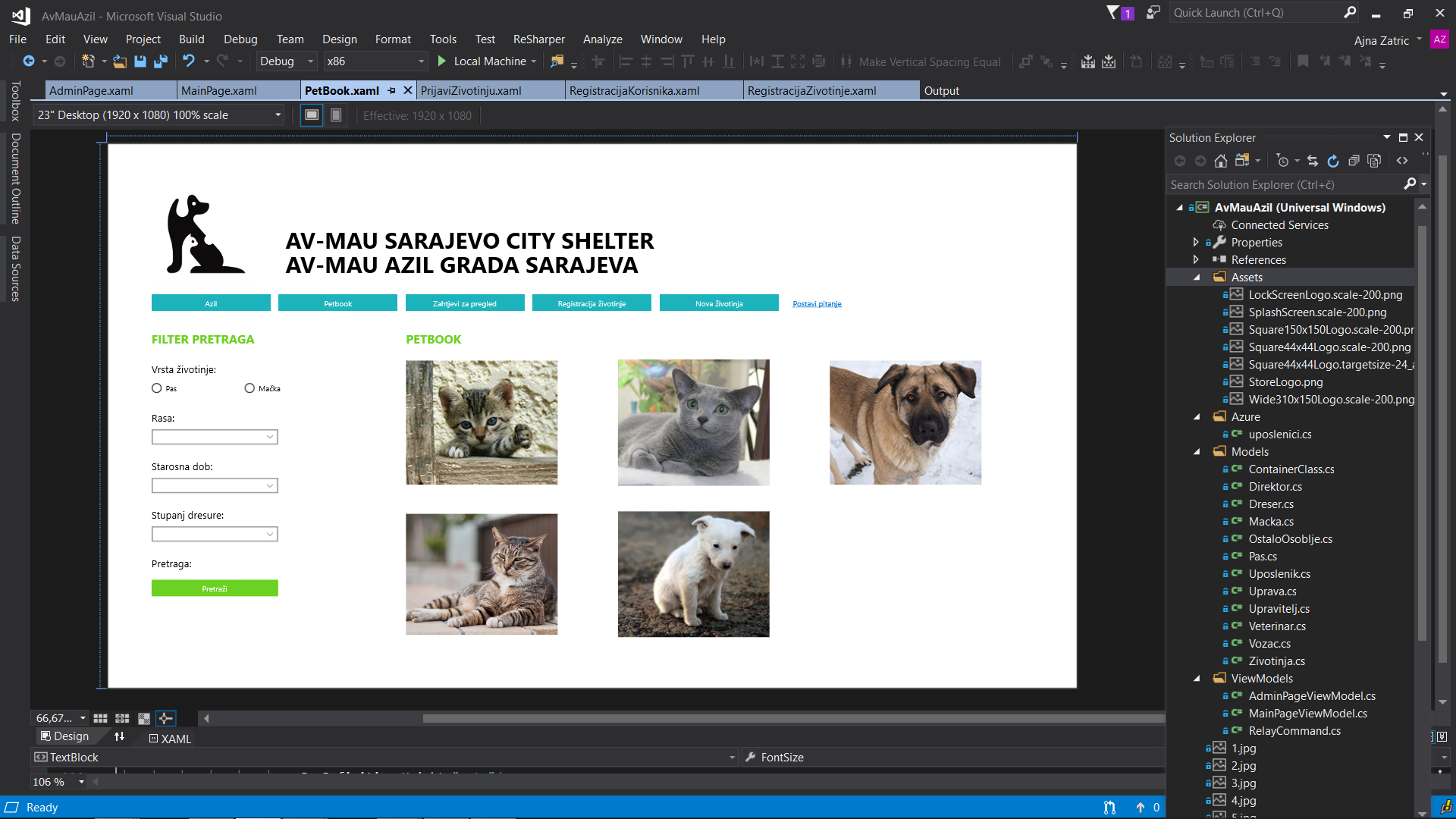
Dijagram klasa

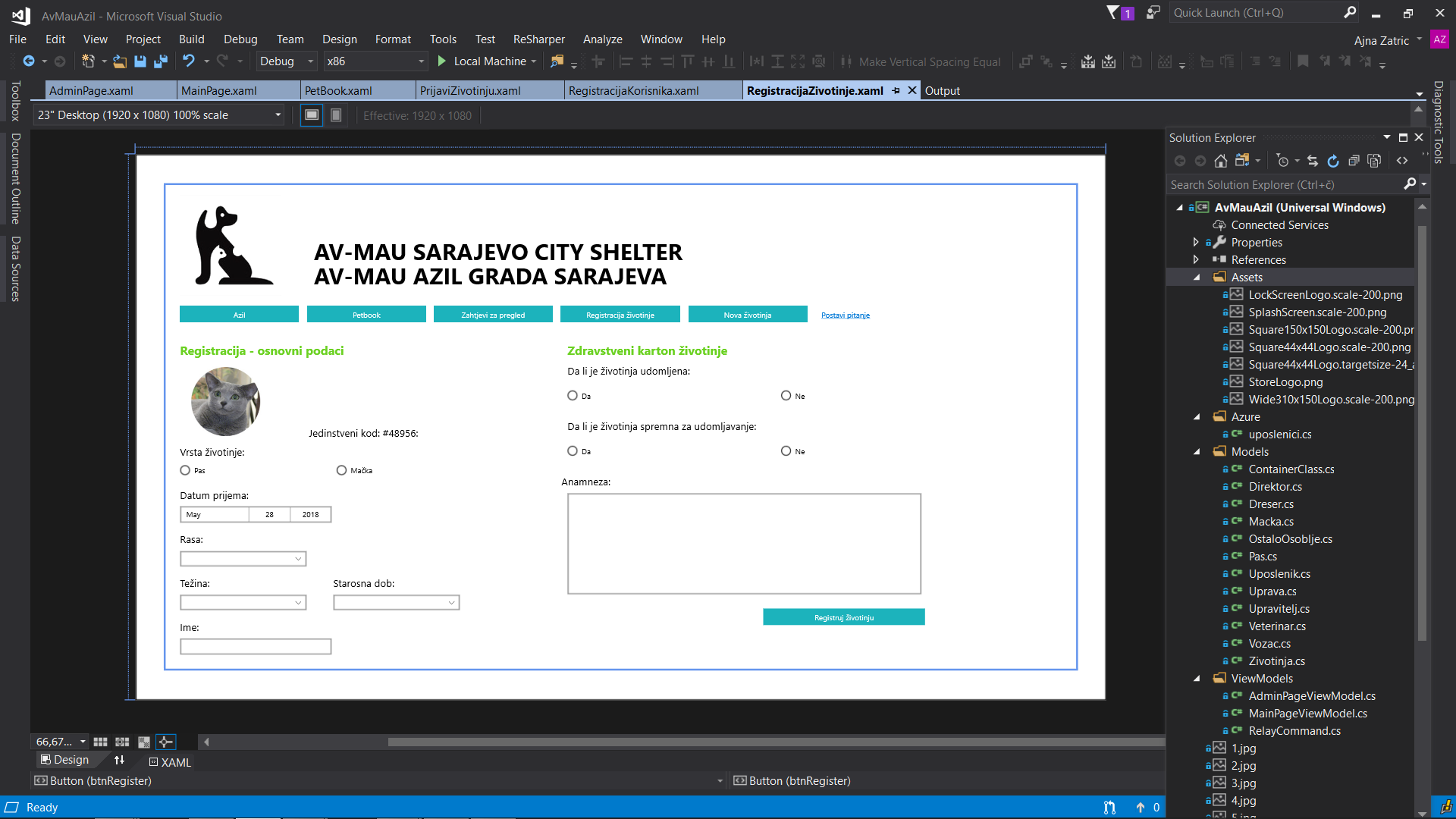


Radili: Nadir Avdagić i Edin Avdić

UWP Programiranje formi u XAML-u







UWP forme u XAMLu

Radila: Ajna Zatrić

Dijagrami sekvenci

Dijagram sekvenci za proces udomljavanja

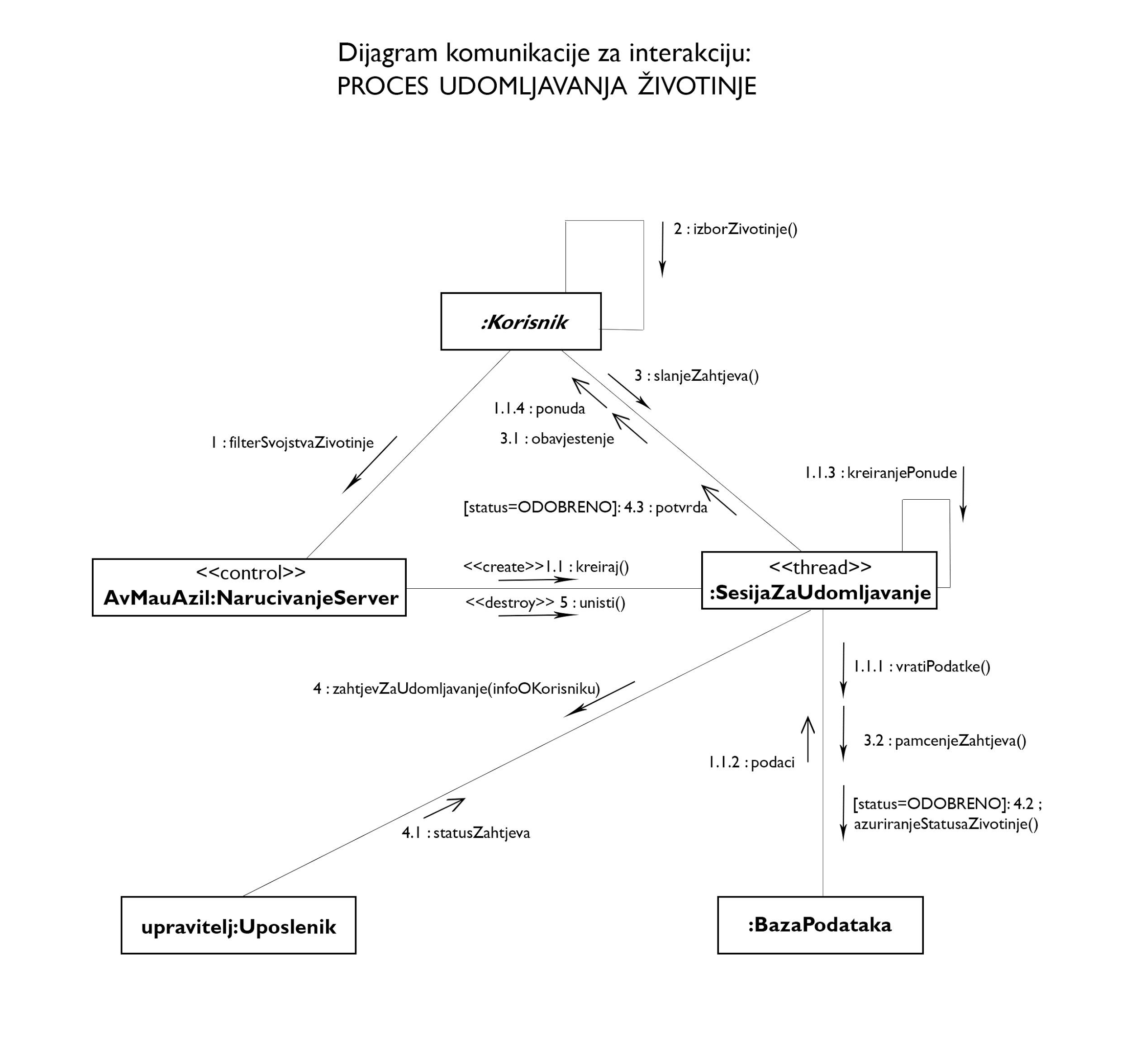


Dijagram sekvenci za pregled životinje



Radili: Nadir Avdagić i Ajna Zatrić

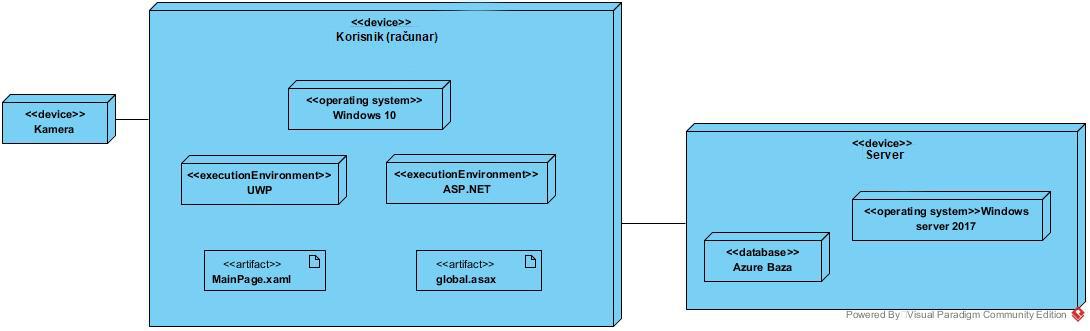
Dijagrami komunikacije



Za ostale dijagrame pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

Radila: Ajna Zatrić

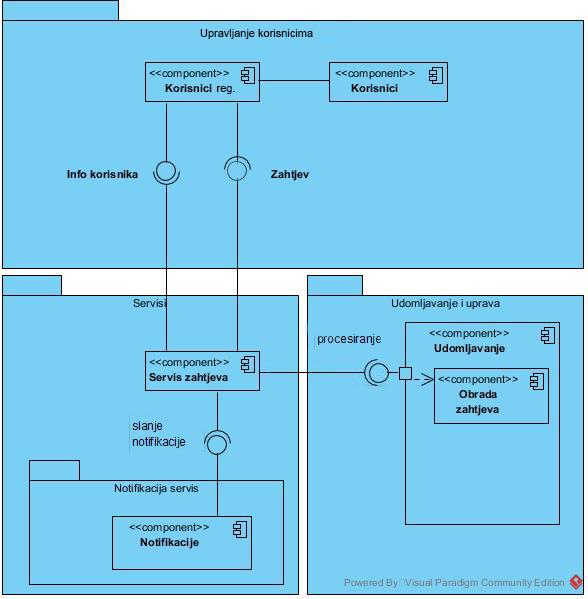
Dijagrami raspoređivanja

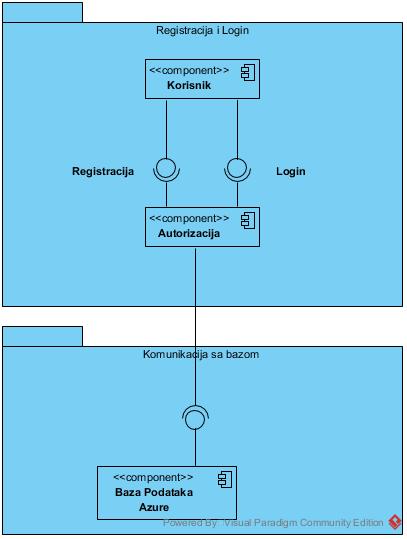


Radila: Ajna Zatrić

Za ostale dijagrame pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

Dijagrami komponenti

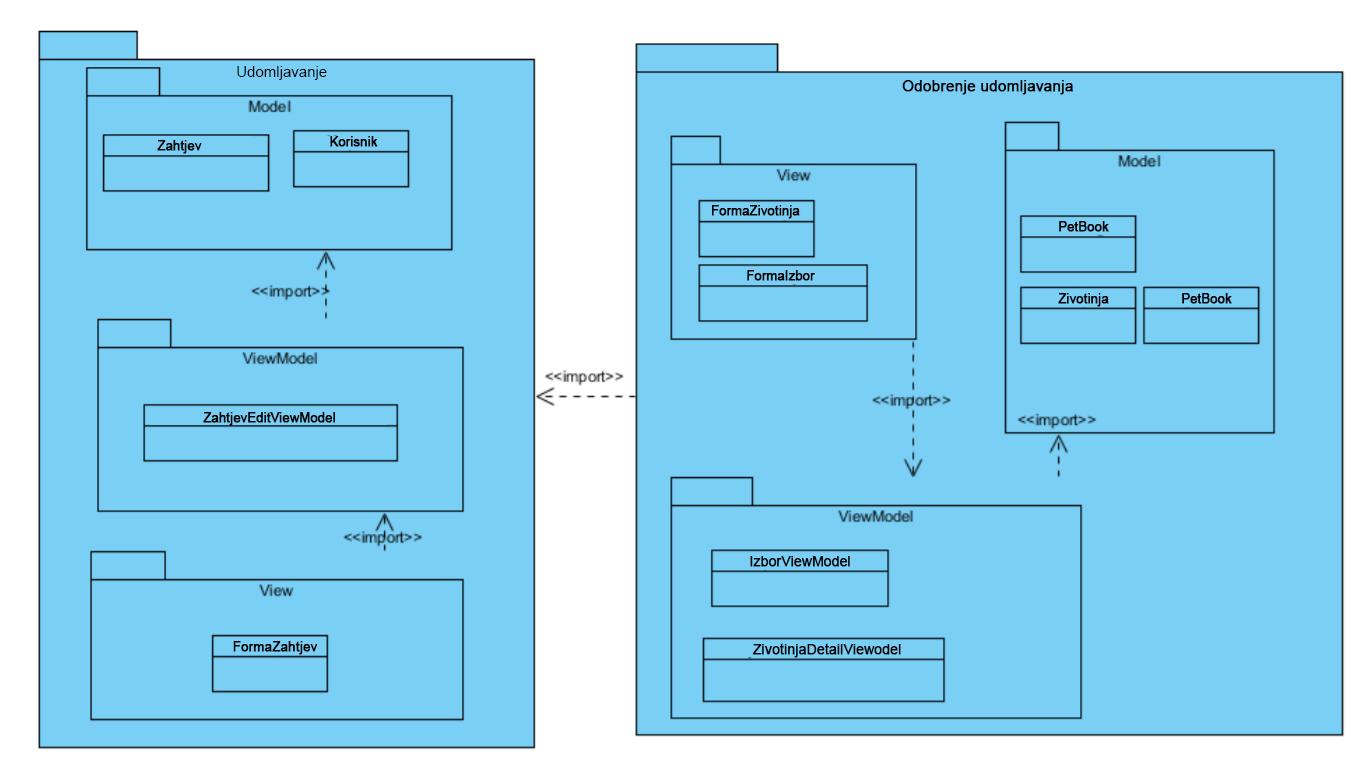


Komunikacija sa bazom

Radila: Ajna Zatrić

