

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.

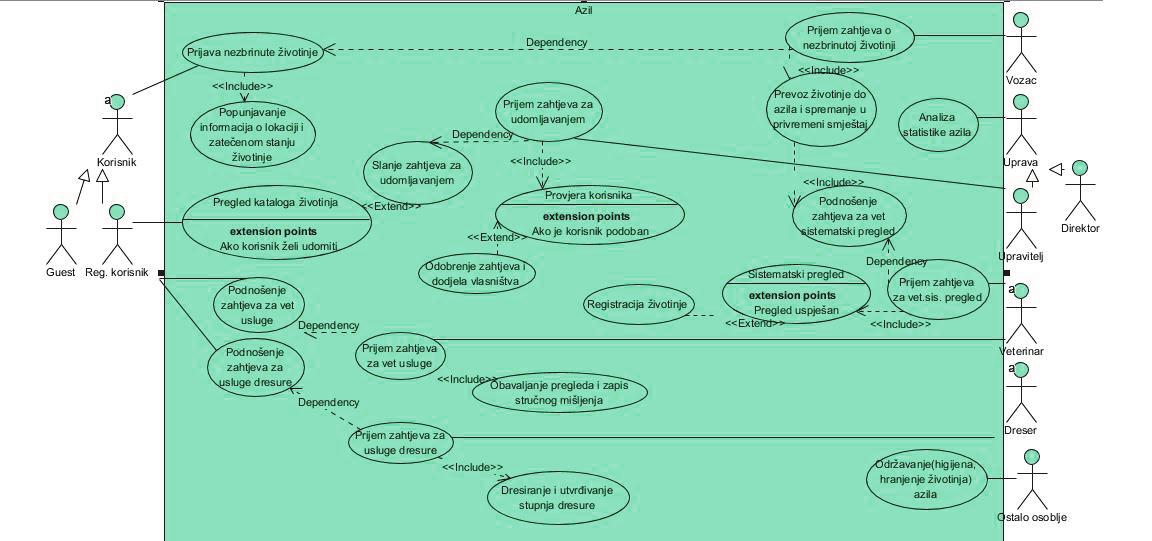
Program projektnih zadataka na predmetu OOAD 2017/2018:

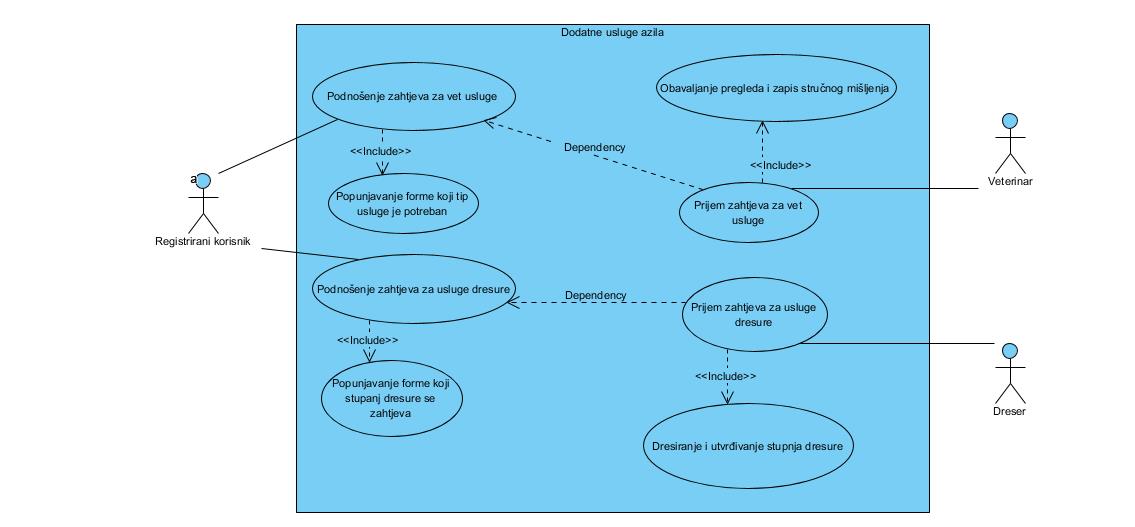
PZ1 - Prijedlog teme . PZ2 - Use case dijagram i scenariji . PZ3 Mokapovi i dijagrami aktivnosti . PZ4 - Korisnički interfejs aplikacije (u XAMLu za admina) i dijagram klasa . P5 - Re-dizajnirati dijagram klasa tako da prati MVVM pattern i popraviti uočene prekšene SOLID principe tako da budu ispoštovani . 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

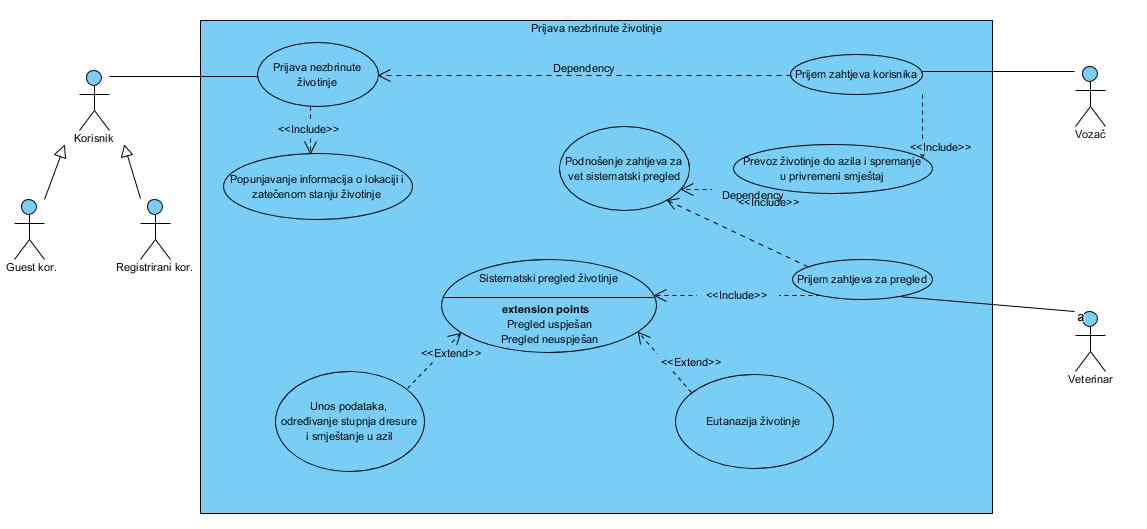
Prijedlog teme

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

Use case dijagram i scenariji





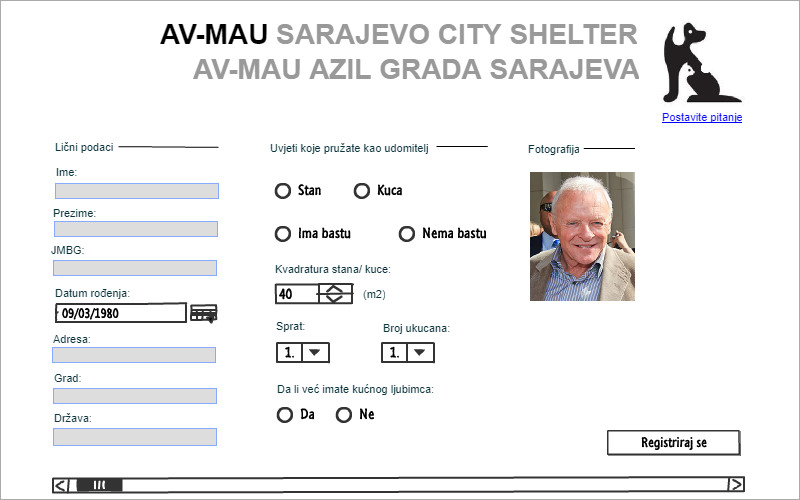
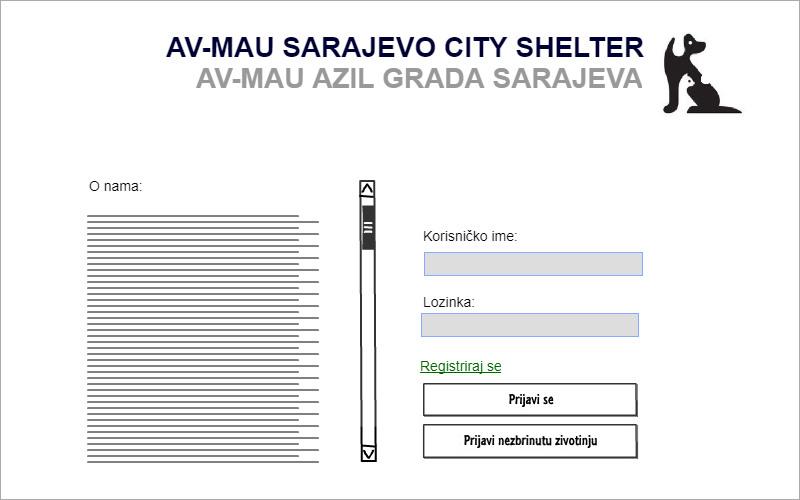


Sve Use case dijagrame projekta i scenarije - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

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



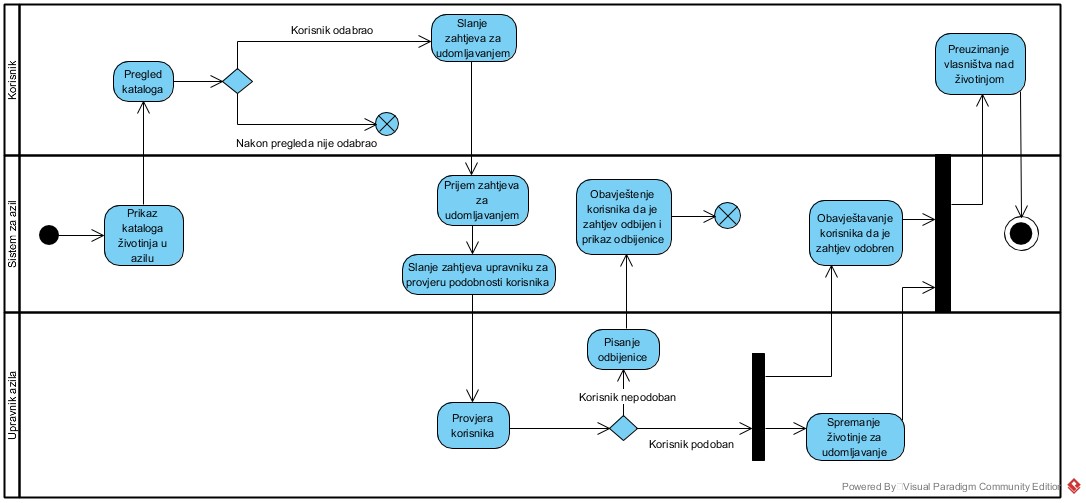
Pretraga PetBook-a filterom

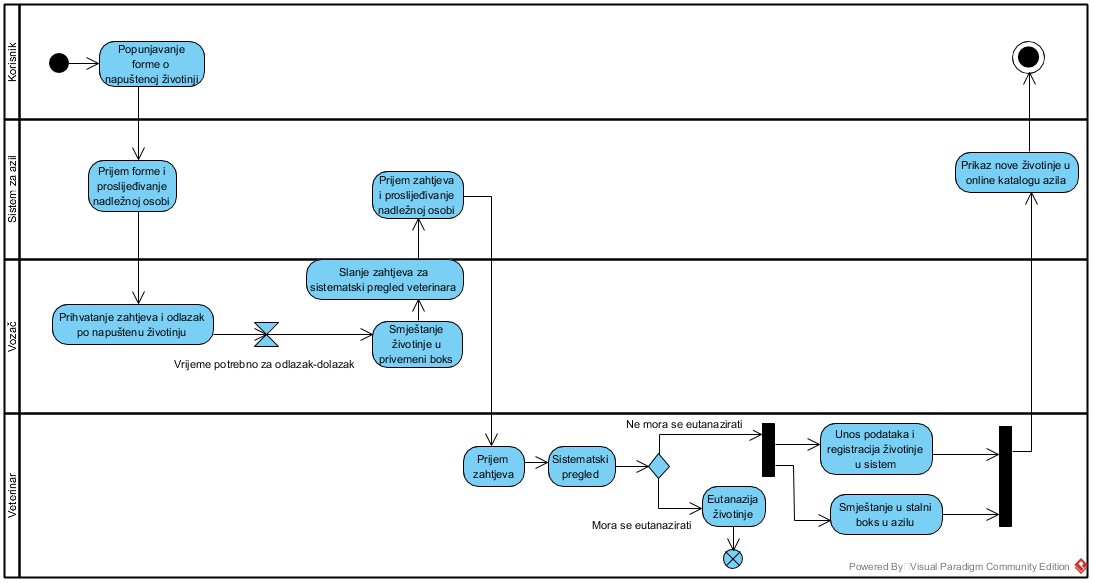


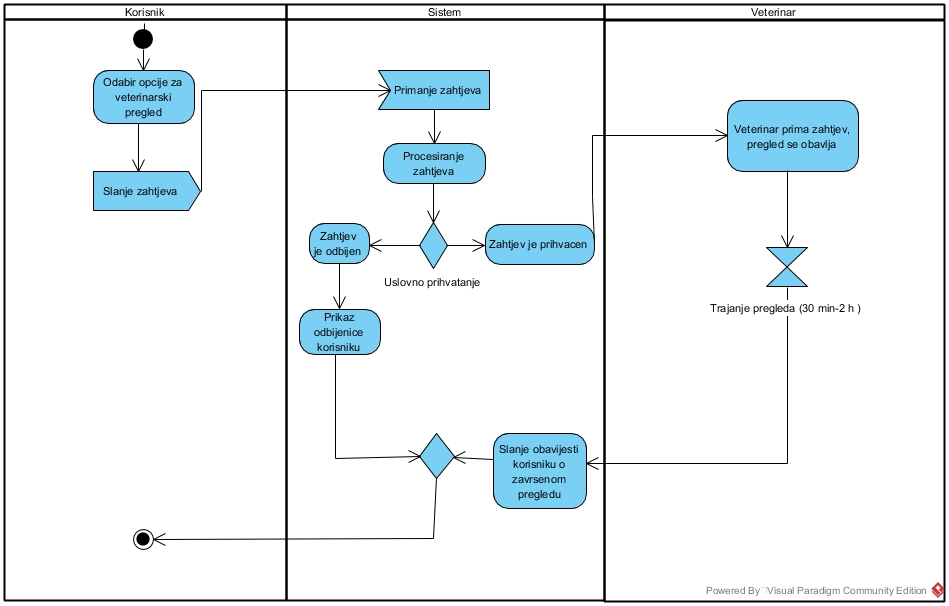
Prototipove svih formi - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/Prototip)

