Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Seminar iz Objektnog oblikovanja

**Aplikacija za praćenje tjelesne težine, obroka i aktivnosti kućnih ljubimaca – Chubster**

Tehnička dokumentacija

Petra Žrvnar

Dominik Tomić

Zagreb, Siječanj 2020.

Sadržaj

[1. Uvod 3](#_Toc30943994)

[Design story 3](#_Toc30943995)

[2. Funkcionalni zahtjevi 5](#_Toc30943996)

[Dionici sustava 5](#_Toc30943997)

[Funkcionalni zahtjevi registriranog korisnika 5](#_Toc30943998)

[UC1: Registracija 5](#_Toc30943999)

[UC2: Prijava 6](#_Toc30944000)

[UC3: Odjava 6](#_Toc30944001)

[UC4: Dodavanje novog kućnog ljubimca 6](#_Toc30944002)

[UC5: Pregled podataka o ljubimcima 7](#_Toc30944003)

[UC6.1: Izmjena podataka o ljubimcima 7](#_Toc30944004)

[UC6.2: Postavljanje ciljeva 8](#_Toc30944005)

[UC7: Brisanje kućnog ljubimca 8](#_Toc30944006)

[UC8: Unos vaganja 8](#_Toc30944007)

[UC9: Unos obroka 9](#_Toc30944008)

[UC10: Unos aktivnosti 9](#_Toc30944009)

[UC11: Izmjena unesenih zapisa o vaganju, obroku i aktivnosti 10](#_Toc30944010)

[UC12: Brisanje unesenih zapisa o vaganju, obroku i aktivnosti 10](#_Toc30944011)

[UC13: Tablični i grafički prikaz uneseni podataka o vaganju, obroku i aktivnosti 10](#_Toc30944012)

[Specifikacija API sučelja 11](#_Toc30944013)

[Registracija 11](#_Toc30944014)

[Prijava 11](#_Toc30944015)

[Odjava 11](#_Toc30944016)

[Dodavanje novog kućnog ljubimca 11](#_Toc30944017)

[Pregled podataka o ljubimcima 12](#_Toc30944018)

[Izmjena podataka o ljubimcima i postavljanje ciljeva 12](#_Toc30944019)

[Brisanje kućnog ljubimca 12](#_Toc30944020)

[Unos vaganja 12](#_Toc30944021)

[Unos obroka 12](#_Toc30944022)

[Unos aktivnosti 12](#_Toc30944023)

[Izmjena unesenih zapisa o vaganju 13](#_Toc30944024)

[Izmjena unesenih zapisa o obroku 13](#_Toc30944025)

[Izmjena unesenih zapisa o aktivnosti 13](#_Toc30944026)

[Brisanje zapisa o vaganju 13](#_Toc30944027)

[Brisanje zapisa o obroku 13](#_Toc30944028)

[Brisanje zapisa o aktivnosti 14](#_Toc30944029)

[Tablični i grafički prikaz uneseni podataka o vaganju, obroku i aktivnosti 14](#_Toc30944030)

[3. Model domene 15](#_Toc30944031)

[Dijagram razreda 15](#_Toc30944032)

[Opis dijagrama razreda 15](#_Toc30944033)

[4. Implementacija 17](#_Toc30944034)

[Implementacija podatkovne razine i objektno – relacijsko mapiranje 17](#_Toc30944035)

[Implementacija modela domene 18](#_Toc30944036)

[Konfiguracija i mapiranje 20](#_Toc30944037)

[Rad s bazom podataka 22](#_Toc30944038)

[Implementacija web aplikacije 25](#_Toc30944039)

[Organizacija repozitorija i ostvarenje aplikacije 25](#_Toc30944040)

[Korisničko sučelje i implementacija funkcionalnih zahtjeva 27](#_Toc30944041)

[Implementacija desktop aplikacije 30](#_Toc30944042)

[Organizacija repozitorija i ostvarenje aplikacije 30](#_Toc30944043)

[Korisničko sučelje 31](#_Toc30944044)

[5. Zaključak 34](#_Toc30944045)

# Uvod

Cilj ovog seminarskog rada je modeliranje i implementacija sustava za praćenje tjelesne težine, obroka i dnevne aktivnosti kućnih ljubimaca – implementacija sustava ***Chubster***. Ovaj sustav svojim korisnicima služi kao pomoć pri sukobu s problemom pretilosti kućnih ljubimaca. Danas sve veći broj ljubimaca ima povećanu tjelesnu masu, ili je pretilo. Razlog tomu je najčešće taj što vlasnici ljubimaca ne znaju pravilno hraniti svoje ljubimce.