Za ostale dijagrame pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

Dijagrami paketa



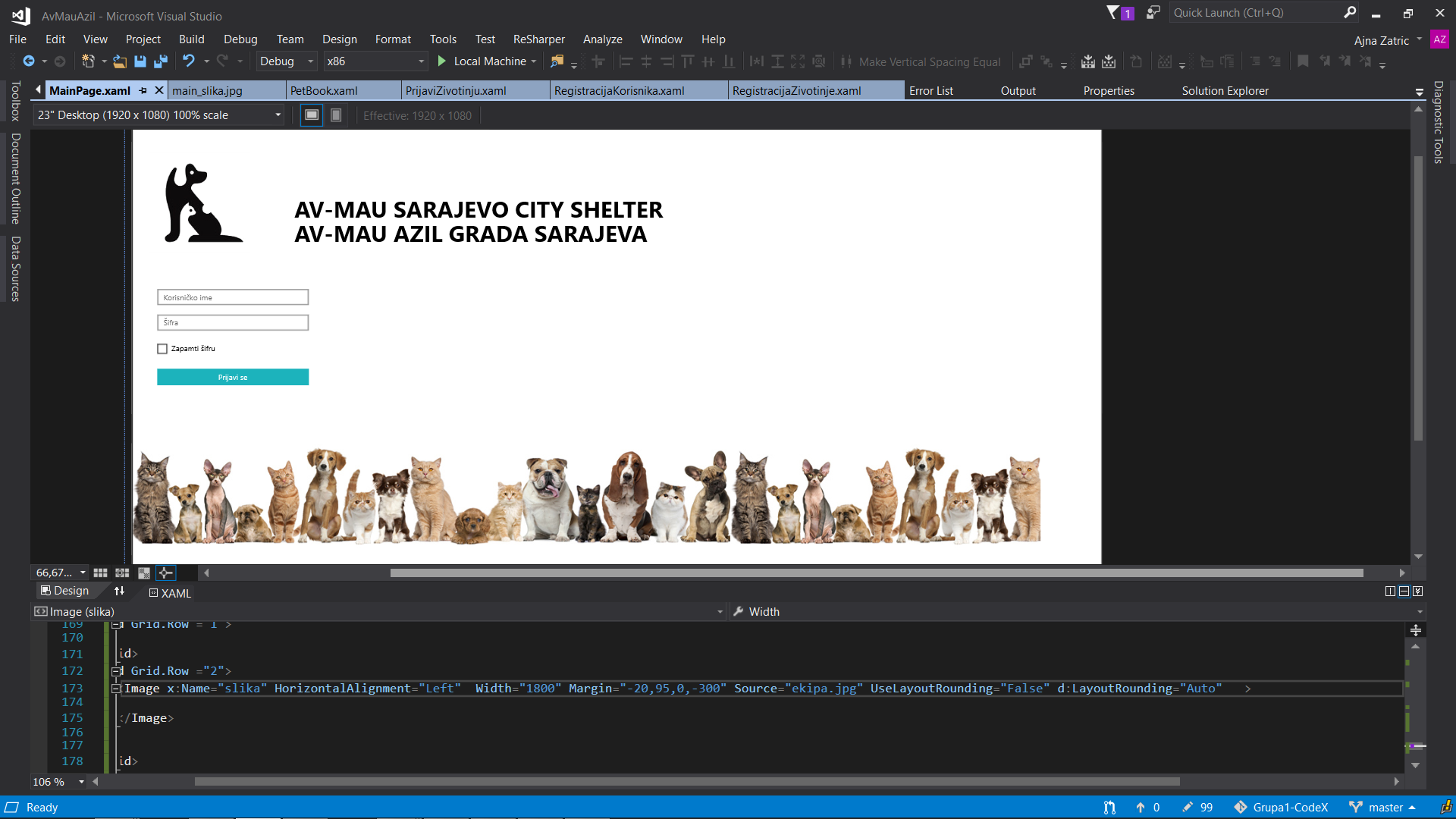
Radila: Ajna Zatrić

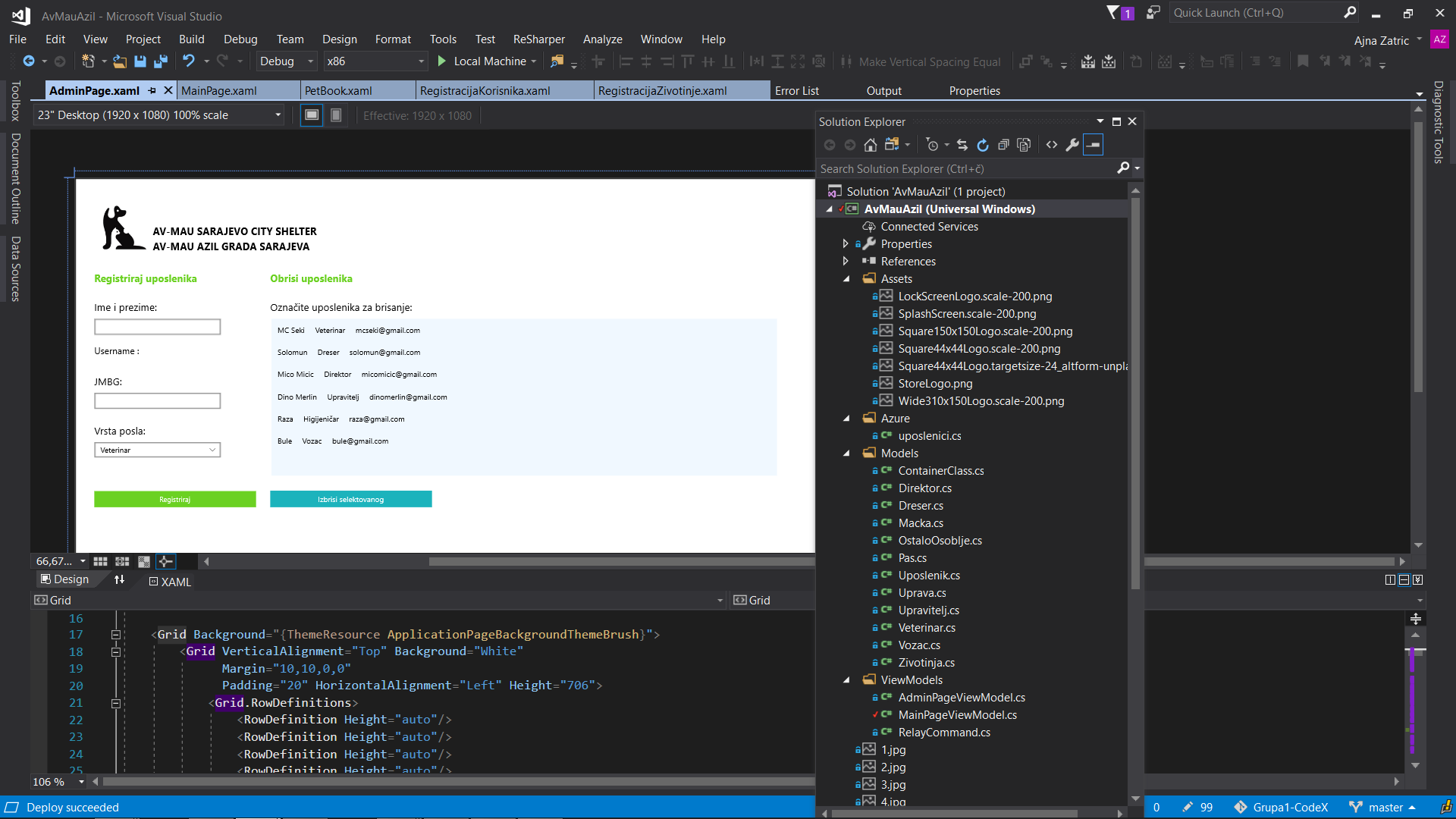
Za ostale dijagrame pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

|  |  |  |
| --- | --- | --- |
|  | Zadatak/Aktivnost |  |
| 1. | UWP-Dizajn i implementacija korisničkog Interfejsa (View) komponente za forme administracije. | Ajna Zatrić |
| 2. | UWP - Implementacija modela-funkcionalnosti aplikacije za forme administracije. (Arhitektura aplikacije je MVVM implementacija se vrši na osnovu urađenog modela) | Edin Avdić i Nadir Avdagić |
| 3. | UWP-Perzistencija podataka – rad sa bazom podataka na cloudu, spašavanje podataka za administraciju | Edin Avdić, Ajna Zatrić i Nadir Avdagić |
| 4. | Kreiranje ASP.NET projekta, arhitektura aplikacije je MVC, implementacija modela | Ajna Zatrić |