Radila: Ajna Zatrić

Dijagrami aktivnosti

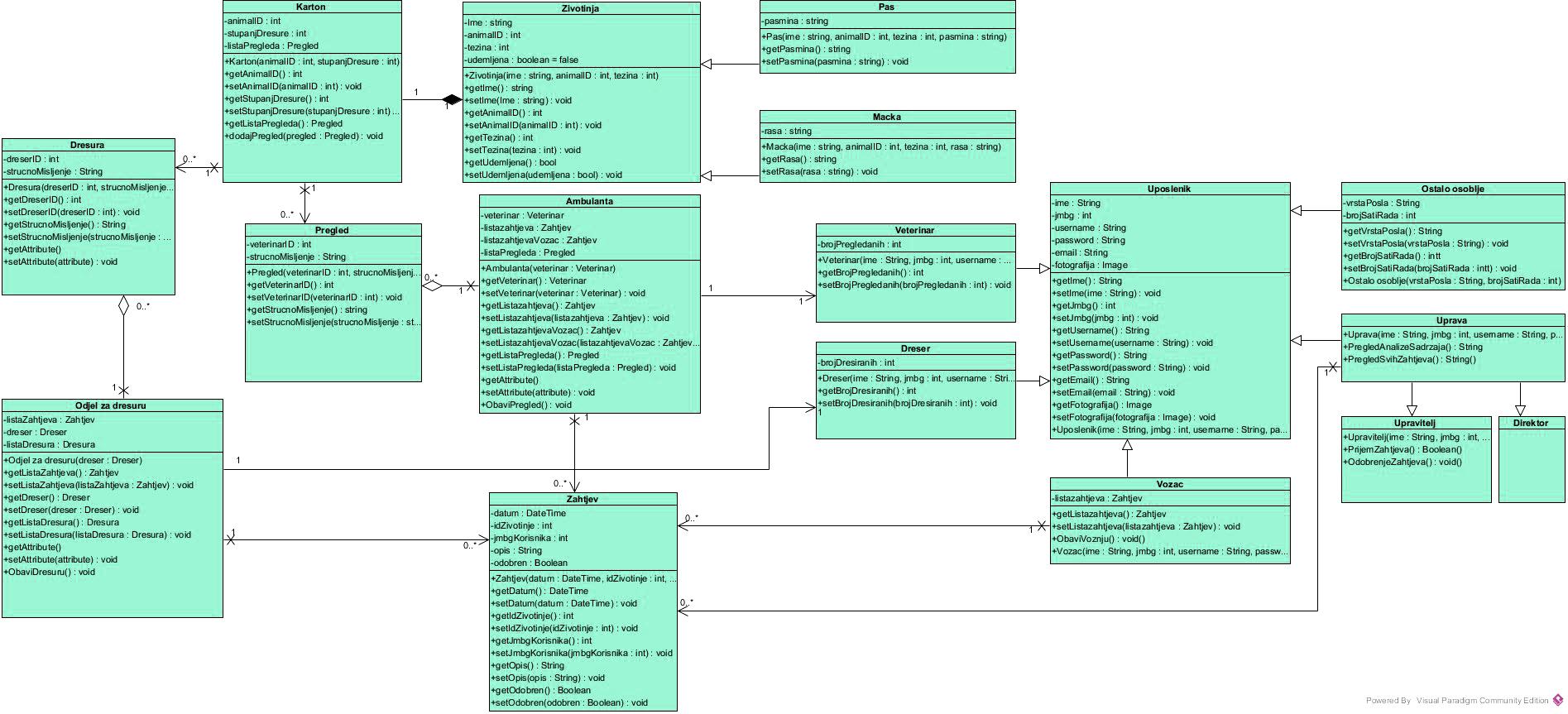




 Sve dijagrame aktivnosti - možete naći [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/DijagramiAktivnosti).