Ovaj sustav omogućuje svojim korisnicima, vlasnicima ljubimaca, da vode evidenciju o težini njihovih ljubimaca, količini kalorija koje unose njihovi ljubimci i količini dnevne aktivnosti njihovih ljubimaca. Tako sustav korisnicima omogućuje da vode dnevnik težine, obroka i aktivnosti za sve svoje ljubimce. Korisnici mogu imati više ljubimaca, te za svakog od njih voditi evidenciju o težini, obrocima i aktivnosti.

Sustav svojim korisnicima također služi kao izvor korisnih informacija o prehrani njihovih ljubimaca. Tako korisnici mogu, koristeći ovaj sustav, saznati koliko bi zapravo trebao težiti njihov ljubimac, koliko bi dnevno kalorija trebao unositi, te koliko bi ljubimac trebao biti aktivan.

Sustav služi kao jedinstvena cjelina gdje korisnici mogu pohranjivati, uređivati i pregledavati zapise o težini, aktivnosti i količini unesenih kalorija njihovih ljubimaca. Sustav korisnicima omogućuje da unesene podatke pregledaju u tabličnom ili grafičkom obliku, što korisnicima omogućuje praćenje trenda unosa kalorija, težine i količine aktivnosti njihovih ljubimaca.

Danas već postoje slični sustavi za praćenje težine, unosa kalorija i aktivnosti, kod ljudi. Korisnici tim sustavima vode evidenciju o svojoj težini, aktivnosti i svojem unosu kalorija. Primjer takvog sustava je Samsung Health. Korisnici uporabom ovog sustava mogu pratiti trend svoje težine, unosa kalorija i aktivnosti.

Iako postoje sustavi za praćenje ovih podataka u ljudi, trenutno ne postoji sustav koji bi svojim korisnicima omogućio da prate težinu, unesene kalorije i aktivnost njihovih kućnih ljubimaca.

U nastavku dokumenta će se prvo opisati funkcionalni zahtjevi sustava. Zatim će biti prikazan dijagram razreda modela domene. Nakon toga će se opisati podatkovni sloj i način na koji je podatkovni sloj povezan sa slojem poslovne logike. Na kraju je dan prikaz izrađene desktop i web aplikacije, odnosno dan je prikaz na koji način su implementirani ključni funkcionalni zahtjevi.

## Design story

***Chubtser*** je sustav koji korisnicima omogućuje **vođenje evidencije o težini**, **količini unesenih kalorija**, te **aktivnosti** njihovih kućnih ljubimaca. Ovaj sustav tako služi kao svojevrstan dnevnik težine, količine unesenih kalorija i količine aktivnosti, koje vlasnici vode za svoje ljubimce.

Vlasnik ljubimca, u daljnjem tekstu korisnik, se prilikom početka rada **mora prijaviti na sustav** uporabom postojećeg **korisničkog računa**. Korisnički račun se sastoji od e-mail-a i lozinke. Kako bi se prijavio na sustav, korisnik se mora **registrirati**. Registracijom nastaje korisnički račun kojeg korisnik koristi za prijavu na sustav. Korisnik se u jednom danu može prijaviti **više puta** na sustav. Na sustavu istovremeno može biti **prijavljeno više korisnika**.

Svaki korisnik **može imati više ljubimaca**. Korisnik može **pregledati i mijenjati podatke** o svojim ljubimcima, te također **brisati** ljubimce koji su vezani uz njegov korisnički račun. Za svakog ljubimca su definirani **ciljevi**. Ljubimac ima tri cilja, **ciljanu težinu, ciljanu količinu dnevne aktivnosti i ciljanu dnevnu količinu unesenih kalorija**. Ove ciljeve prilikom stvaranja ljubimca **izračunava i postavlja sustav**, a korisnik ih naknadno može **proizvoljno mijenjati**. Tako korisnik prilikom stvaranja ljubimca **ne može ručno postaviti ciljeve**, ali ih naknadno, nakon uspješno provedenog stvaranja ljubimca, može izmijeniti.