| 5. | Kreiranje kontrolera i pogleda View-a i povezivanje sa modelom | Ajna Zatrić, Edin Avdić i Nadir Avdagić |
| 6. | Rad sa Azure bazom podataka (model baze je urađen iz UWP aplikacije) | Ajna Zatrić i Edin Avdić |
| 7. | Rad sa vanjskim uređajima, specifičnim funkcionalnostima (Kamera) | Ajna Zatrić |
| 8. | Implementacija ASP.NET WEB API servisa | Edin Avdić |
| 9. | Poziv WEB API servisa iz ASP.NET MVC aplikacije | Edin Avdić |
| 10. | Deployment ASP.NET WEB API servisa na AZURE |  |
| 11. | Dokumentacija | Ajna Zatrić i Nadir Avdagić |
| 12. | Refaktoring i dizajn paterni | Ajna Zatrić |

1.UWP-Dizajn i implementacija korisničkog Interfejsa (View) komponente za forme administracije.

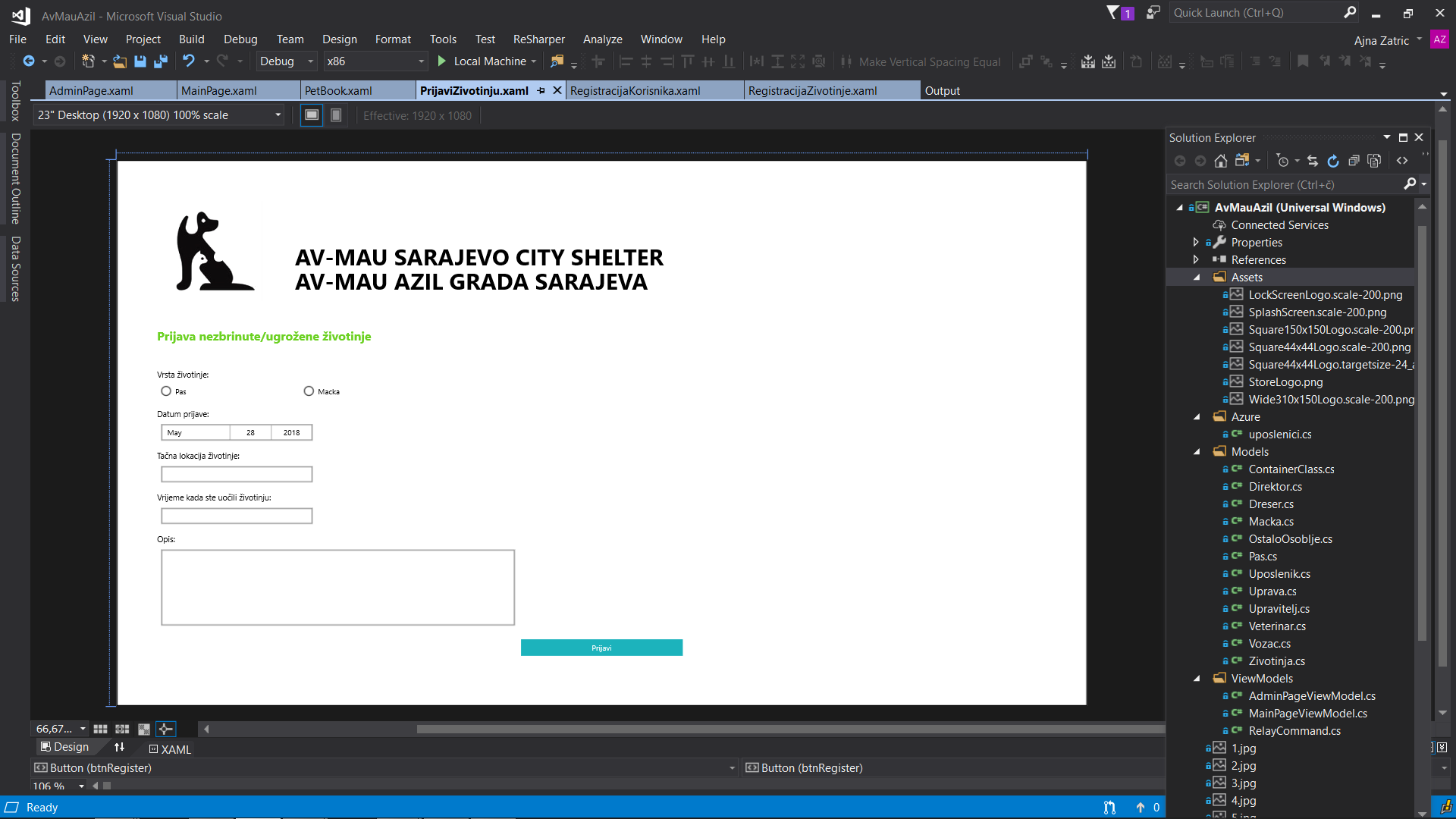
Korisnički interfejs - forme za administratora, kreirane u XAMLu. Za ostale forme, molimo Vas pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/Projekat)