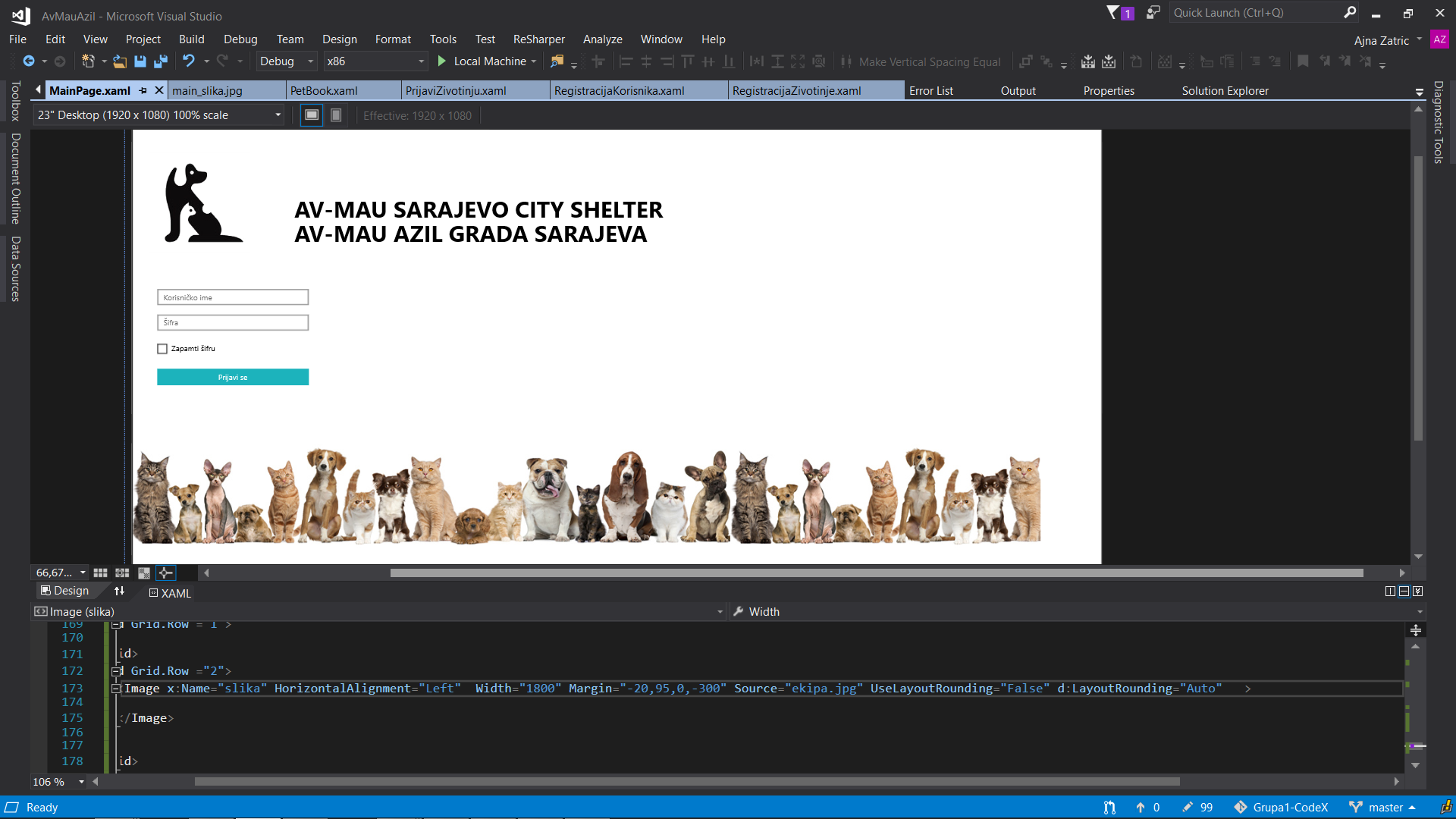
Radili: Nadir Avdagić i Edin Avdić

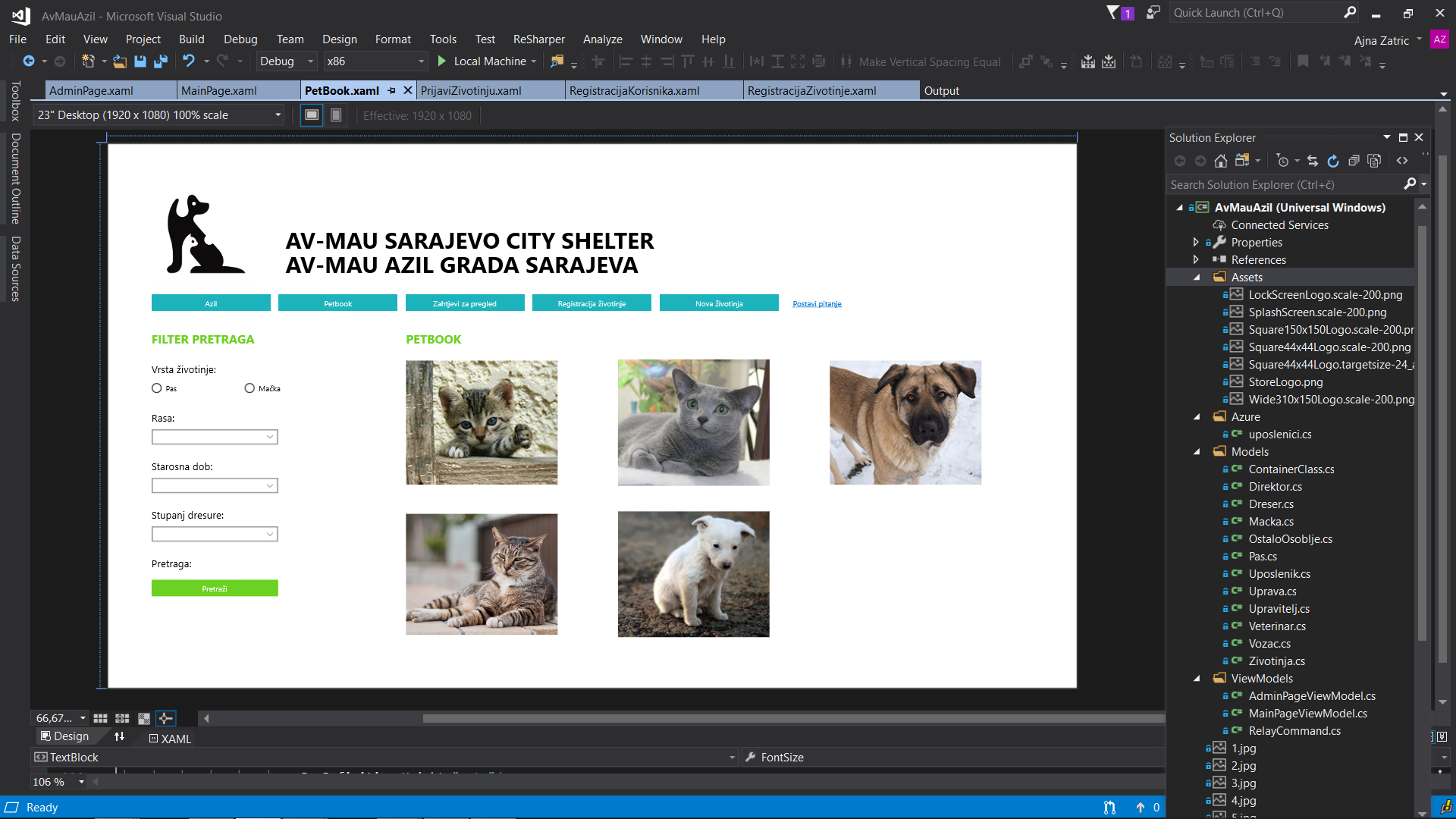
Dijagram klasa

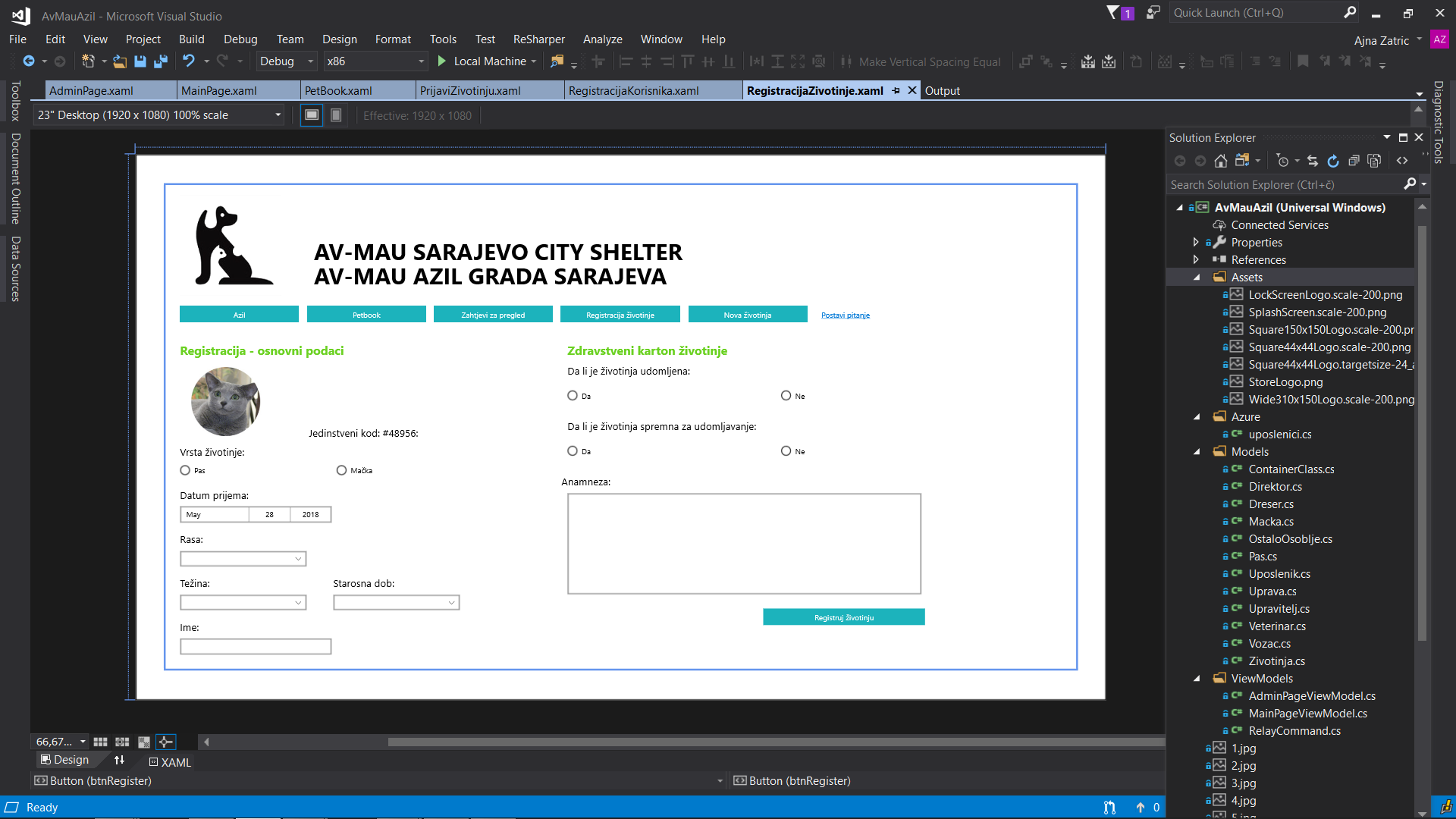


Radili: Nadir Avdagić i Edin Avdić

UWP Programiranje formi u XAML-u







UWP forme u XAMLu

Radila: Ajna Zatrić

Dijagrami sekvenci

Dijagram sekvenci za proces udomljavanja

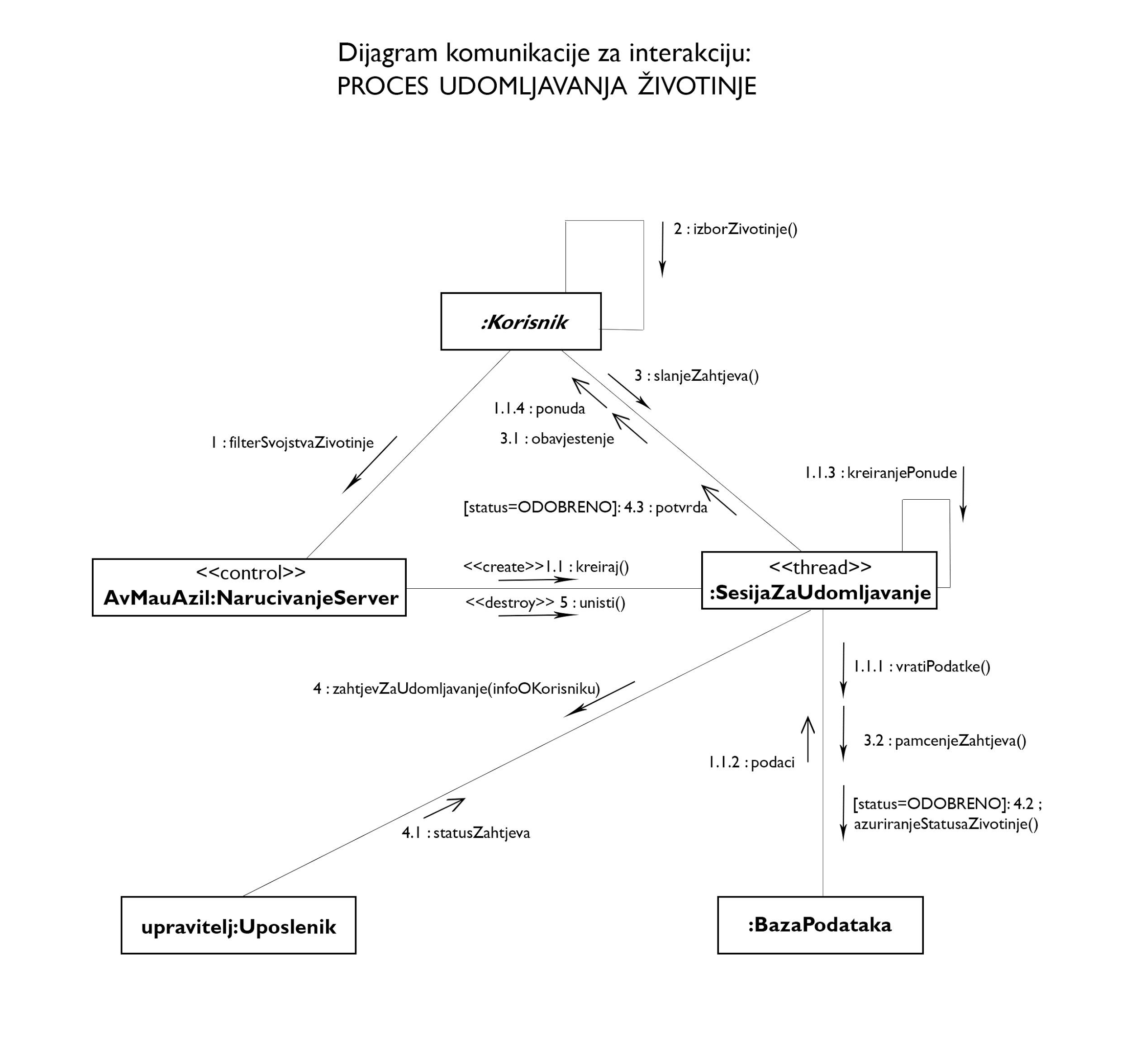


Dijagram sekvenci za pregled životinje



Radili: Nadir Avdagić i Ajna Zatrić

Dijagrami komunikacije



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

Radila: Ajna Zatrić

Dizajn paterni - analiza 10 dizajn paterna i mogućnosti njihove implementacije i primjene u projektu “AvMau Azil”

**Composite patern**

Composite patern omogućava grupama objekata koje predstavljaju hijerarhiju da budu tretirane na isti način kao jedna instanca objekta. Jednostavnije rečeno, funkcija ovog paterna je da se jedna komponenta i druga komponenta mogu tretirati na isti način. Tipične operacija nad komponentama uključuju add, remove, display, find i group. Aplikacije za puštanje muzike (iTunes, Winamp..) ili aplikacije za kreiranje albuma sa slikama (npr. Flickr) koriste ovaj patern. Stavke se smještaju u velike liste, koje se zatim na određeni način struktuiraju.

- **PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Ako posmatramo PetPhoto odjeljak u našoj aplikaciji, vidjećemo da je u planu da postoje različiti načini na koje možemo prikazivati slike koje su u njega unijete: hronološki ili na osnovu imena pod kojim su unete (npr. Ljeto2918uAzilu). Jedna fotografija se može pojaviti u okviru više albuma. Kreiranje albuma stvara kompozitni objekat, koji ne uključuje stvarno kopiranje fotografija na više lokacija.

**Iterator patern**

Iterator patern omogućuje kretanje kroz kolekciju na formalizovan način. Konkretnije, omogućava sekvencijalni pristup elementima u kolekciji a da se pri tome ne mora znati njena struktura. Pored toga, patern omogućava filtriranje elemenata na različite načine. Uz koncept iternatora definira se i koncept enumeratora koji su odgovorni za dobavljanje sljedećeg elementa u sekvenci koju definiše određeni kriterij. Za takvu sekvencu se kaže da je „enumerable“.Primjeri mogu biti sljedeći cijeli broj, ili sljedeći naručen proizvod sortiran po datumu.

- **PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Iterator patern će naći odličnu primjenu u našem projektu za filtriranje Petbook-a, odnosno za prolazak kroz kolekcijuživotinja prilikom traženje životinje za udomljavanje od strane korisnika.

**Memento patern**

Memento patern omogućava vraćanje objekta u prethodno stanje, bez narušavanja principa enkapsulacije i privatnosti objekta. Primjer primjene memento paterna su „checkpoints“ sačuvani u video igrici gdje je igrač u mogućnosti da se vrati na nivoe koje je već osvojio. Drugi primjer je un-do operacija u aplikacijama koje procesiraju riječi. Memento koristi Originator klasu, koja je objekat koji će biti sačuvan i „restauriran“ kasnije. Memento klasa čuva „historijsku“ informaciju Originatora u nekom trenutku.

- **PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Memento patern će biti koristan da se obezbijedi “undo” mehanizam u našoj aplikaciji, kada interno stanje objekta treba povratiti ili obnoviti u nekom kasnijem stadijumu. Primjer je klasa Korisnika kojoj možemo sačuvati njene originalne vrijednosti, kako bi kasnije mogli izvršiti un-do na bilo kakve izmjene.

**Prototype patern**

Prototype patern je kreacijski patern koji se odnosi na kloniranje objekta. On ima dvije prednosti: ubrzava instanciranje veoma velikih klasa koje se dinamički učitavaju (kada je kopiranje objekata brže), i čuva znanje o sastavnim dijelovima velikih struktura podataka koje se mogu kopirati bez da je potrebno znati podklase od kojih su kreirane. Suština kod ovog paterna jeste da program kreira objekat željenog tipa, ne tako što će da kreira novu instancu, već tako što će da kopira praznu instancu date klase.

- **PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Kao primjer upotrebe ovog paterna, možemo posmatrati fiktivno “AvMauAzil” proširenu aplikaciju koja bi radila sa slikama i koja bi čuvala grupe fotografija. Sa takvom aplikacijom u određenom momentu, mi možemo poželeti da arhiviramo jednu grupu fotografija, tako što ćemo ih kopirati u drugi album. U ovom slučaju, arhiva postaje kontejner za prototipove koji se može kopirati kada god je to neophodno. Objekti se obično instanciraju od klasa koje predstavljaju dio programa. Prototype patern predstavlja alternativu ovom pristupu jer se objekti kreiraju na osnovu prototipova.

**Proxy pattern**

Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija se aktivacija postiže na osnovu postavljenih pravila. Proxy patern rješava probleme kada se objekt ne može instancirati direktno (npr. zbog restrikcije pristupa). Struktura Proxy patterna je sastavljena od klasa:

1. Subject (ISubject) zajednički interfejs za realne/stvarne subjekte i proksije- surogate (proxies) koji omogućava da se oni koriste naizmjenično.

2. RealSubject je glavni objekat kojeg “predstavlja” proxy.

3. Proxy - implementira isti interfejs kao RealSubject tako da se Proxy može koristiti umjesto RealSubject objekta. Proxy vrši kontrolu pristupa RealSubject objektu - može kreirati i brisati taj objekat.

**PRIMJER UPOTREBE U PROJEKTU „AV-MAU AZIL“:** Ukoliko bi u budućnosti u Azilu bio razvijan sistem naplate karticom (na primjer za kupovinu hrane ili drugih rekvizita za ljubimce), svaki put kada se vrši naplata sa kartice, bio bi korišten proxy pattern. Kartica korisnika je ustvari proxy reprezentacija bankovnog računa. Proxy omogućuje naplatu sa kartice klijenta bez pristupa svim podacima računa korisnika (koji je u pozadini, i koji je implementiran „skupljom“ strukurom). Na ovaj način, Proxy obezbeđuje skladište za kompleksniju klasu čije je instanciranje „skupo“.

**Decorator pattern**

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Može se naprimjer koristi za implementaciju različitih kompresija videa, simultano prevođenje. Ključna tačka implementacije kod ovog paterna jeste da "dekorator" objekti u isto vreme i implementira isti interfejs kao i originalna klasa i sadrži instancu njegove implementacije (originalnu klasu).

Decorator pattern čine sljedeće klase:

1. Component – Originalna klasa (koja sadrži interfejs koji se može mijenjati ili mu se mogu dinamički dodati operacije)

2. IComponent – interfejs koji identificira klase objekata koji trebaju biti dekorisani

3. Decorator – klasa koja odgovara IComponent interfejsu i implementira dinamički prošireni interfejs.

**PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Recimo da AvMAU Azil planira uvesti malu biblioteku i videoteku u svoju instituciju, u svrhe podizanja svijesti o životinjama lutalicama. Dakle, kreirali bi projekat u okviru kojeg bi postojećim klasama koje predstavljaju knjige i filmove, dodali funkcionalnost da se mogu izdavati. Ukratko, dodali bi dvije klase Knjiga I Video.Nakon dodavanja osnovnih klasa dodali bi I klasu Iznajmljivo, koja predstavalja dekorator klasu koja će dodati nove funkcionalnosti postojećim klasama. Na kraju u okviru klase Program mijenjamo kod kako bi main metoda kreirala originalne i dekorisane(Iznajmljive) objekte.

**Facade pattern**

Facade patern se koristi kada sistem ima više identificiranih podsistema (subsystems) pri čemu su apstrakcije i implementacije podsistema usko povezaneOsnovna namjena Facade paterna je da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika).Može se više fasada postaviti oko postojećeg skupa podsistema i na taj način formirati više prilagođenih pogleda na sistem.

- Struktura Facade patterna sačinjena je od sljedećih klasa:

1. Facade – definira i implementira jedinstven interfejs za skup operacija nekog podsistema

2. SubsystemClassN – definira i implementira N-ti interfejs u skupu interfejsa nekog podsistema

- Implementacija podsistema je u okviru odvojenih imenovanih prostora (ili biblioteka), u

kojima se sa specifikatorima pristupa može postaviti željena vidljivost klase.

**PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Facade patern omogućuje jednostavan interfejs i kontroliše pristup nizu komplikovanih interfejsa podsistema. Tipičan primjer gdje bi se ovaj patern mogao upotrebiti u okviru našeg projekta jese za eventualno naručivanje namirnica za kućne ljubimce preko interneta. Ukoliko vlasnik ljubimca često naručuje hranu za jubimca preko našeg sajta, on može sačuvati podatke o svojooj kreditnoj kartici I o adresi gde namirnice trebaju biti dostavljene. Na ovaj način bi ubrzali proces naručivanja robe. Ono što razlikuje Façade patern od Adapter paterna ili Decorator paterna je to što su interfejsi koji se kreiraju u potpunosti novi (nisu povezani sa postojećim zahtjevima i ne moraju se podudarati sa postojećim interfejsima).