Za svakog ljubimca, korisnik može voditi evidenciju o težini, unesenim kalorijama i količini aktivnosti. Korisnik može stvarati nove zapise o težini, unesenim kalorijama i količini aktivnosti za svakog ljubimca. Tako stvorene zapise korisnik naknadno može **urediti ili izbrisati**.

Korisnik može **pregledati unesene zapise** za svoje ljubimce. Sustav tako korisniku omogućuje da zapise pregleda u obliku tablice i u obliku grafa. Korisnik pregledava zapise po datumima kada su ti zapisi nastali.

# Funkcionalni zahtjevi

U nastavku poglavlja su opisani dionici sustava, te njihovi funkcionalni zahtjevi. Ovim zahtjevima su opisane usluge sustava koje su na raspolaganju njegovim korisnicima.

**Dionici sustava**

Dionici sustava su svi entiteti koji imaju određeni interes za sustav. Za ovaj sustav postoji samo jedna vrsta dionika, a to su:

* Registrirani korisnici

U nastavku ovog poglavlja će se dalje samo razmatrati i opisati funkcionalni zahtjevi registriranog korisnika, kako je to jedina vrsta dionika.

**Funkcionalni zahtjevi registriranog korisnika**

Registrirani korisnik jest korisnik sustava koji **posjeduje ispravan korisnički račun** u bazi podataka.

U nastavku poglavlja će zahtjevi registriranog korisnika biti opisani kroz *use case*-ove. U **svakom** opisanom *use case*-u postoji samo jedan dionik – **registrirani korisnik**, (u nastavku teksta **korisnik**), te se u nastavku **neće** eksplicitno navoditi dionici pojedinog *use case*-a.

### UC1: Registracija

* **Sažetak:** Stvaranje korisničkog računa u bazi podataka, koji služi za prijavu na sustav.
* **Obrazloženje:** Kako bi korisnik mogao koristiti sustav, on **nužno mora imati postojeći i ispravan korisnički račun** u bazi podataka uz pomoć kojeg se može prijaviti na sustav. Korisnički račun nastaje postupkom registracije.
* **Preduvjeti:** -
* **Osnovni način izvršavanja:** Korisnik stvara novi korisnički račun. Kako bi stvorio novi korisnički račun, korisnik mora unijeti sljedeće podatke:
  + E-mail
  + Lozinka
  + Ponoviti unos lozinke

Ukoliko su uneseni podatci ispravni, sustav provodi registraciju i stvara novi korisnički račun u bazi podataka. Korisnik se zatim može prijaviti na sustav.

* **Alternativni načini izvršavanja:** Ukoliko je jedan ili više podataka neispravno, sustav neće provesti registraciju. Sustav će korisniku dojaviti da su uneseni podatci prilikom registracije neispravni, te će mu dojaviti točan razlog njihove neispravnosti (npr. lozinka i ponovno unesena lozinka se ne podudaraju). Korisnik zatim može ponoviti registraciju, naravno uz pretpostavku izmijene podataka.

### UC2: Prijava

* **Sažetak:** Korisnik se prijavljuje kako bi mogao započeti s korištenjem sustava
* **Obrazloženje:** Kako bi korisnik koristio funkcionalnosti sustava, on se **nužno mora prijaviti na sustav** uporabom **postojećeg korisničkog računa**.
* **Preduvjeti:** Uspješno proveden postupak registracije, odnosno u bazi podataka se nalazi korisnički račun kojim se korisnik prijavljuje na sustav (korisnički račun je ispravan i pohranjen u bazi podataka).
* **Osnovni način izvršavanja:** Korisnik se prijavljuje na sustav. Korisnik unosi sljedeće podatke kako bi se prijavio na sustav:
  + E-mail
  + Lozinka

Ukoliko su uneseni podatci ispravni (odgovaraju postojećem korisničkom računu), sustav provodi prijavu korisnika.

* **Alternativni način izvršavanja:** Ukoliko uneseni korisnički račun **ne postoji** ili je **krivo unesena lozinka**, sustav **ne provodi prijavu** i korisniku dojavljuje da su podatci za prijavu neispravni. Korisnik zatim može ponoviti prijavu na sustav.