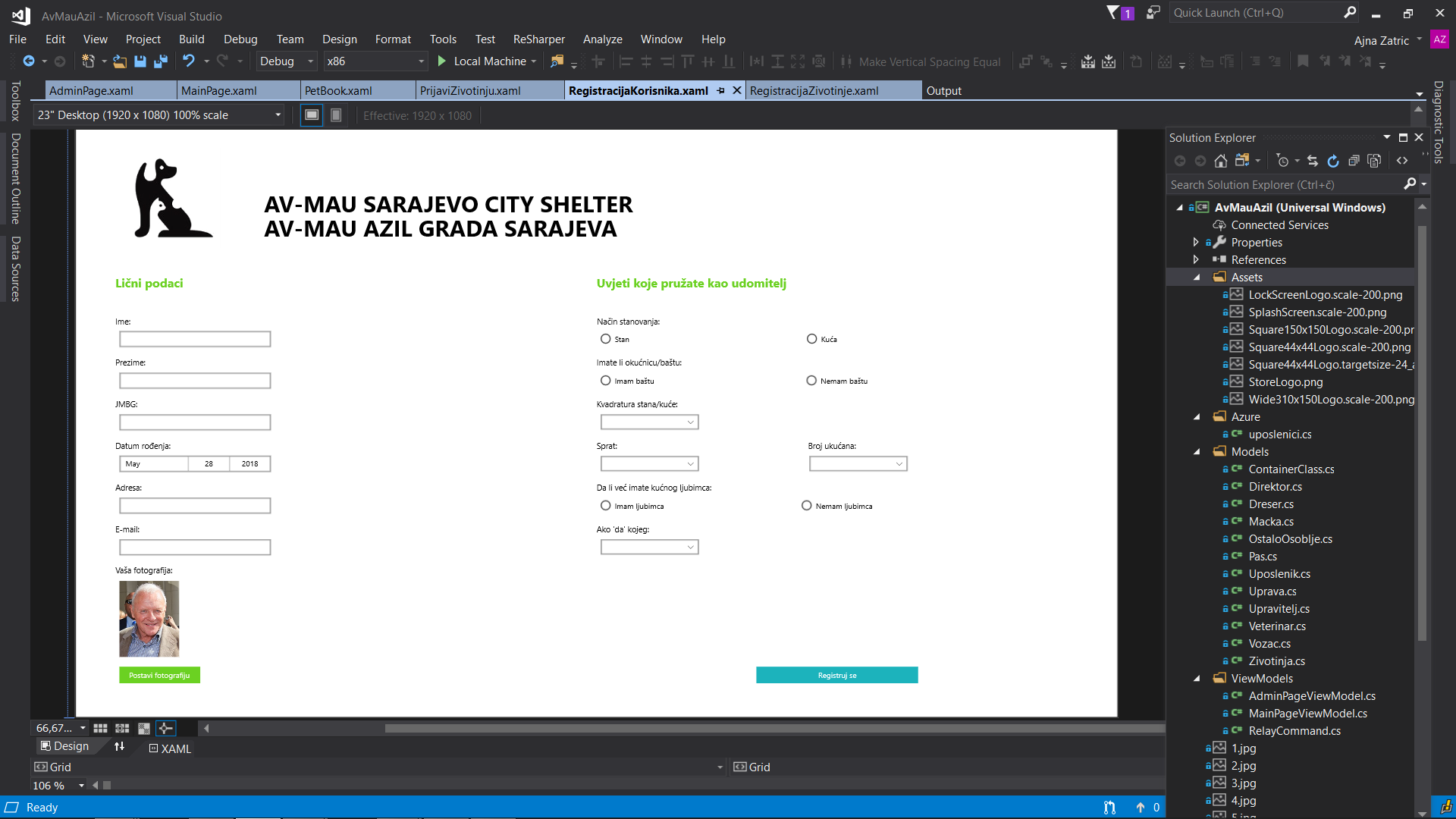


2.UWP - Implementacija modela-funkcionalnosti aplikacije za forme administracije. (Arhitektura aplikacije je MVVM implementacija se vrši na osnovu urađenog modela)

U Solution Explorer-u je vidljiva upotreba Model View View Model arhitekturalnog pattern-a za UWP dio aplikacije.

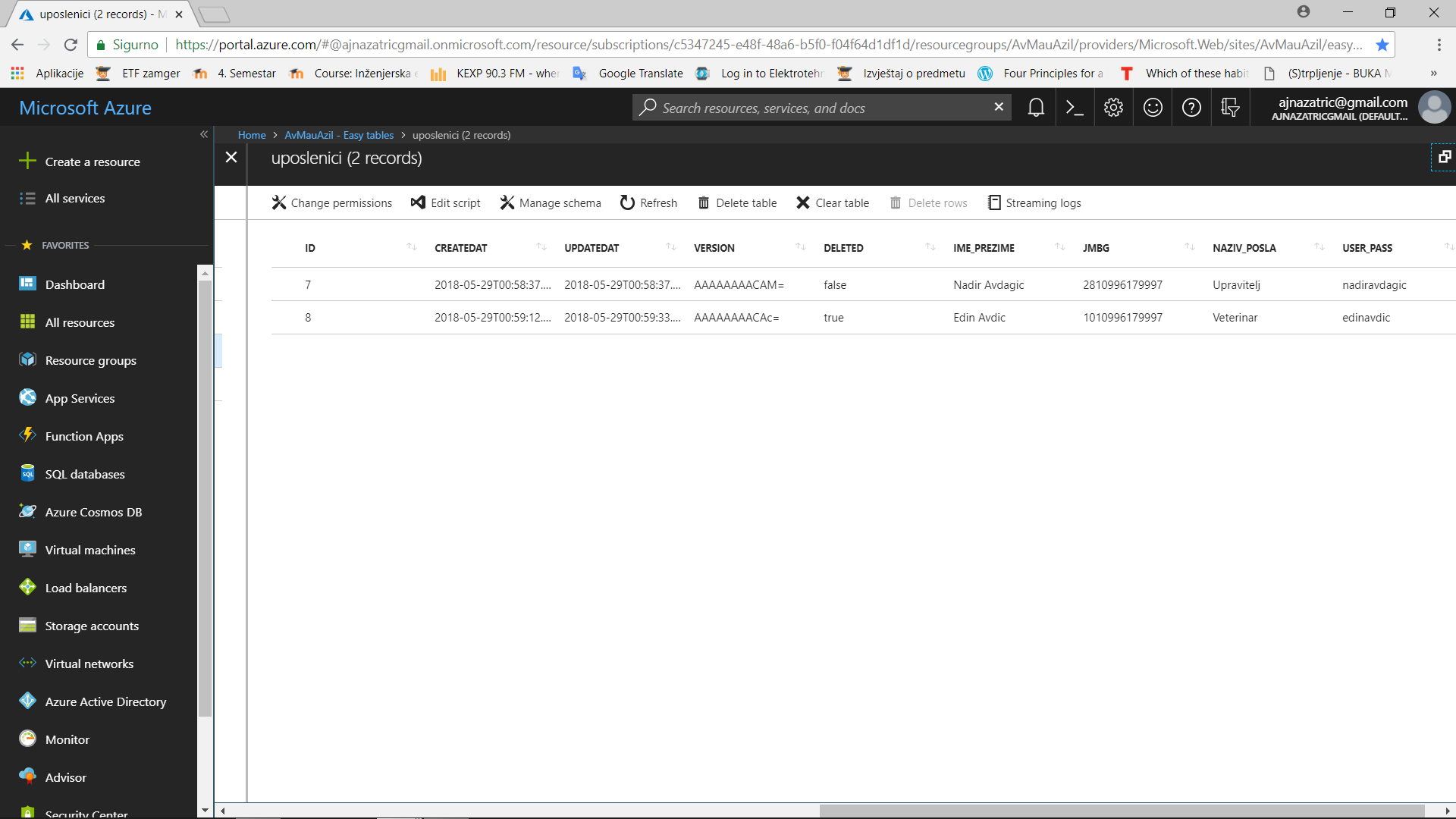


Prijava nezbrinute životinje



Registracija korisnika

3.UWP-Perzistencija podataka – rad sa bazom podataka na cloudu, spašavanje podataka za administraciju

Obezbijeđena je potpuna perzistencija sa Azure (Easy Tables) bazom podataka na cloudu. Uspješno dodavanje,brisanje i ažuriranje uposlenika od administratora, preko admin forme. 

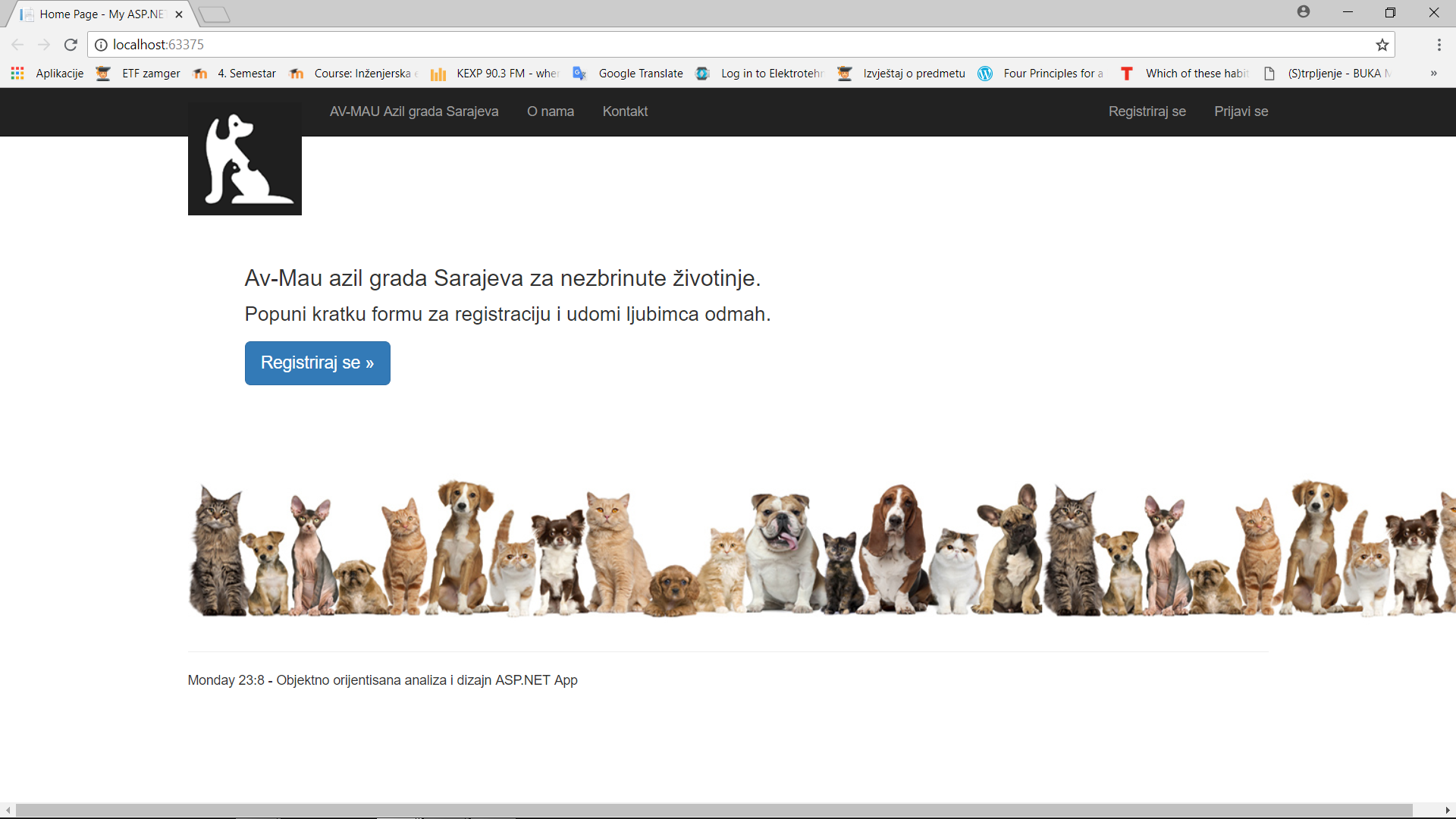
4. Kreiranje ASP.NET projekta, arhitektura aplikacije je MVC, implementacija modela