**Singleton patern (unikat)**

Singleton pattern ograničava instanciranje klase i osigurava da samo jedna instanca date klase postoji i pruža globalnu tačku pristupa ka toj instanci. Patern osigurava da je klasa instancirana samo jednom i da su svi zahtevi upućeni ka tom jednom i samo jednom objektu. Tipičan primjer su globalni objekti (kao što je neka Configuration klasa) ili deljeni resursi (kao što je event queue).

**PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Singleton patern bi koristili u kombinaciji sa Configuration klasom koja po svojim svojstvima i namjeni odgovora upotrebi sa singleton patternom. Configuration klasa predstavlja konfiguraciju, računara, .Net klijent aplikacija, ASP.Net aplikacija, Web direktorija i dr. Ova konfiguracijska klasa će u projektu biti instanciarana samo jednom i imati globalnu tačku pristupa.

**Observer patern**

Observer patern definiše način na koji jedna ili više klasa treba da budu upozorene na promjene u drugoj klasi, odnosno obaviještene da su nastale promjene u drugoj klasi. Observer pattern se koristi ukoliko postoje jedna-na-više veze između objekata takve da ako se jedan objekat modifikuje, ostali će biti obavješteni automatski. Patern koristi tri klase aktera: subjekat, observer i objekat. Observer nadzire subjekat i svaki put kad se on promijeni obavještavaju se objekti.

**PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** U okviru projekta “AvMau Azil” dobar primjer upotrebe Observer paterna bi bio prilikom slanja notifikacija o novo-pristigloj životinji u Azil. Vozač po preuzimanju životinje sa terena, šalje notifikaciju ostalim uposlenicima Azila. Nakon toga Veterinar zna da treba obaviti pregled, Upravitelj da će biti registrirana nova životinja u azil, Higijeničar treba da spremi boks itd..

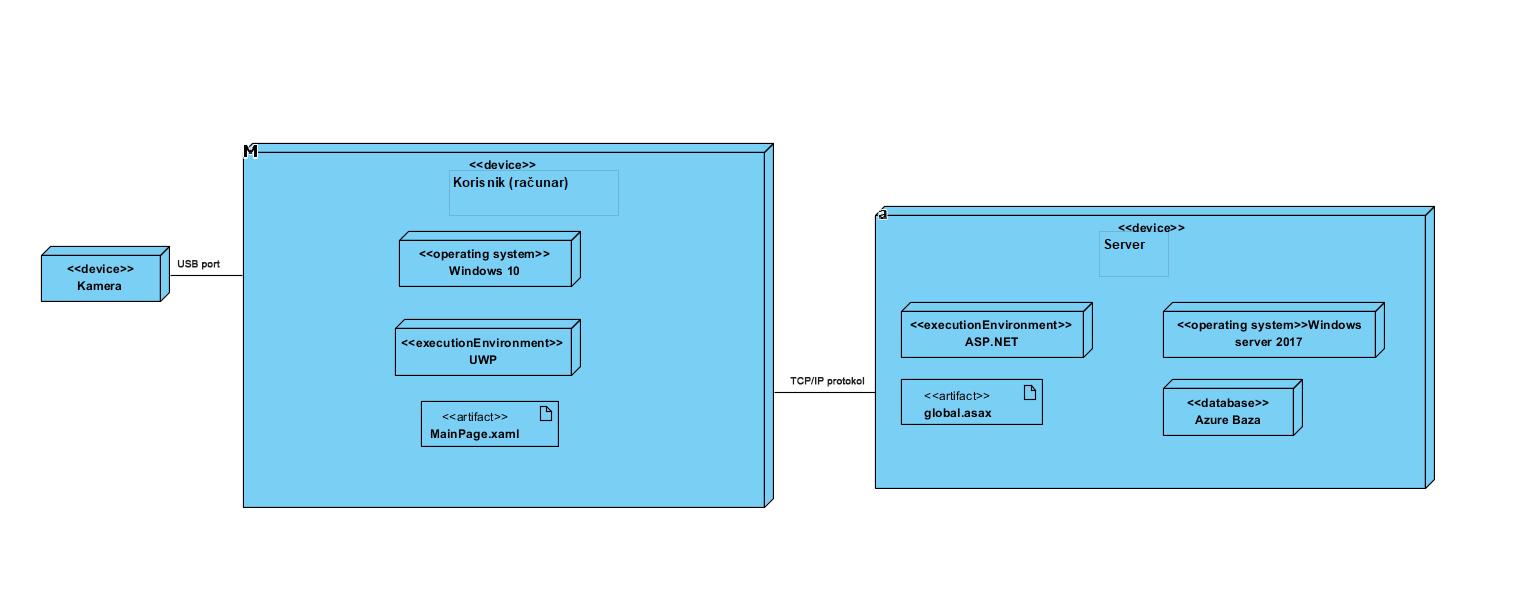
**Interpreter patern**

Interpreter patern spada u patene ponašanja i određuje kako evaluirati/interpretirati rečenice ili izraze u jeziku. Ovaj patern implementira Expression interfejs koji govori kako da se interpretira određeni kontekst. Često se koristi u kompajlerima i parserima.

**PRIMJER PRIMJENE U PROJEKTU “AV-MAU AZIL”:** Ovaj patern se može koristiti u slučaju procjene validnosti šifre admina ili korisnika prilikom korištenja aplikacije. Kontrola paterna se odnosi na provjeru da li je unesena šifra po traženom standardu.