### UC3: Odjava

* **Sažetak:** Odjava korisnika sa sustava.
* **Obrazloženje:** Nakon završetka rada sa sustavom, korisnik se može odjaviti sa sustava.
* **Preduvjeti:** Korisnik se uspješno prijavio na sustav.
* **Osnovni način izvršavanja**: Korisnik više nema potrebu za radom sa sustavom. Korisnik se odjavljuje sa sustava. Odjavom korisnik više ne može koristiti sustav. Kako bi nastavio koristiti sustav, može se nakon odjave, ponovno prijaviti na sustav.
* **Alternativni načini izvršavanja:** -

### UC4: Dodavanje novog kućnog ljubimca

* **Sažetak:** Korisnici dodaju novog kućnog ljubimca u sustav.
* **Obrazloženje:** Kako bi vodili evidenciju o težini, obrocima i aktivnosti, korisnici prvo **moraju dodati ljubimca** u sustav.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav.
* **Osnovni način izvršavanja:** Kako bi stvorio novog kućnog ljubimca, korisnik mora unijeti sljedeće podatke o ljubimcu:
  + Ime kućnog ljubimca
  + Spol
  + Datum rođenja
  + Slika kućnog ljubimca
  + Razina aktivnosti, pri čemu razina može biti:
    - Visoka – više od 60 minuta aktivnosti dnevno
    - Srednja – od 30 do 60 minuta aktivnosti dnevno
    - Niska – manje od 30 minuta aktivnosti dnevno
  + Stanje tijela (Reed, 2014), pri čemu stanje tijela može biti:
    - Vrijednost od 1 do 3 – pothranjenost
    - Vrijednost od 4 do 5 – optimalno stanje
    - Vrijednost od 6 do 7 – povećana težina
    - Vrijednost od 8 do 9 – pretilost
  + Jeli ljubimac kastriran/steriliziran ili nije
  + Vrsta ljubimca, pri čemu vrsta može biti:
    - Mačka
    - Pas
  + Trenutna težina ljubimca

Unosom ovih podataka se stvara novi kućni ljubimac u sustavu, a korisnik koji ga je stvorio se postavlja kao vlasnik novostvorenog ljubimca. Nakon što korisnik stvori novog ljubimca, sustav će mu na temelju unesenih podataka izračunati **optimalnu tjelesnu težinu** ljubimca, te dojaviti jeli njegov ljubimac pretio, premršav ili optimalne tjelesne težine. Također će mu sustav preporučiti koliko kalorija njegov novo uneseni ljubimac treba dnevno unositi, te koliko tjelesne aktivnosti dnevno treba dobivati. Sustav ove izračune temelji na unesenim podatcima o **stanju tijela**, **razini aktivnosti**, **trenutnoj težini** te o podatku jeli ljubimac **kastriran (steriliziran) ili nije**. Dnevna količina kalorija se izražava u kcal, a količina dnevne aktivnosti se izražava u minutama. Sustav automatski postavlja izračunate vrijednosti kao **ciljeve** za ljubimca, a svrha i značenje ciljeva je opisano u poglavlju *UC6.2 Postavljanje ciljeva*.

* **Alternativni način izvođenja:** Neki od unesenih podataka za ljubimca su neispavani. Sustav ne dodaje ljubimca ukoliko su neki od unesenih podataka neispravni. Sustav također ne izračunava ciljeve za ljubimca. Sustav korisnika obavještava da su neki od unesenih podataka neispravni. Korisnik može ponoviti postupak dodavanja kućnog ljubimca, uz izmjenu neispravnih podataka.