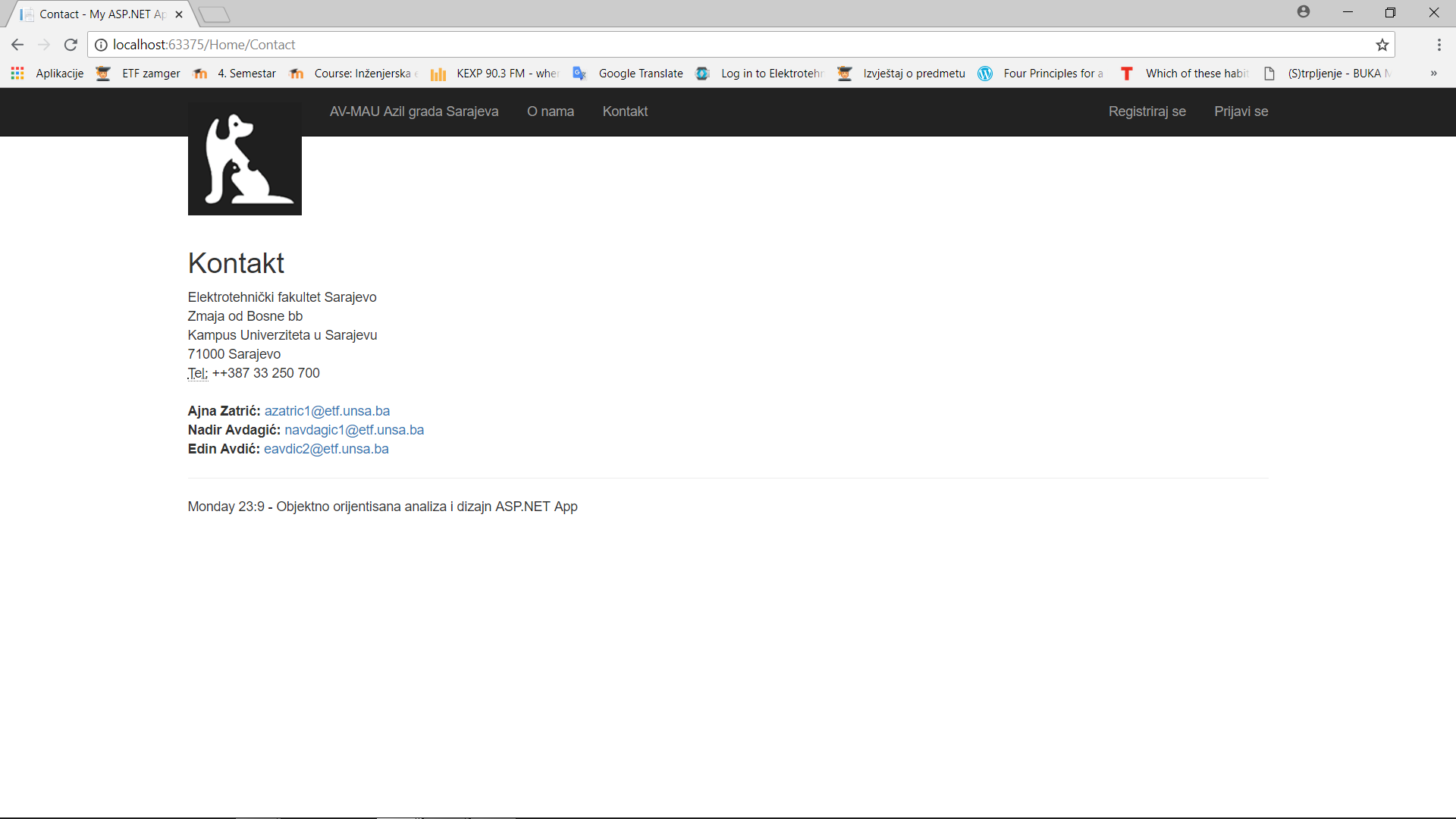
Kreiran je ASP.NET projekat, koristeću Model View Control arhitekturalni patern. Model je implementiran što se vidi na idućoj slici.