Radila: Ajna Zatrić

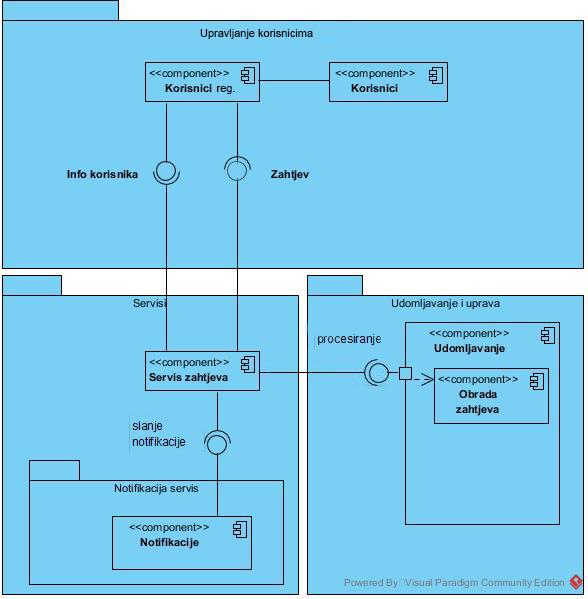
Dijagrami raspoređivanja

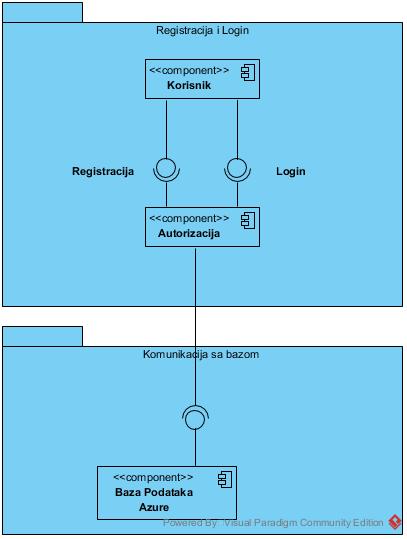


Radila: Ajna Zatrić

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

Dijagrami komponenti

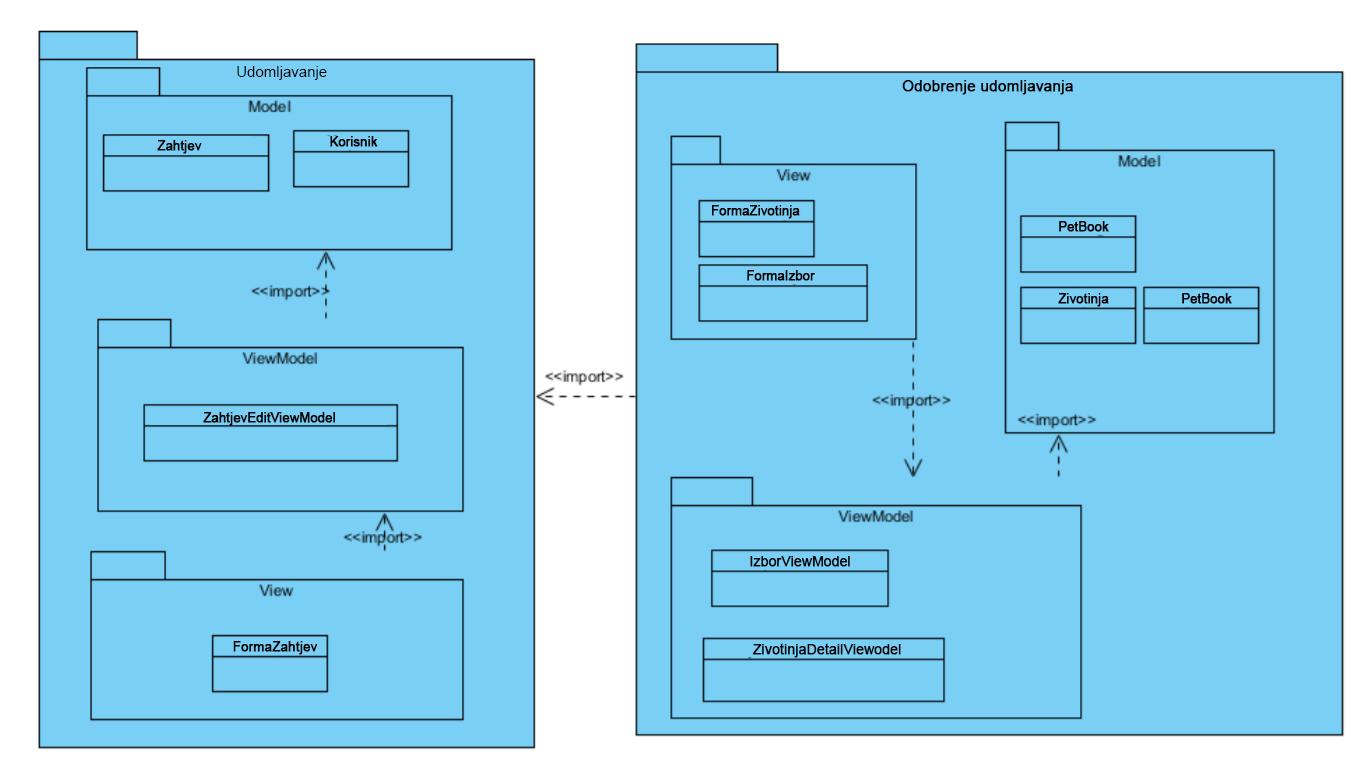


Komunikacija sa bazom

Radila: Ajna Zatrić

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

Dijagrami paketa



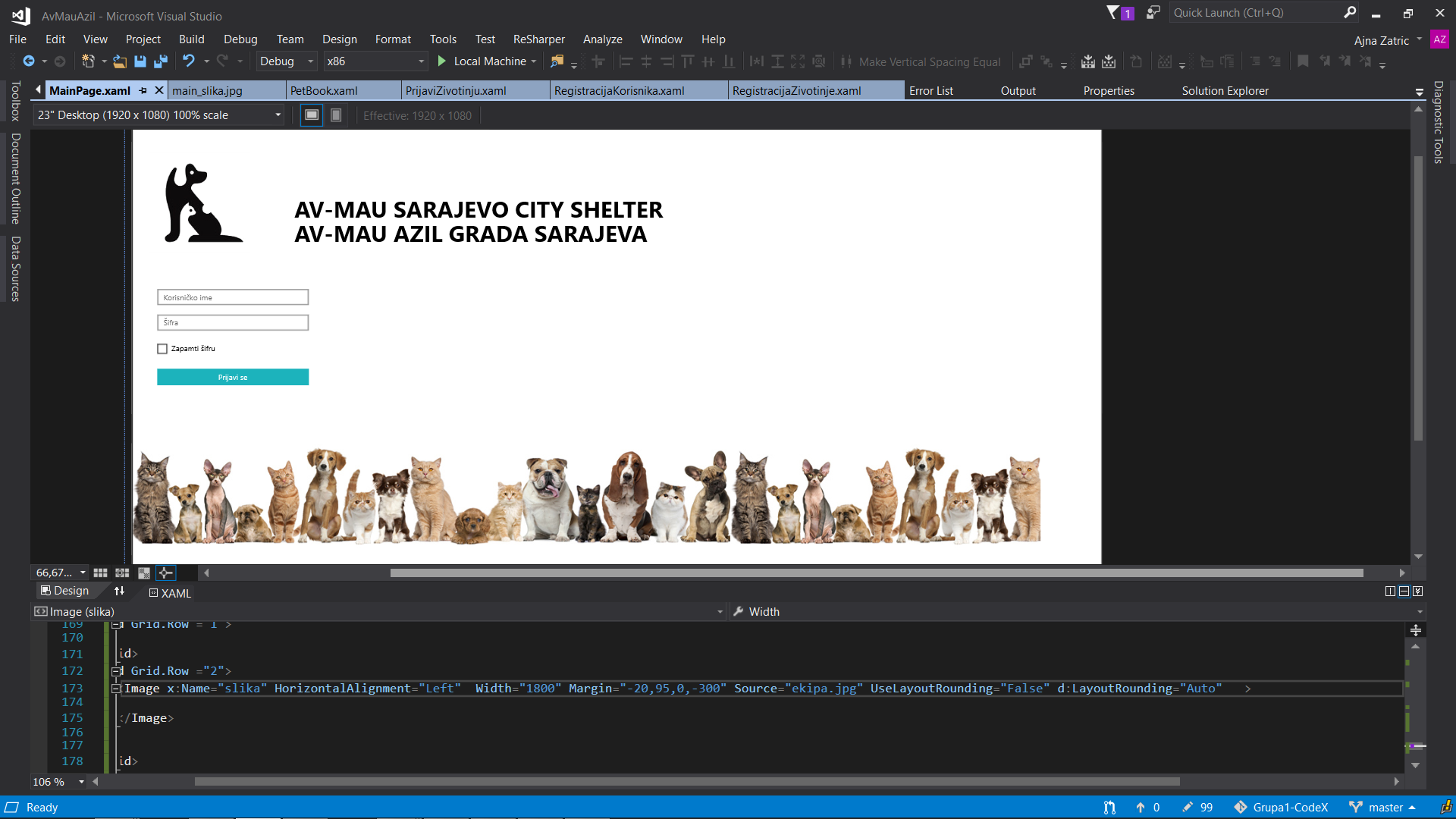
Radila: Ajna Zatrić

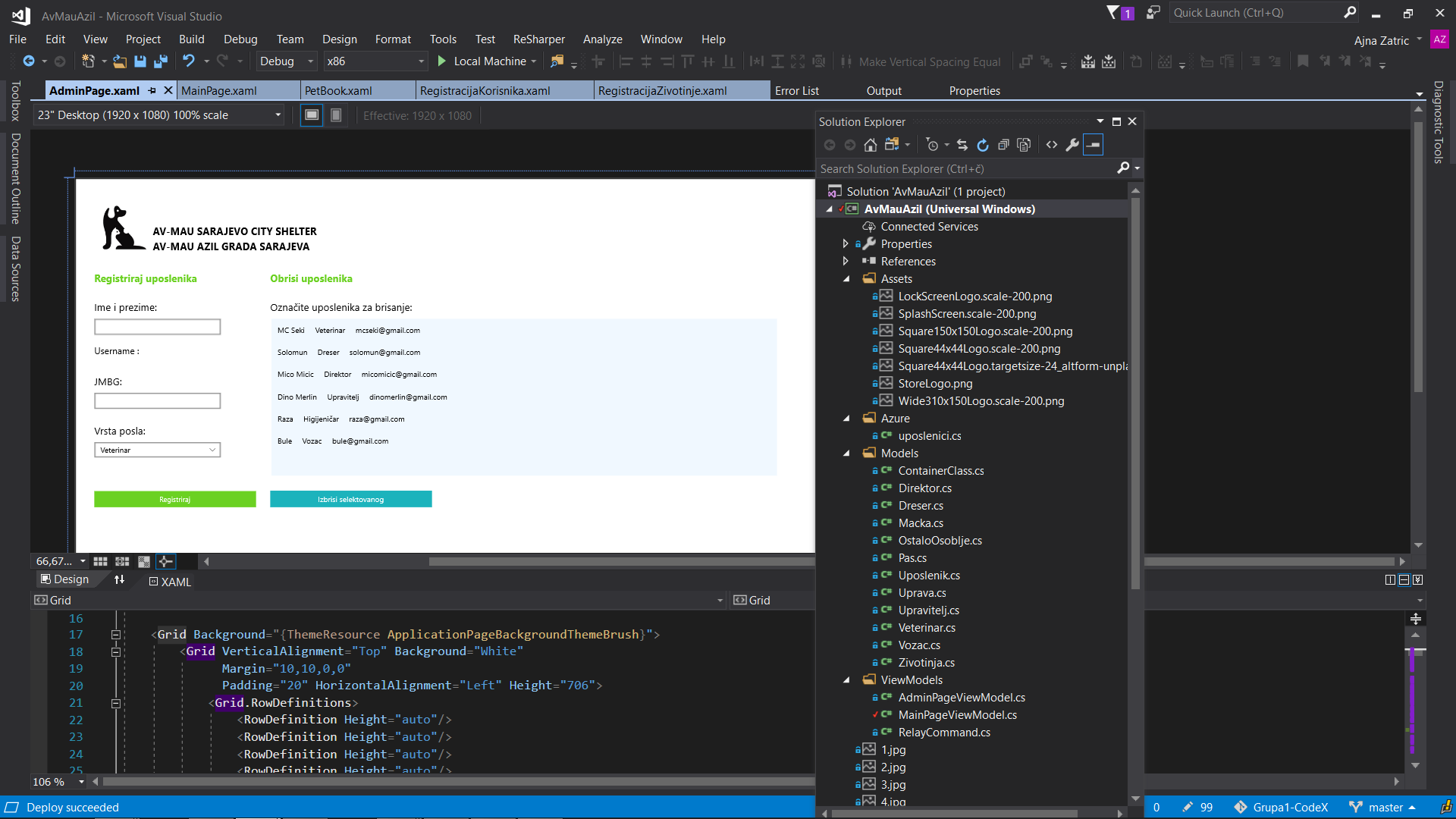
Za ostale dijagrame pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX)

|  |  |  |
| --- | --- | --- |
|  | 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.

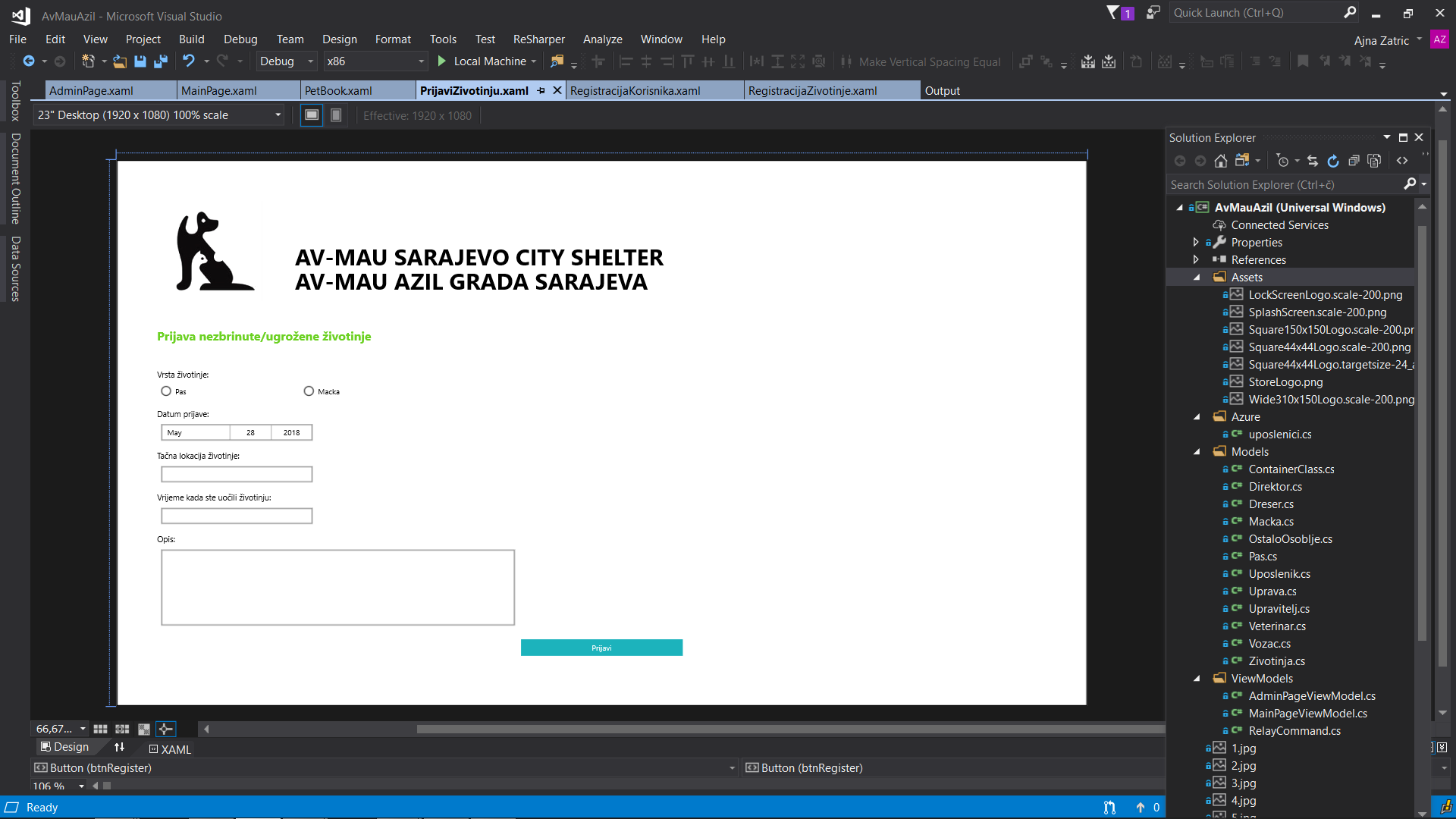
Korisnički interfejs - forme za administratora, kreirane u XAMLu. Za ostale forme, molimo Vas pogledajte [ovdje](https://github.com/ooad-2017-2018/Grupa1-CodeX/tree/master/Projekat)



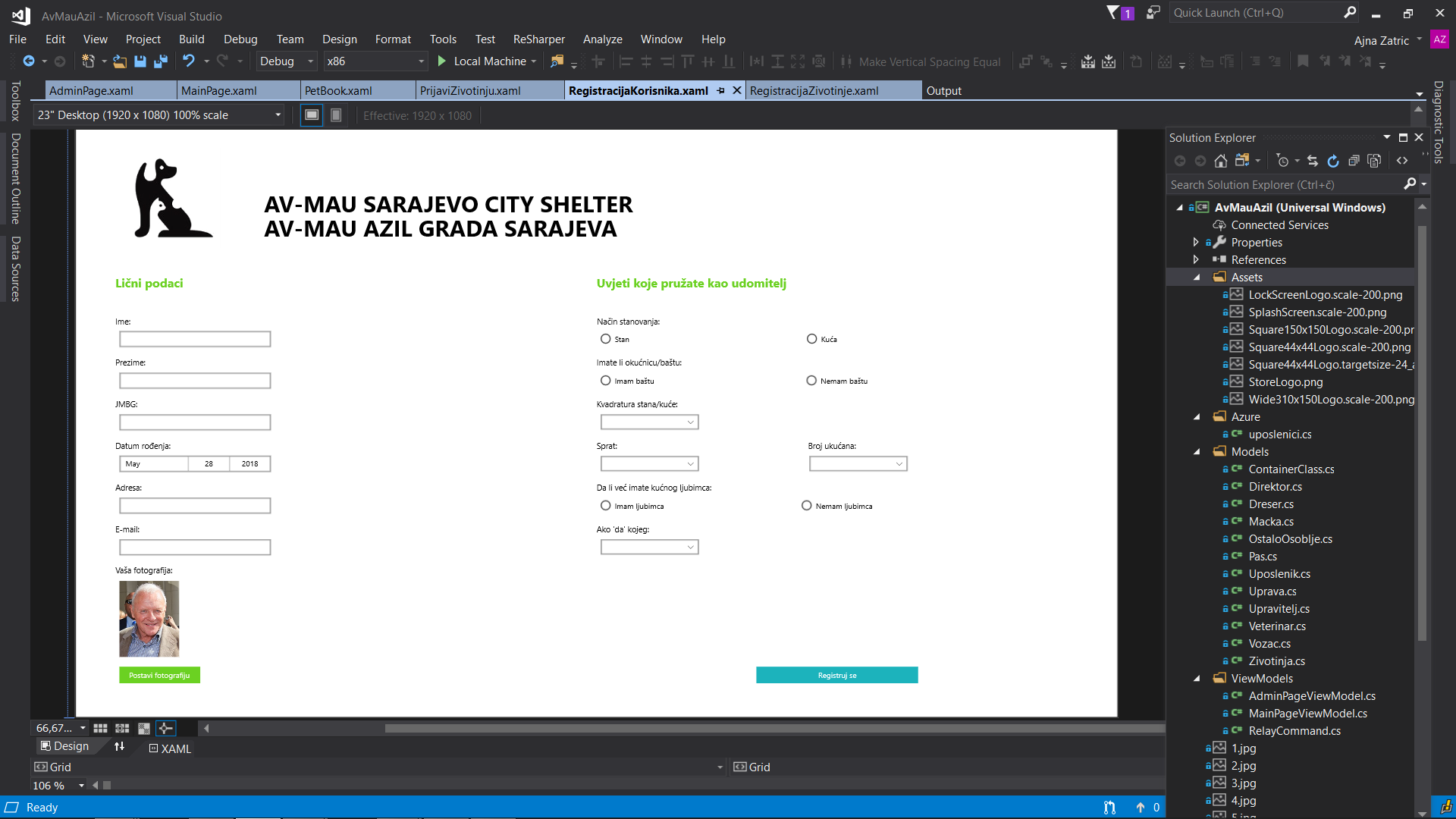


2.UWP - Implementacija modela-funkcionalnosti aplikacije za forme administracije. (Arhitektura aplikacije je MVVM implementacija se vrši na osnovu urađenog modela)

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

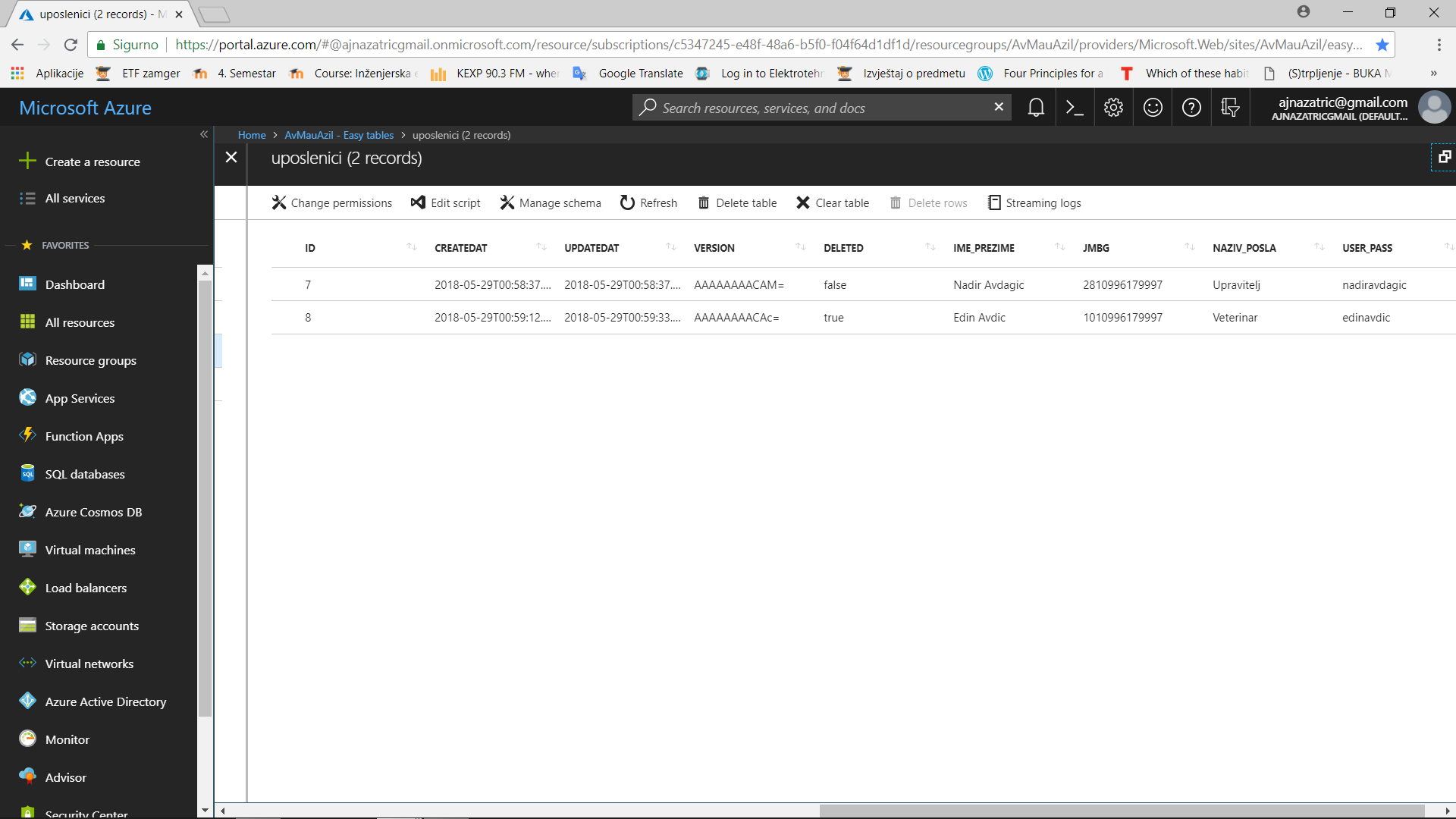


Prijava nezbrinute životinje



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. 

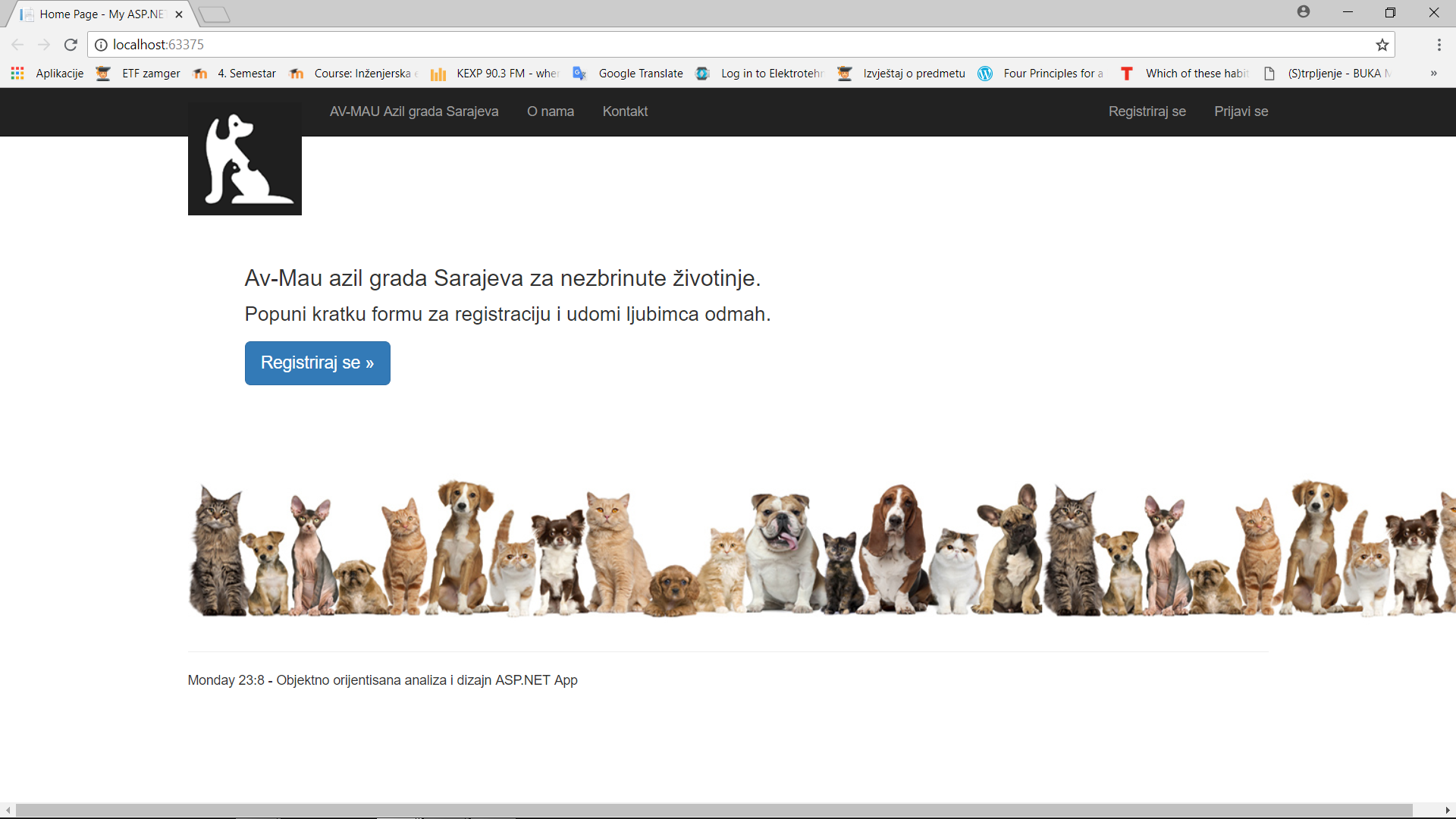
4. Kreiranje ASP.NET projekta, arhitektura aplikacije je MVC, implementacija modela

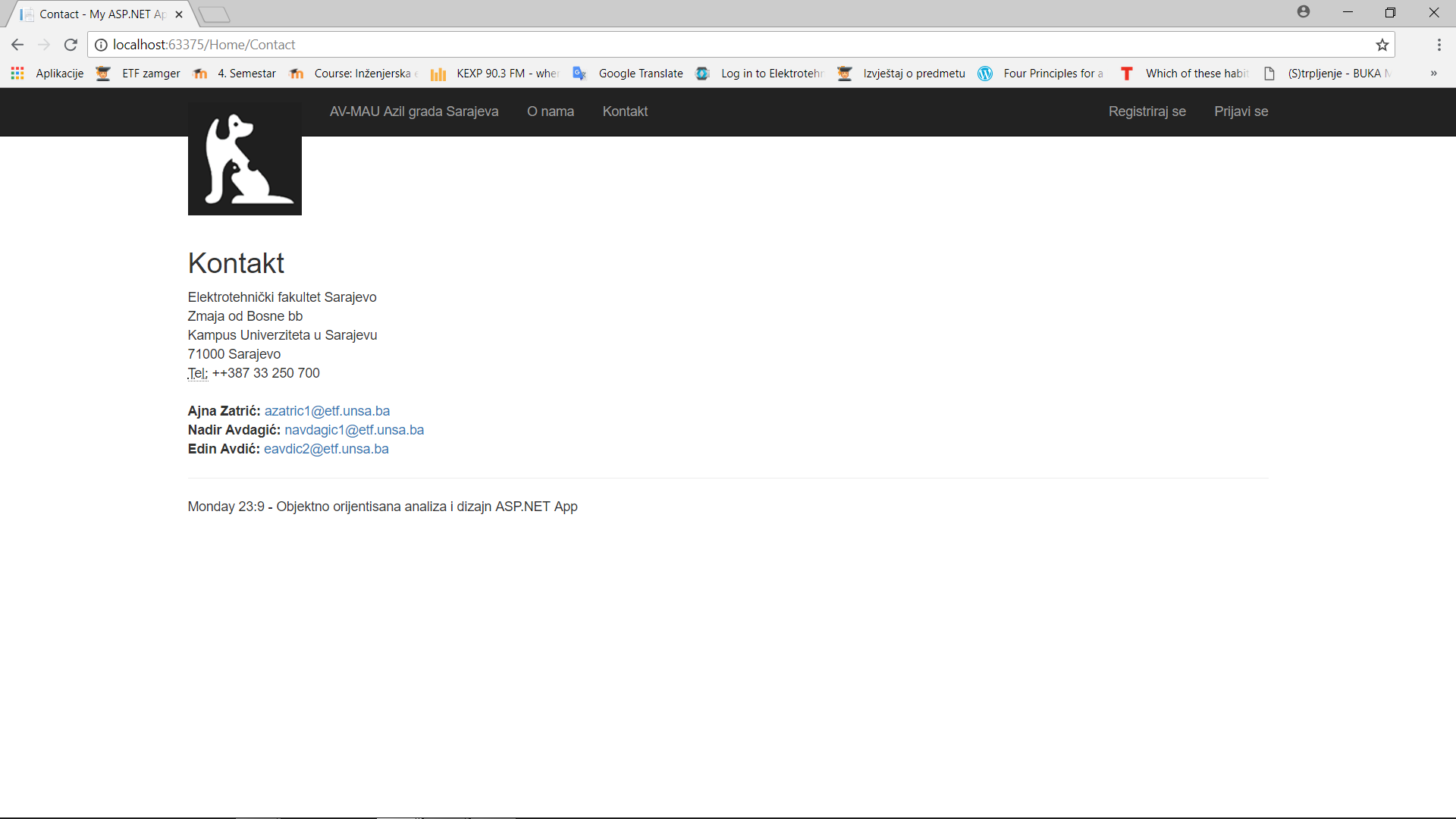
Kreiran je ASP.NET projekat, koristeću Model View Control arhitekturalni patern. Model je implementiran što se vidi na idućoj slici.