5. Kreiranje kontrolera i pogleda View-a i povezivanje sa modelom

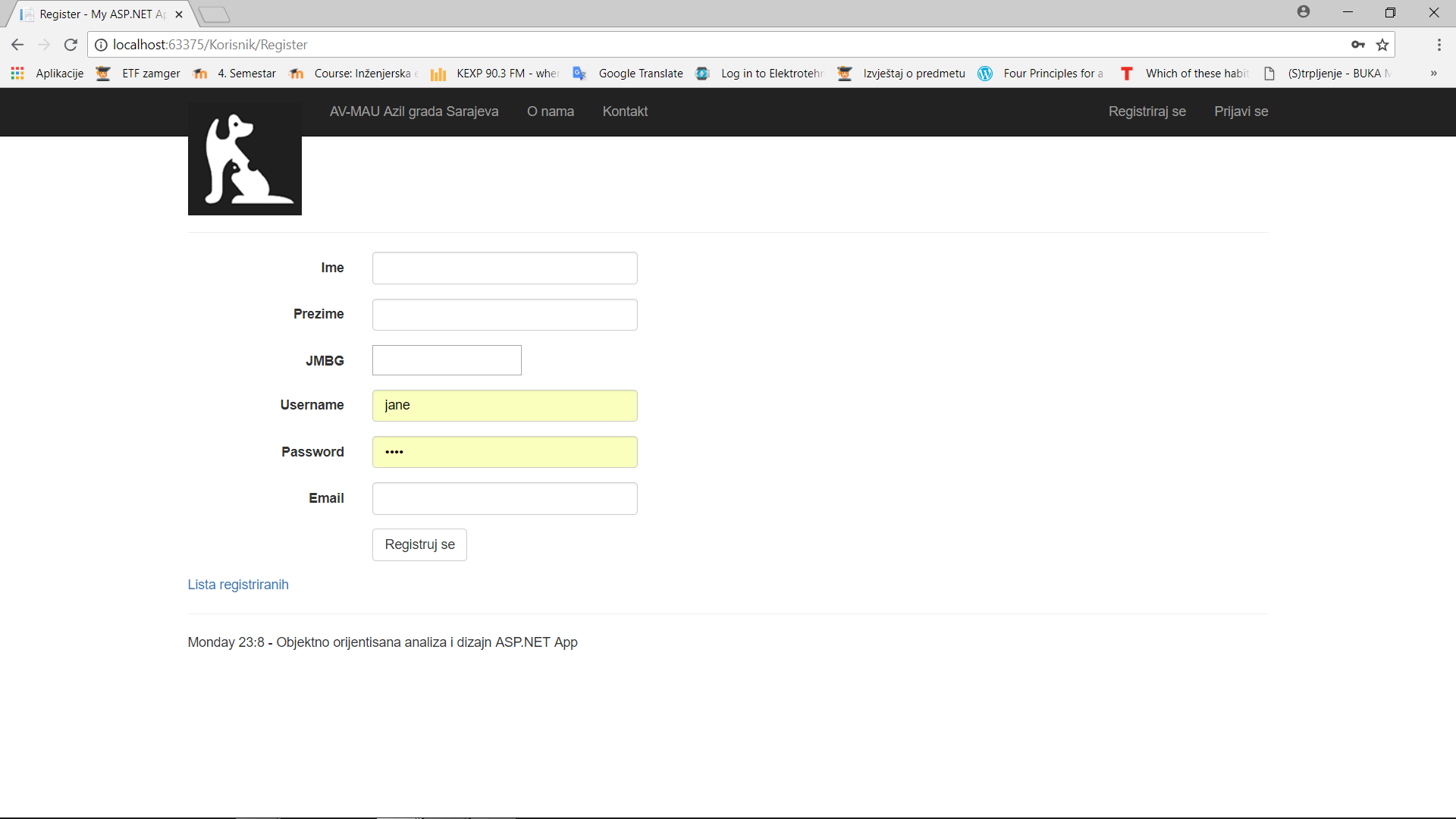
ASP.NET

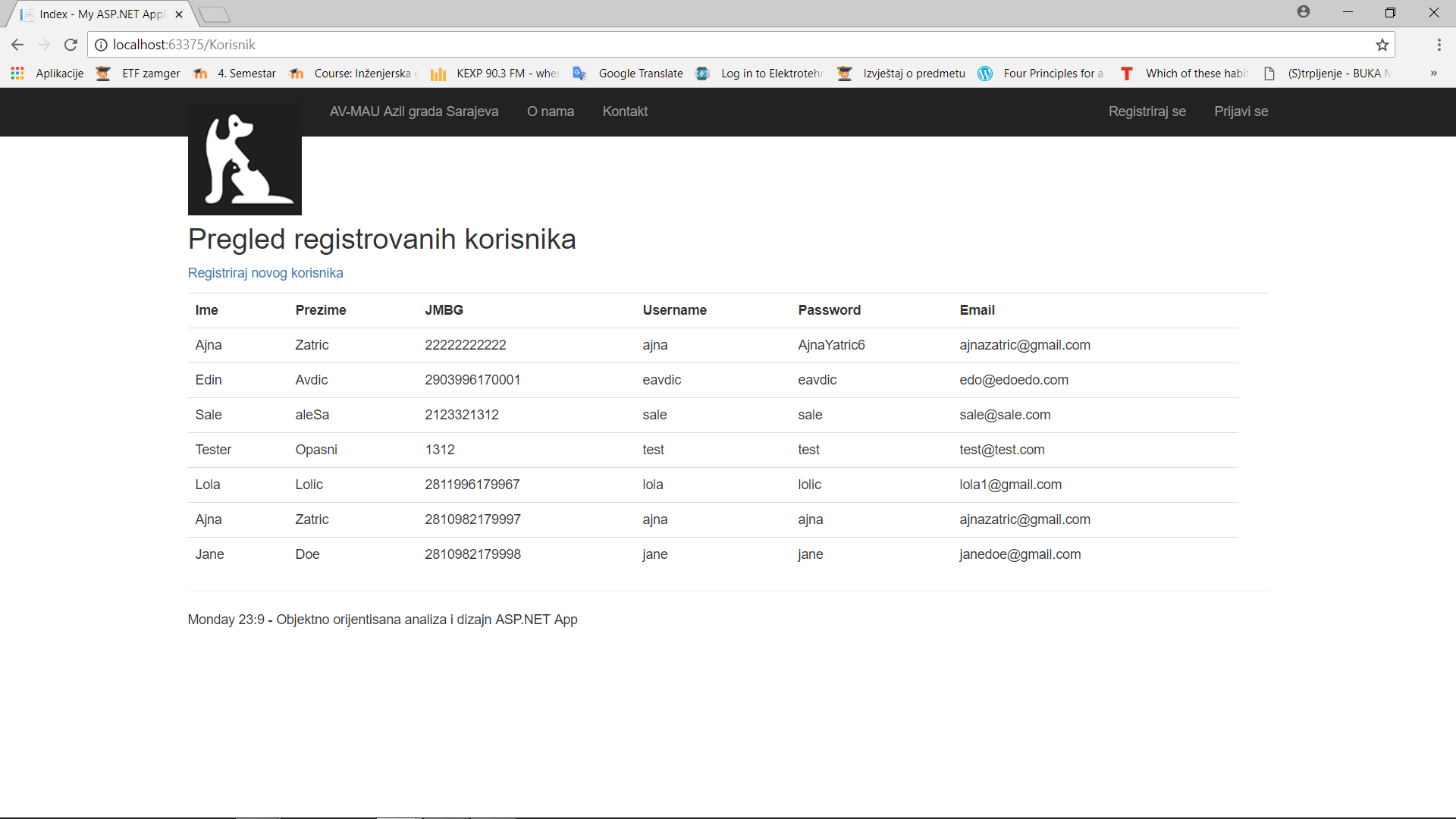




6.Rad sa bazom podataka

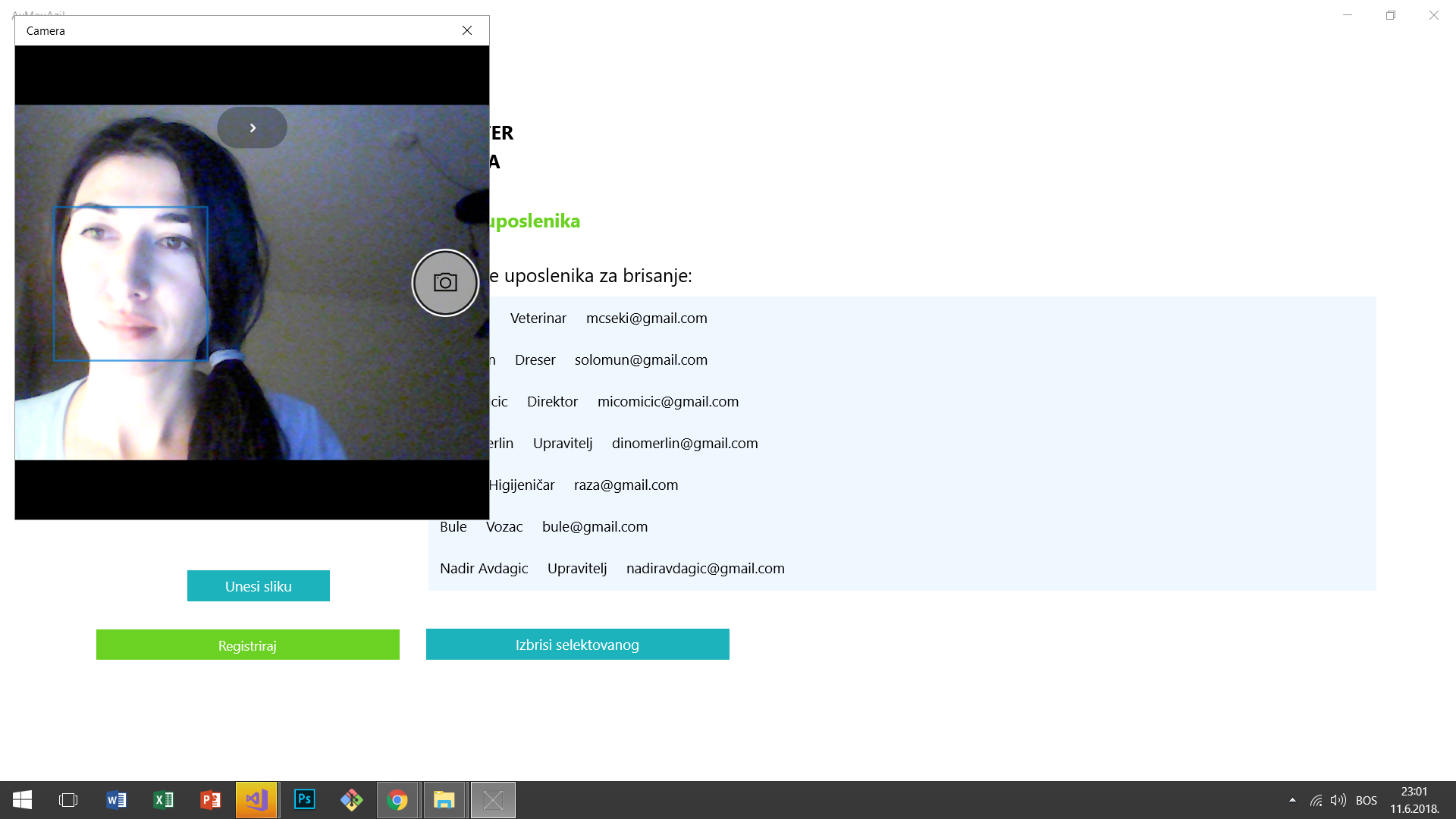
Omogućeno registriranje korisnika preko ASP.NEt dijela aplikacije kao što je vidljivo iz sljedećih slajdova.

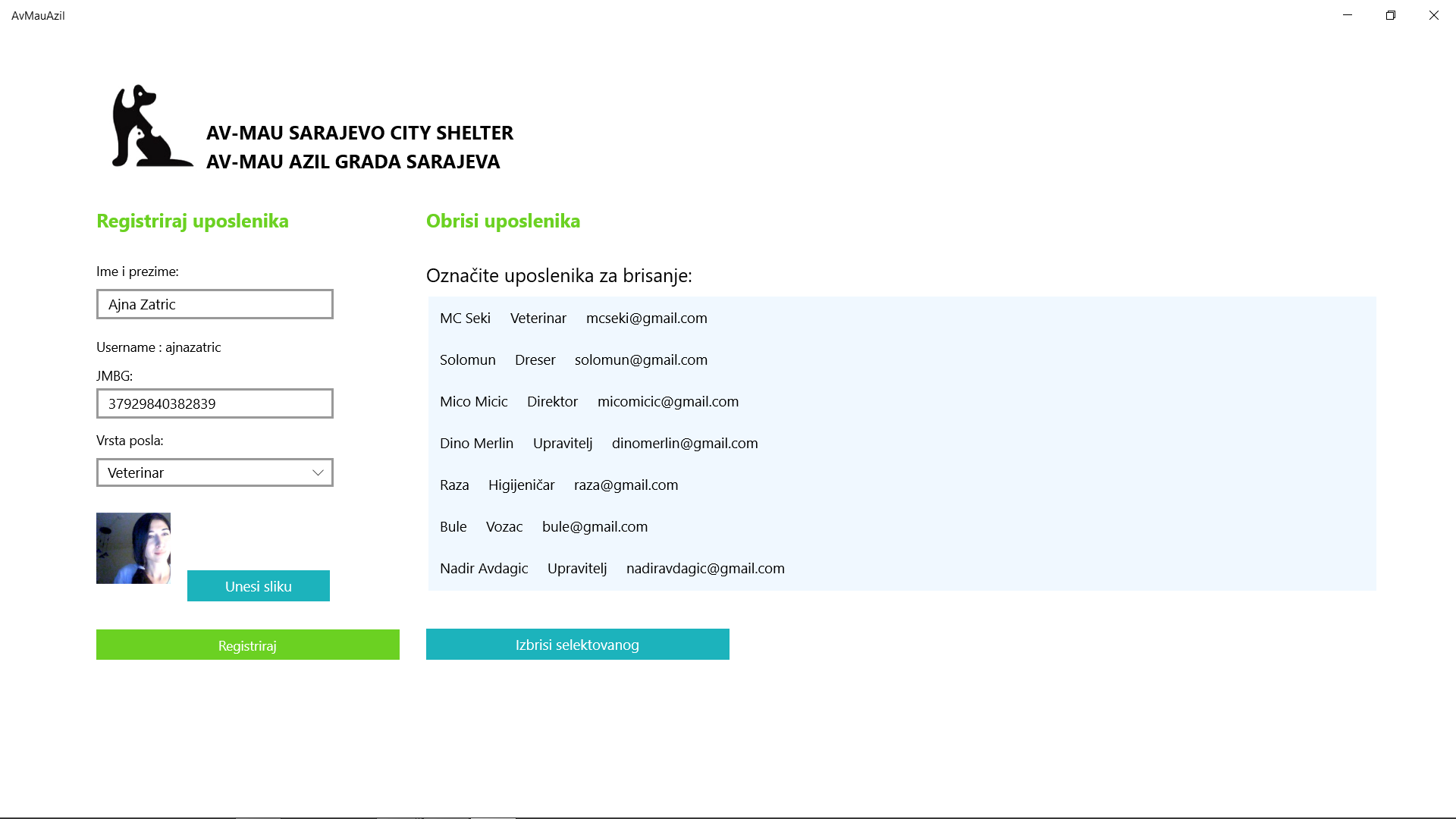




7. Rad sa vanjskim uređajima, specifičnim funkcionalnostima (Kamera)

Omogućen je automatizovan rad sa vanjskim uređajem - kamerom. Administrator prilikom evidentiranja novog uposlenika, može unijeti njegovu sliku klikom na dugme “Unos slike” čime se otvara kamera direktno iz aplikacije.