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

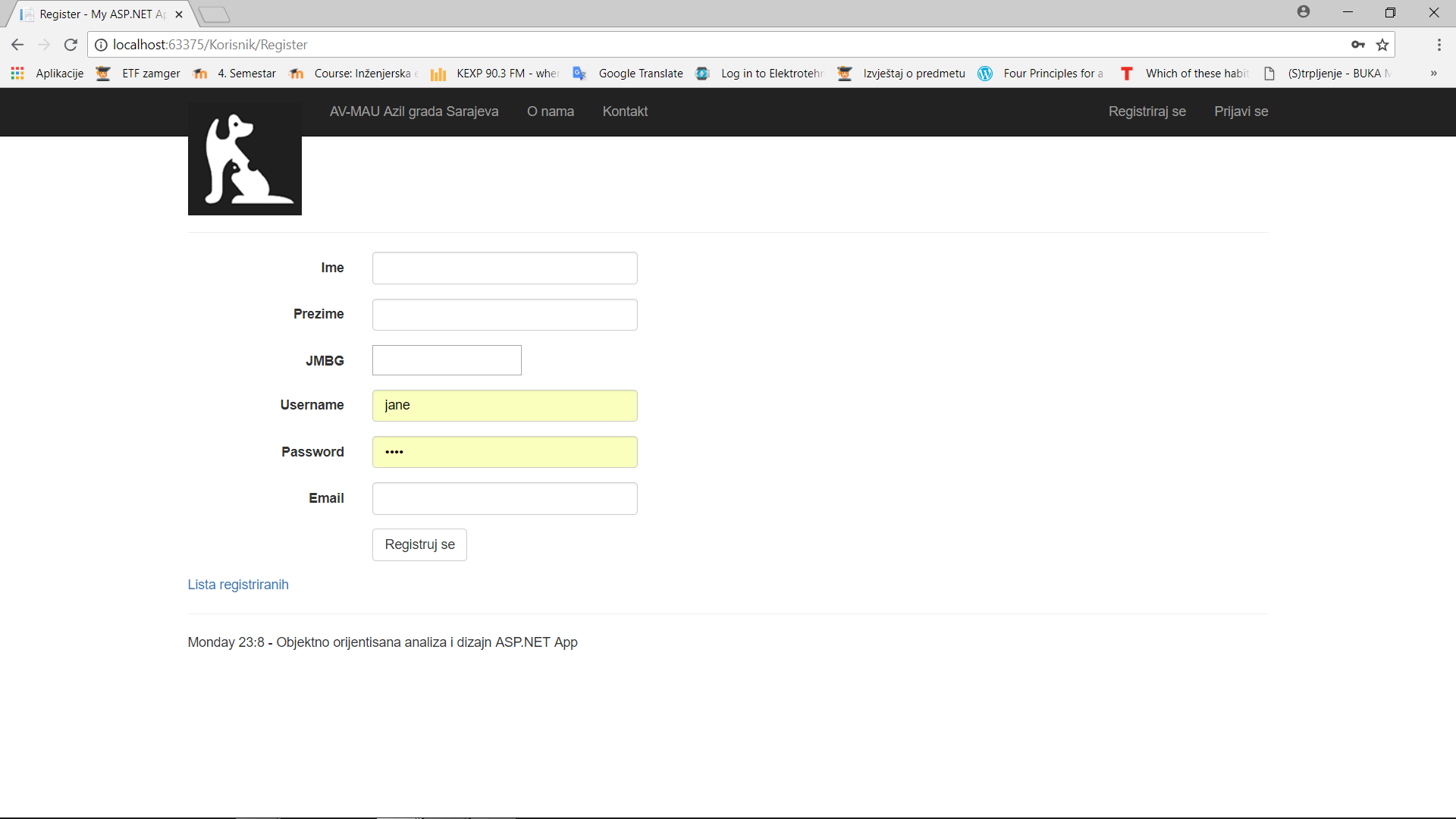
ASP.NET

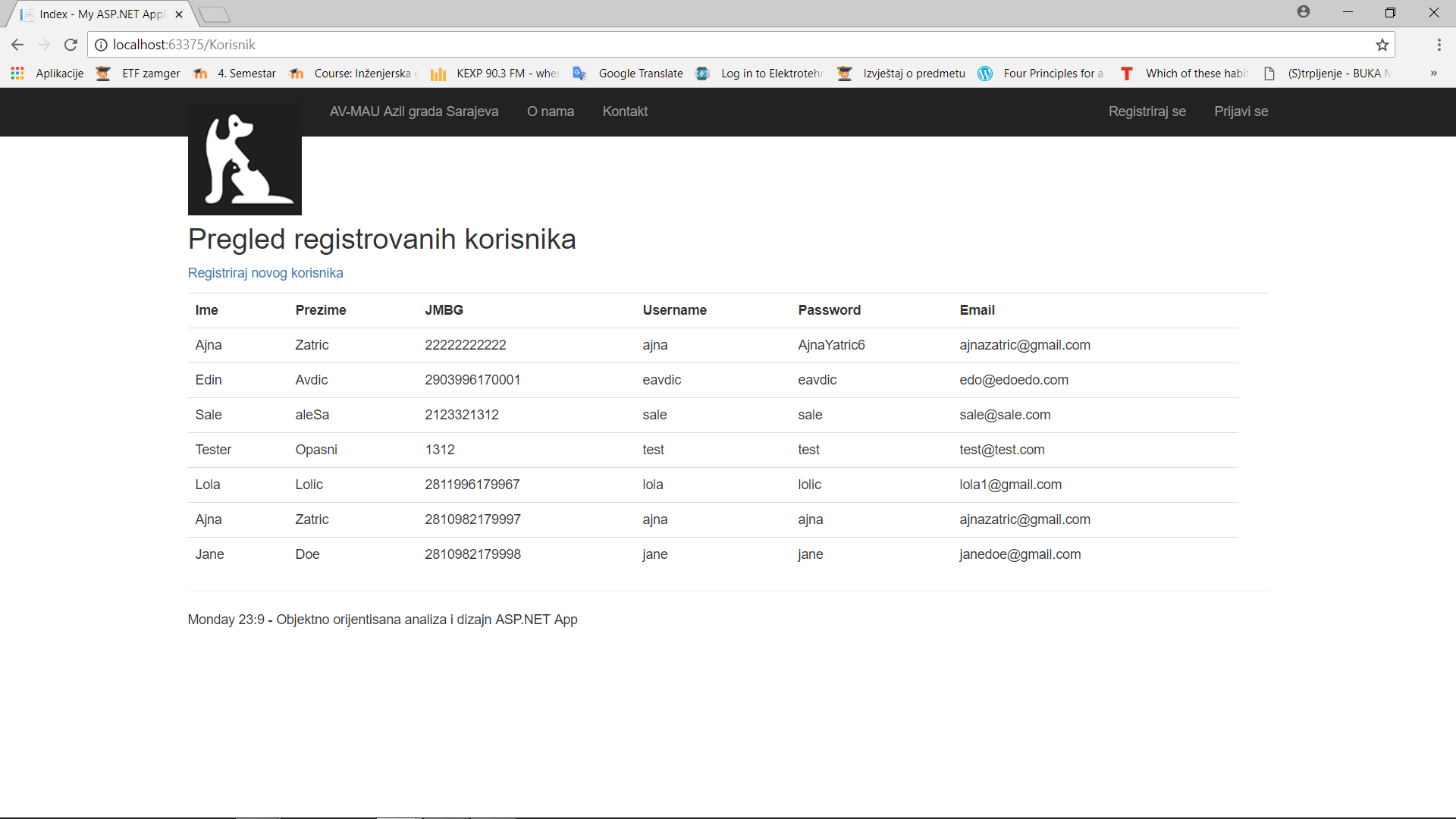




6.Rad sa bazom podataka

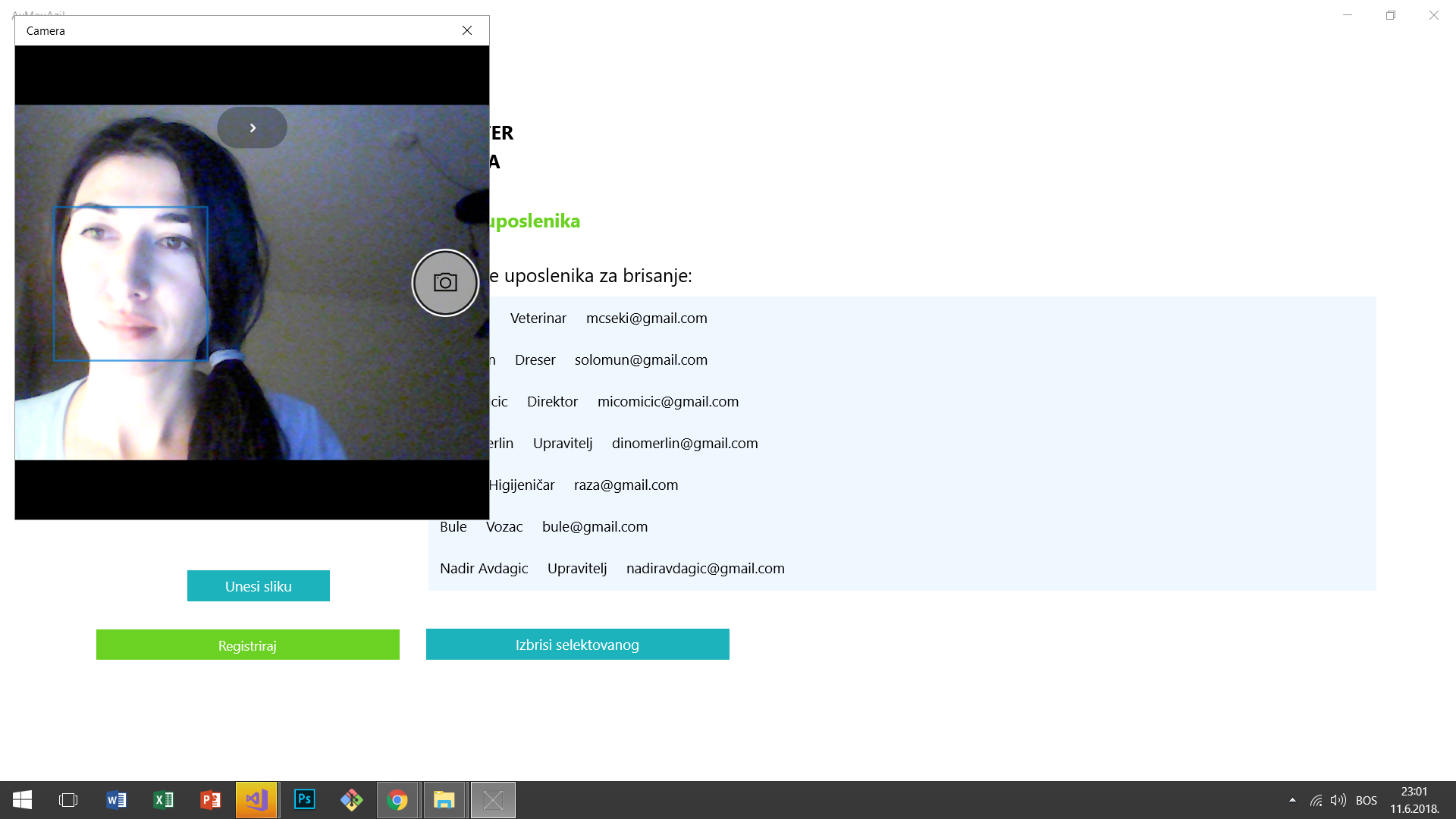
Omogućeno registriranje korisnika preko ASP.NEt dijela aplikacije kao što je vidljivo iz sljedećih slajdova.

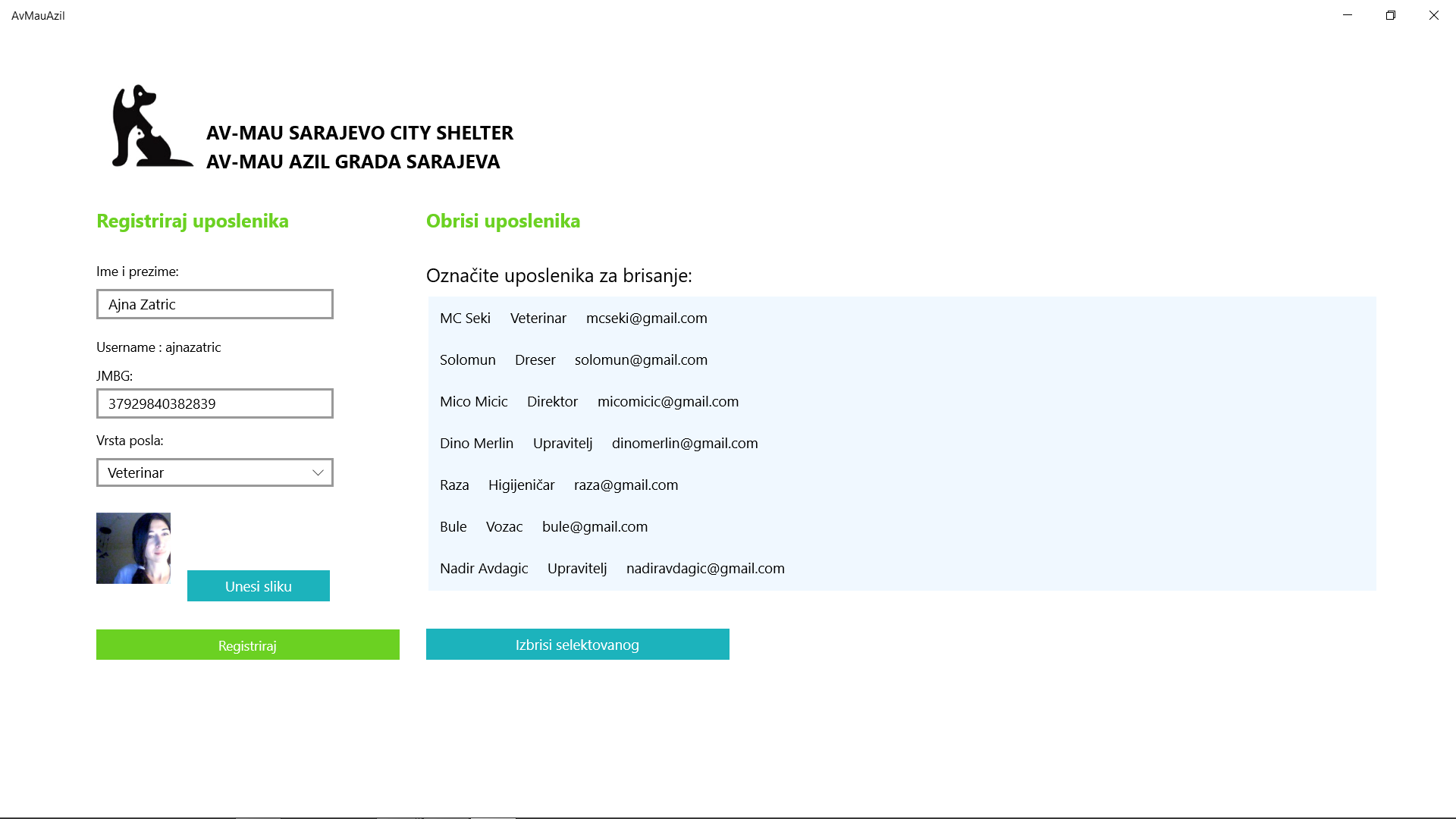




7. Rad sa vanjskim uređajima, specifičnim funkcionalnostima (Kamera)

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.

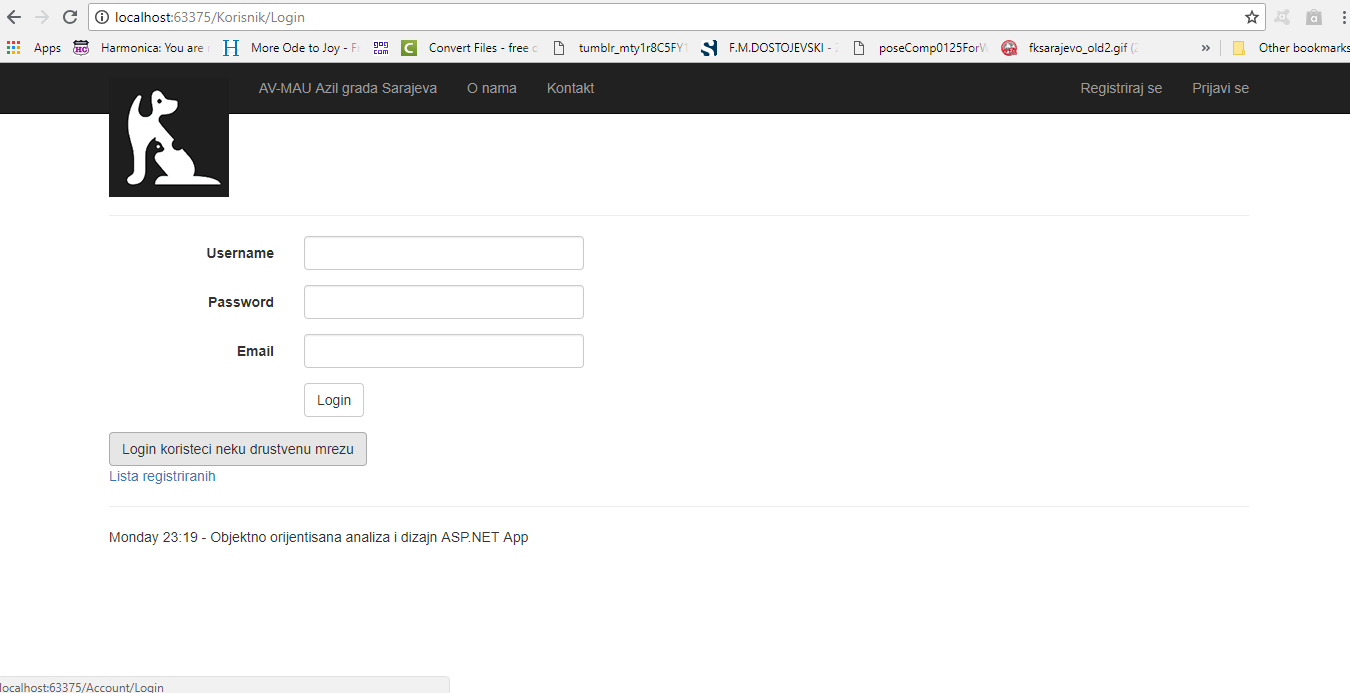


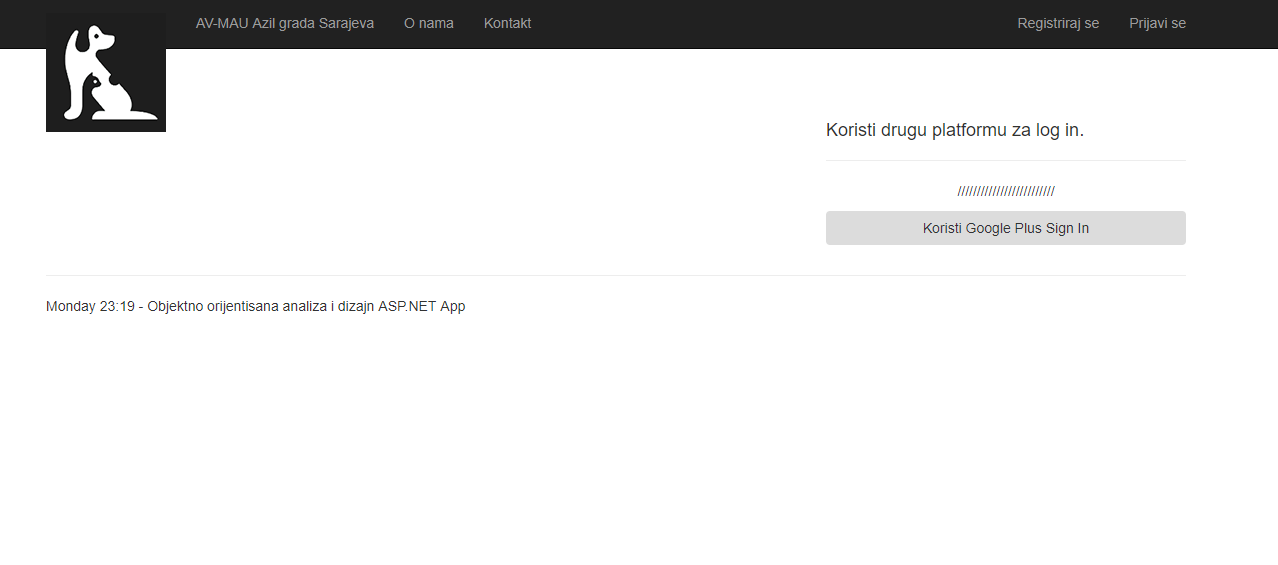


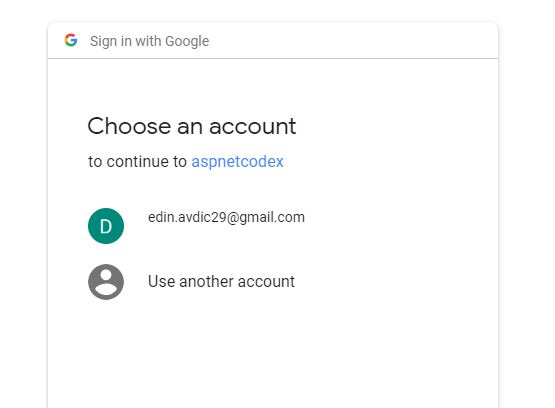
8.-9.-10.Implementacija ASP.NET WEB API servisa i

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).

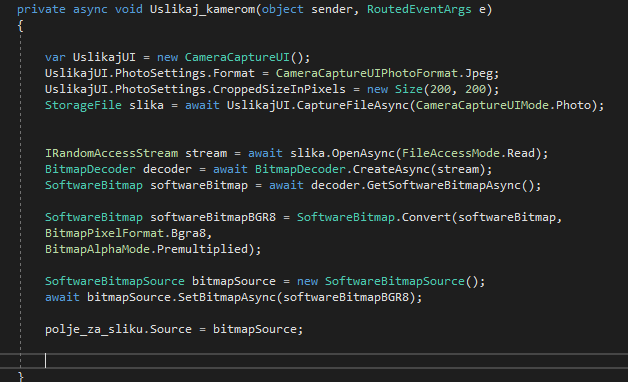
# Indikacija 1: Klase se veoma malo razlikuju pa nije potrebno nasljeđivanje. “Collapse hierarchy - collapse a superclass and subclass if their implementations are very similar.”

**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.

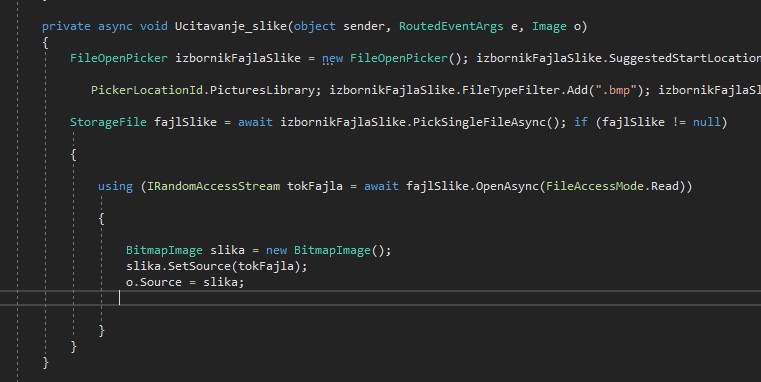


# Indikacija 3: Rutina ima loše - neopisno ime. Ako rutina ima loše ime, potrebno joj je promijeniti ime u definiciji i na mjestima poziva. “A routine has a poor name”

# **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.



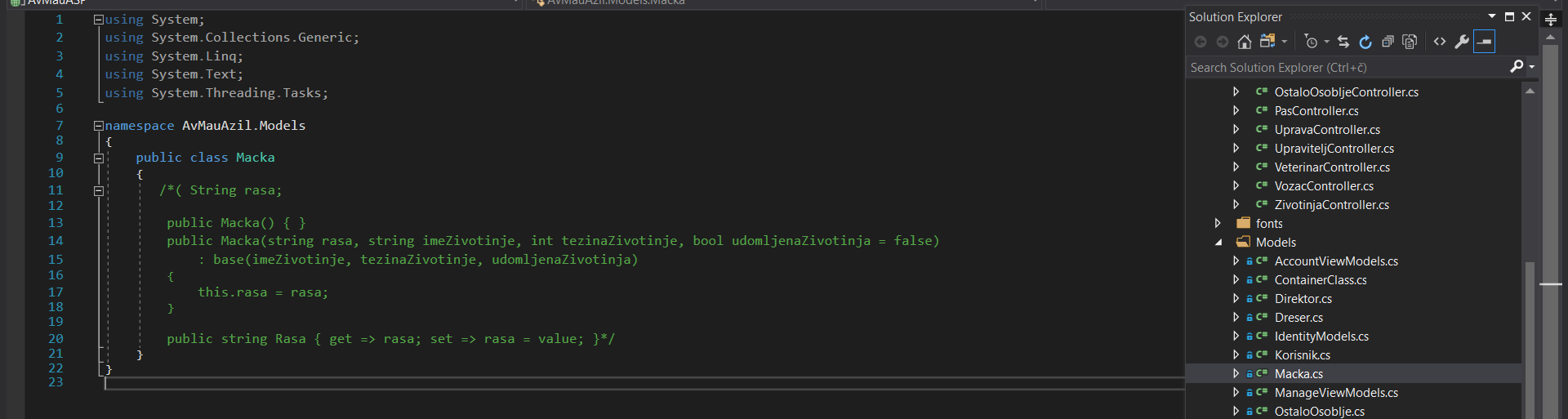
# 

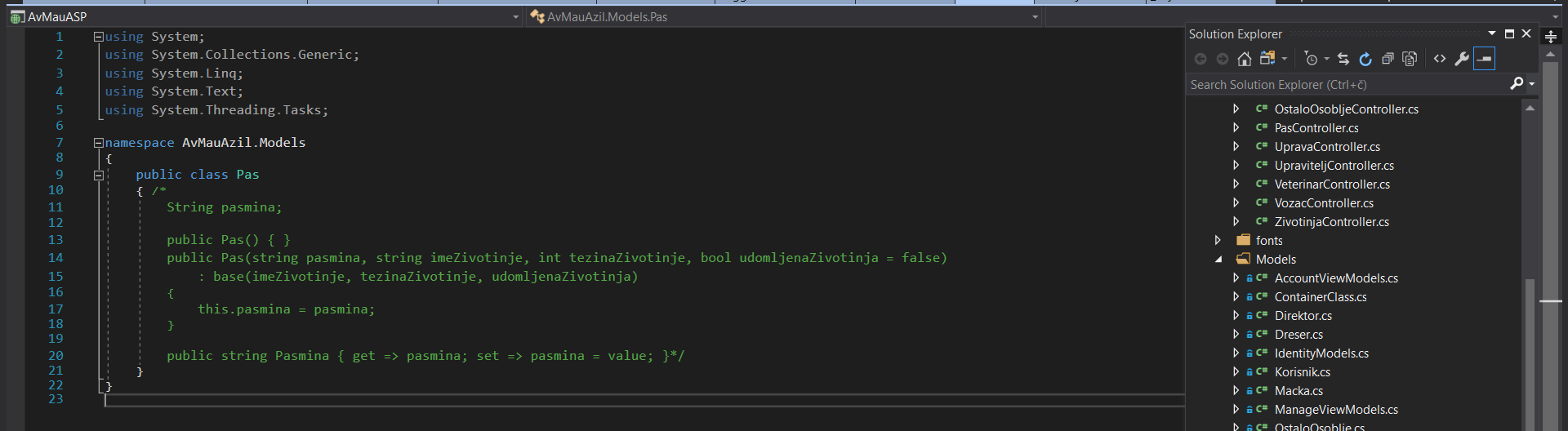
# Indikacija 5: Podaci su javni. “Data members are public”.

**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”

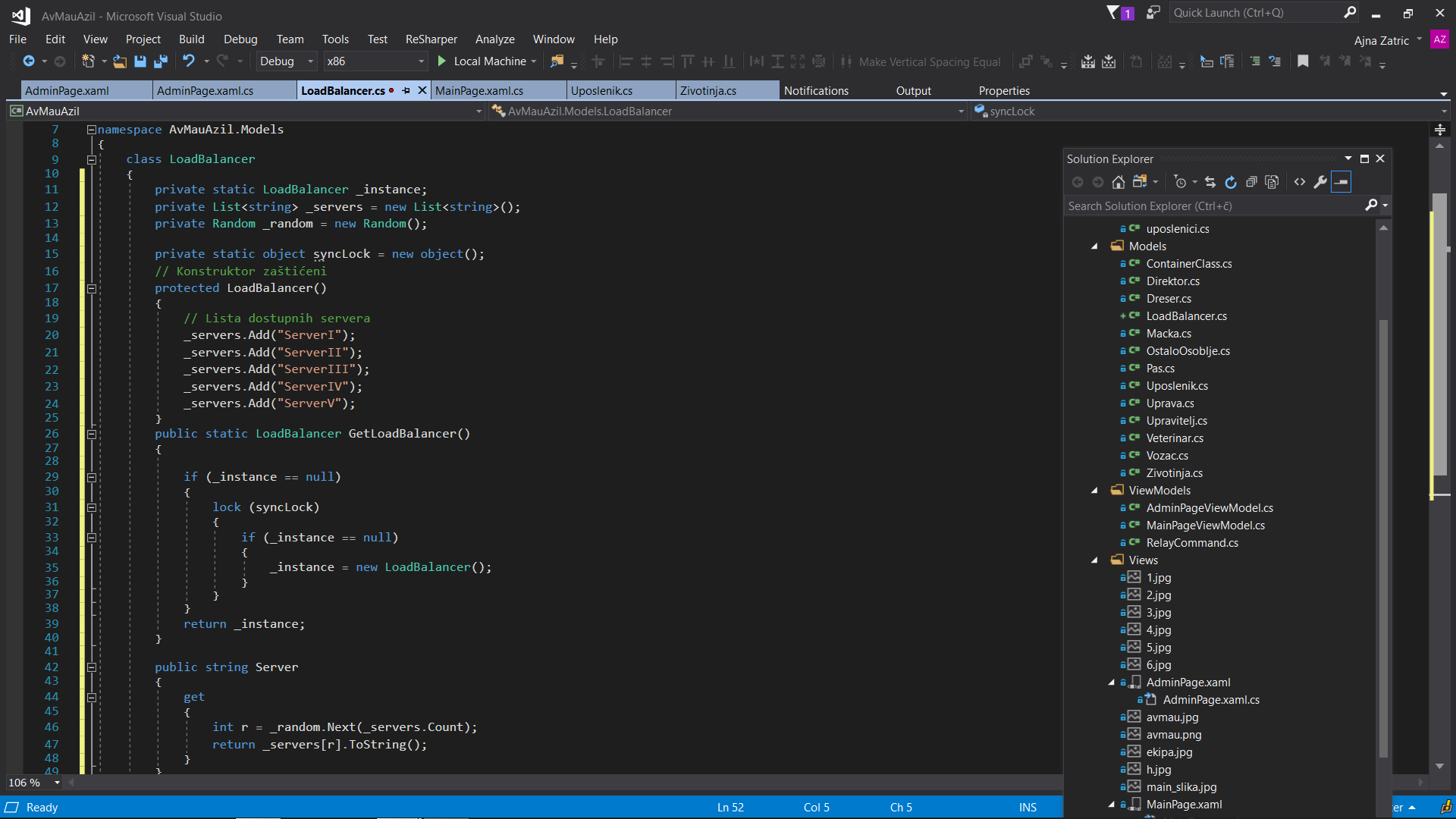
# **Obrazloženje zašto je refaktoring urađen i šta se time postiglo:** Ponekad rezultat refaktoringa koda jeste činjenica da klasa nema mnogo da radi. U našem slučaju smo ukidanjem hijerarhije klasa Životinja -> Pas i Životinja->Mačka primjetili da su sve odgovornosti klasa Pas i Mačka dodijeljene klasi “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.