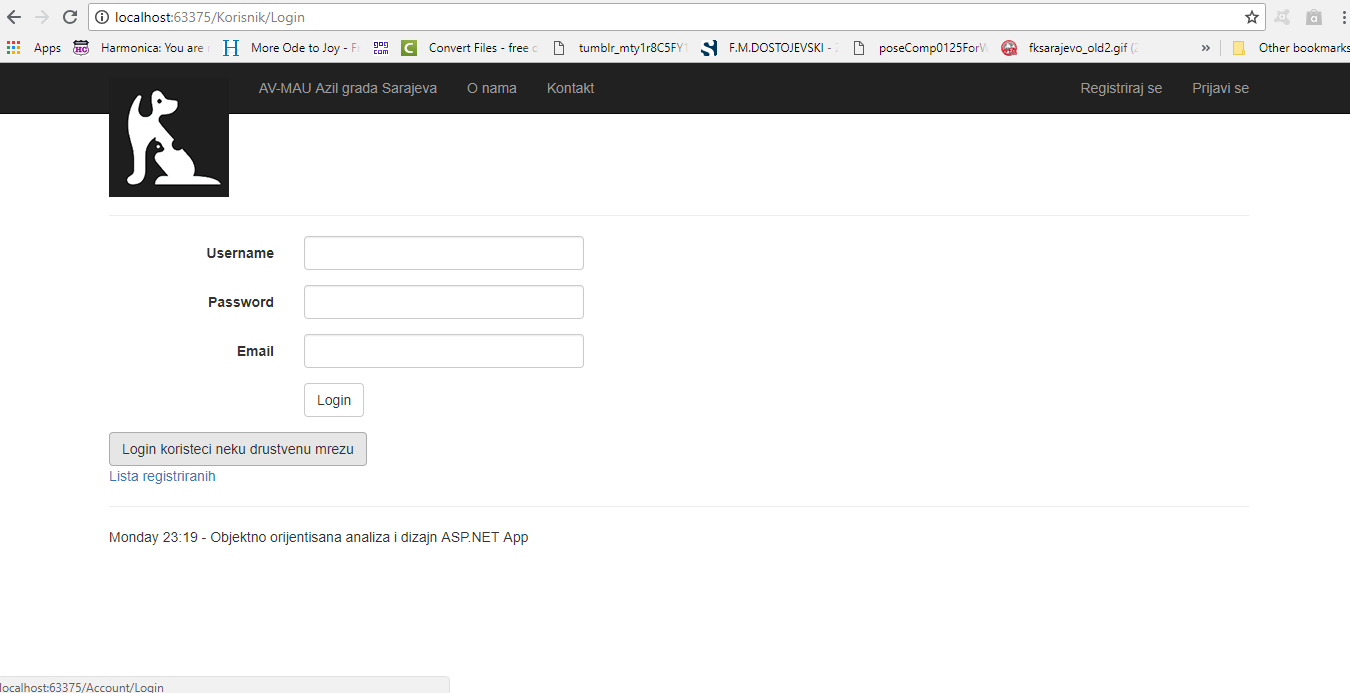


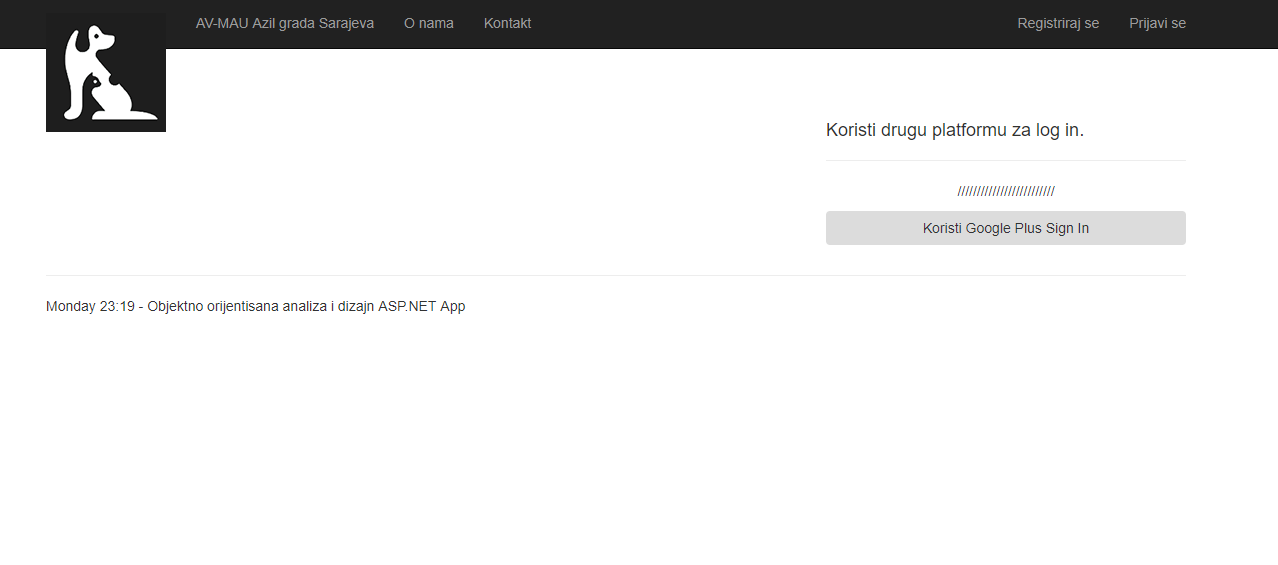


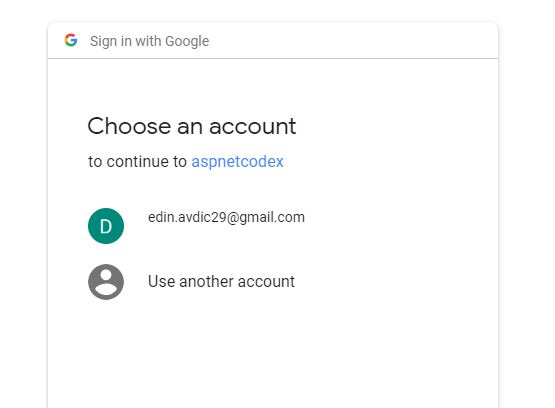
8.-9.-10.Implementacija ASP.NET WEB API servisa i

Poziv WEB API servisa iz ASP.NET MVC aplikacije

Omogućen je LogIn preko Google accounta. Kreirali smo projekat na google developers sajtu, pokrenuli njihov web API za eksterni Sign In koristeci Google+ account.  
Dobili api\_key i u istom projektu povezali URL aplikacije sa koje s mašine pokreće.







11. Dokumentacija

Dokumentacija urađena na osnovu aplikacije, prezentacije, projekta i projektnih zadataka.

12. Refaktoring i paterni

U dokumentaciji koja slijedi pokazana je primjena tehnika restruktuiranja koda na disciplinaran način. Kôd projekta “AvMau azil” je iterativno poboljšan na osnovu kataloga refaktoringa [Fowler, 1999] sa indikacijama kada je refaktoring potreban. Korišteno je 6 modifikacija na osnovu kataloga za refaktoring i djelimičnog refaktoringa primjenom dizajn paterna: Singleton (iz grupe kreacijskih paterna).

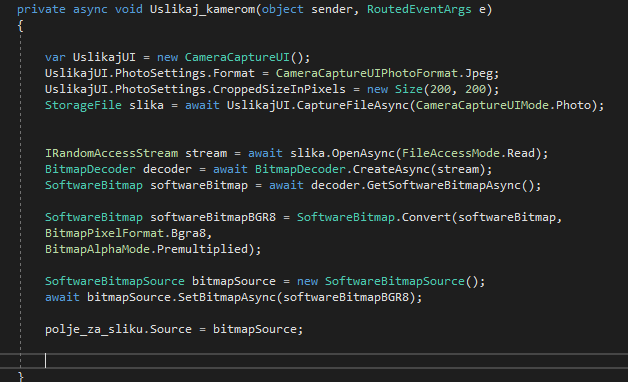
# Indikacija 1: Klase se veoma malo razlikuju pa nije potrebno nasljeđivanje. “Collapse hierarchy - collapse a superclass and subclass if their implementations are very similar.”

**Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Obzirom da je AvMau azil namijenjen isključivo dvijema vrstama životinja: mačkama i psima, i da smo primjetili da su klase “Pas” i “Mačka” izrazito slične u pogledu tipa podataka koje pohranjuju, odlučili smo ukinuti ovu hijerarhiju i osloniti se na boolean ili enum tip podataka u klasi Životinja (Enum VrstaZivotinje - {Pas, Macka}, odnosno Bolean VrstaZivotinje). Postignuto je pojednostavljenje koda.



# Indikacija 2: Algoritam je isuviše kompleksan. Zamijeniti kompleksni algoritam sa jednostavnim. “Substitute a simple algorithm for a complex algorithm.”

# **Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Do znaka upozorenja da smo došli, primjetivši da u datom dijelu koda imamo isuviše komentara. To je ukazivalo na još jedan znak za refaktoring: “Comments are used to explain difficult code (Komentari se koriste da objasne težak kod)”*.* U Admin.xaml.cs fajlu kod metode Slikaj\_kamerom bilo je više komentara. To je nagovijestilo da se funkcija treba zamijeniti jednostavnijom funkcijom istih funkcionalnosti, što smo i uradili. Ovom modifikacijom se postiglo pojednostavljivanje koda.

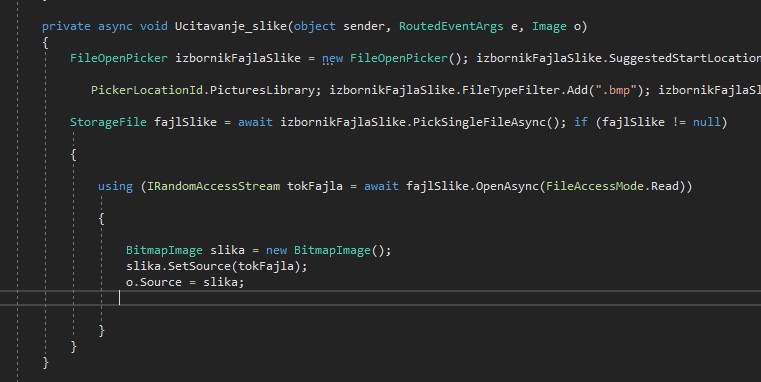


# Indikacija 3: Rutina ima loše - neopisno ime. Ako rutina ima loše ime, potrebno joj je promijeniti ime u definiciji i na mjestima poziva. “A routine has a poor name”

# **Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Analizom koda došli smo do zaključka da ima par metoda koje imaju nejasno i zbunjujuće ime u pogledu onoga što rade (na primjer, metoda “SlikajKamerom” prvobitno je imala dvosmislen naziv “Kamera”). Ime metode smo promijenili u definiciji i na mjestima poziva. Ova modifikacija doprinosi lakšem razumijevanju i komunikaciji u timu. Također, olakšana je dalja nadogradnja i čitljivost i razumljivost koda naročito među članovima tima koji neovisno rade na projektu (zajednički vokabukar među programerima).

# Indikacija 4: Kod je dupliciran.”Code is duplicated”

**Obrazloženje zašto je refaktoring urađen i šta se time postiglo :** Imali smo dvije odvojene funkcije za dodavanje slike za životinju i dodavanje slike za korisnika. Kreirana je jedna funkcija uz minimalne modifikacije, koja se potom poziva u oba slučaja. Funkciji smo dodali i dodatni parametar “Image Object”. Postignuto je poštovanje DRY principa i činjenice da ponavljanje koda sigurno predstavlja grešku u dizajnu.



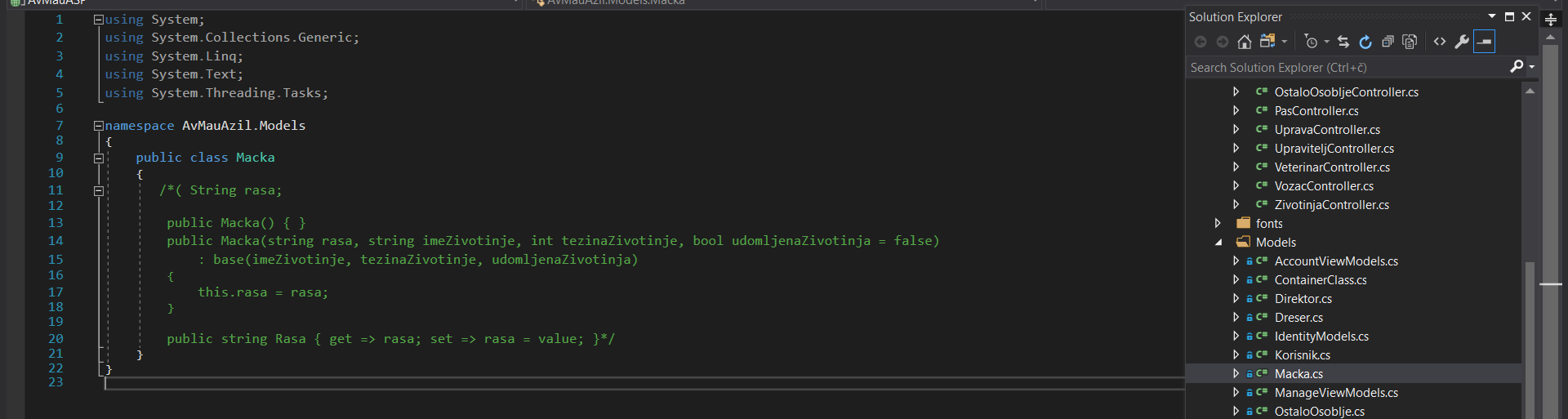
# 

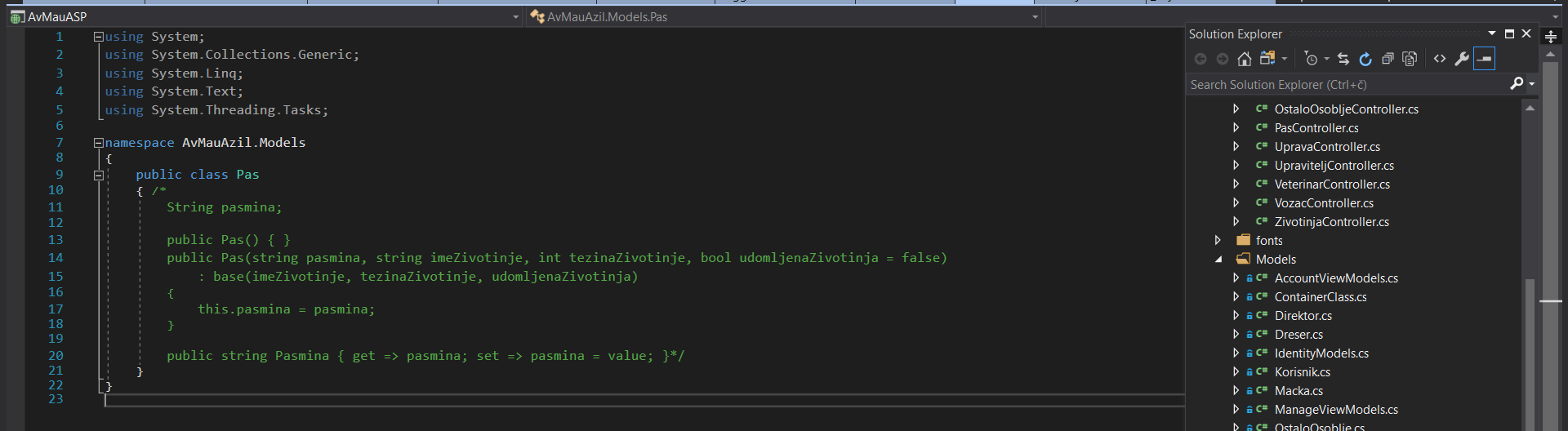
# Indikacija 5: Podaci su javni. “Data members are public”.

**Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Metodu ne koriste drugi modeli a vidljivost je bila public. Refactoring je primjenjen tako što je metoda proglašena privatnom. Ovakvi nepotrebno javni podaci predstavljaju nejasnoću između interfejsa i implementacije. Također, oni ugrožavaju enkapsulaciju i limitiraju buduću fleksibilnost realizirane aplikacije. Ovim refaktoringom se postigla upravo dobro razgraničenje interfejsa i implementacije i buduća fleksibilnost.

# Indikacija 6: Klasa ne radi puno. “ A class doesn’t do very much”

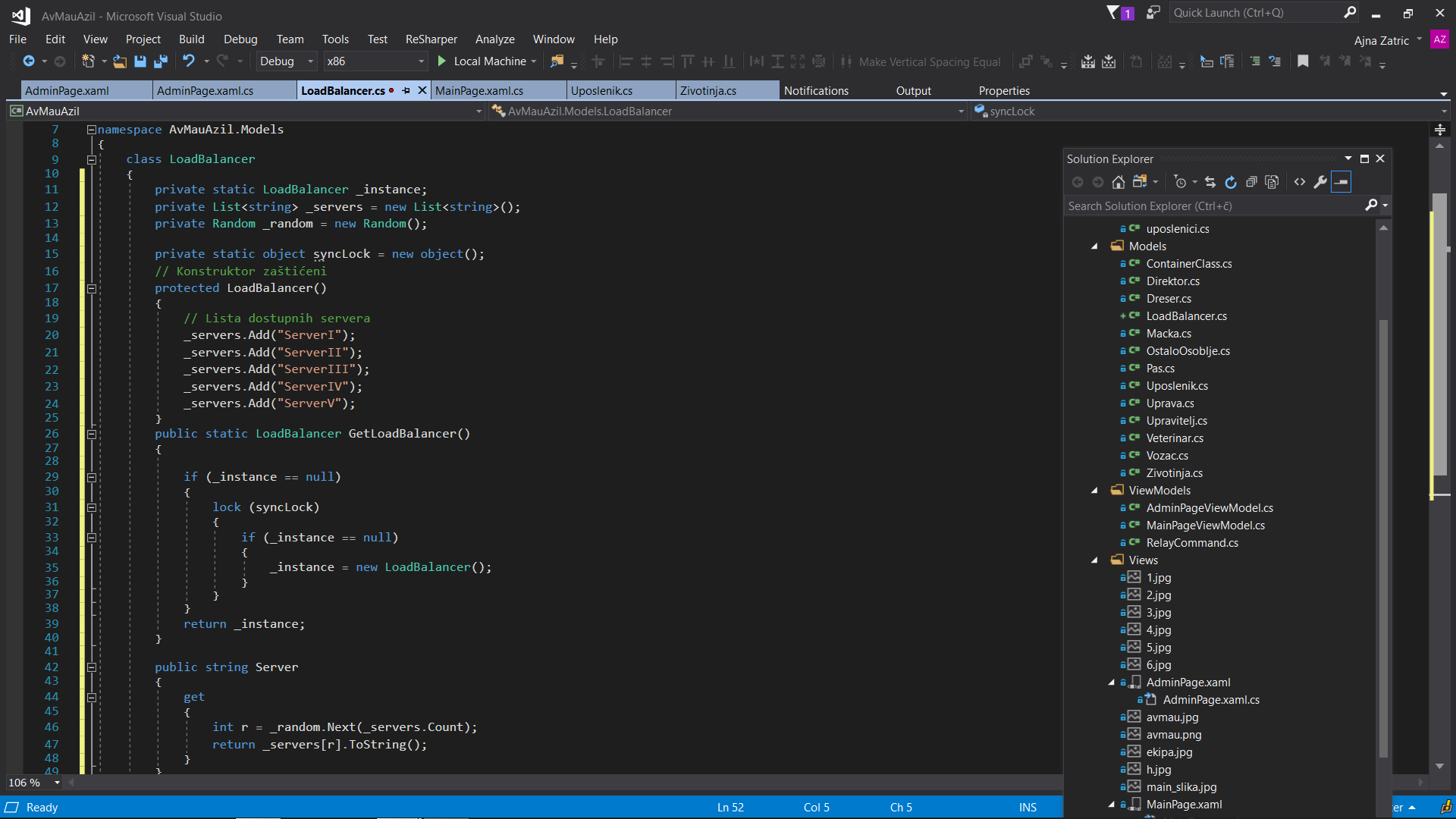
# **Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Ponekad rezultat refaktoringa koda jeste činjenica da klasa nema mnogo da radi. U našem slučaju smo ukidanjem hijerarhije klasa Životinja -> Pas i Životinja->Mačka primjetili da su sve odgovornosti klasa Pas i Mačka dodijeljene klasi “Životinja” , pa smo ove klase eliminirali u potpunosti. (Ostavljene su u projektu i zakomentarisane kako bi se video uzrok refaktoringa).





**Refaktoring u kodu primjenom dizajn paterna**

Primer realne upotrebe Singleton paterna u našem projektu može biti u slučaju LoadBalancing objekta (Load balancing predstavlja distribuciju zahtjeva na više servera u cilju postizanja što boljeg vremena odziva). Samo jedna instanca objekta koji vrši Load balancing može postojati, jer serveri mogu dinamički da postaju dostupni ili nedostupni, pa svaki zahtev mora prolaziti kroz jedan objekat koji poseduje znanje o stanju servera.



Primjer implementacije Singleton patterna u kodu.