### UC5: Pregled podataka o ljubimcima

* **Sažetak:** Korisnik pregledava podatke o svojim ljubimcima.
* **Obrazloženje:** Korisnik može pregledati unesene podatke o svojim ljubimcima.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojeg želi pregledati podatke. Sustav korisniku prikazuje podatke o ljubimcu.
* **Alternativni tijek izvođenja:** -

### UC6.1: Izmjena podataka o ljubimcima

* **Sažetak:** Korisnici mogu izmijeniti podatke o ljubimcima.
* **Obrazloženje:** Ukoliko dođe do promijene podataka, korisnik u svakom trenutku može izmijeniti osnovne podatke o svojim ljubimcima koje je unio u sustav prilikom stvaranja ljubimca.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojega želi izmijeniti podatke. Korisnik može izmijeniti sve podatke o ljubimcu navedene u poglavlju *UC4 Dodavanje novog ljubimca*, **osim** podatka o **trenutnoj težini** ljubimca. Korisnik mijenja postojeće podatke o ljubimcu. Ukoliko su podatci ispravni, sustav pohranjuje promijene.
* **Alternativni tijek izvođenja:** Uneseni podatci o ljubimcu su neispravni. Sustav ne provodi i ne pohranjuje promijene. Sustav korisniku dojavljuje da su podatci neispravno uneseni. Korisnik može ponoviti izmjenu podataka.

### UC6.2: Postavljanje ciljeva

* **Sažetak:** Poseban slučaj *use case*-a *UC6.1 Izmjena podataka o ljubimcima*, jest izmjena podataka o postavljenim **ciljevima** za kućnog ljubimca. Korisnik mijenja ciljeve o težini, dnevnoj količini aktivnosti i dnevnoj količini obroka koje je sustav izračunao prilikom dodavanja novog ljubimca.
* **Obrazloženje:** Sustav prilikom dodavanja novog kućnog ljubimca računa i postavlja vrijednosti ciljeva težine, dnevne količine aktivnosti i dnevne količine unesenih kalorija. Sustav ove podatke izračunava i postavlja na temelju unesenih podataka o ljubimcima. Korisnik te ciljeve u svakom trenutku može izmijeniti.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojega želji izmijeniti postavljeni cilj. Korisnik unosi novu vrijednost odabranog cilja. Sustav provodi i pohranjuje promjene.
* **Alternativni tijek izvođenja:** Korisnik je unio neispravan podatak o cilju. Sustav mu dojavljuje da je unesen podatak neispravan. Sustav ne provodi izmjenu. Korisnik može ponoviti izmjenu cilja unosnom novog ispravnog podatka.

### UC7: Brisanje kućnog ljubimca

* **Sažetak:** Korisnik briše kućnog ljubimca.
* **Obrazloženje:** Korisnik može izbrisati svojeg kućnog ljubimca. Pritom se brišu svi podatci o kućnom ljubimcu iz sustava.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca kojega želi izbrisati. Sustav briše ljubimca i sve podatke o ljubimcu iz sustava. Sustav korisniku dojavljuje da je uspješno provedeno brisanje ljubimca.
* **Alternativni tijek izvođenja:** -

### UC8: Unos vaganja

* **Sažetak:** Korisnik za svoje ljubimce može unositi zapise o vaganju.
* **Obrazloženje:** Kako bi korisnik mogao voditi dnevnik težine svojih ljubimaca, on mora unositi zapise o težini ljubimaca.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac za kojeg se unosi zapis o vaganju, je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojega želi unijeti zapis o vaganju. Korisnik unosi težinu ljubimca (izraženu u kilogramima). Sustav stvara novi zapis i pohranjuje ga u bazi podataka. Sustav korisniku dojavljuje da je uspješno stvoren novi zapis. Sustav korisniku također dojavljuje jeli njegov ljubimac **dostignuo postavljen cilj težine** nakon novo unesenog zapisa o vaganju.
* **Alternativni tijek izvođenja:** Uneseni podatak o težini je neispravan. Sustav ne stvara novi podatak. Sustav korisniku dojavljuje da je uneseni podatak neispravan. Korisnik može ponoviti postupak unosa novog zapisa o vaganju.

### UC9: Unos obroka

* **Sažetak:** Korisnik može za svoje ljubimce unosit zapise o obrocima.
* **Obrazloženje:** Kako bi korisnik mogao voditi dnevnik obroka svojih ljubimaca, on mora unositi zapise o obrocima, odnosno unesenim kalorijama ljubimaca.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac za kojeg se unosi zapis o obroku je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojega želi unijeti zapis o obroku. Korisnik unosi količinu unesenih kalorija ljubimca (izraženu u kcal). Sustav stvara novi zapis i pohranjuje ga u bazi podataka. Sustav korisniku dojavljuje da je uspješno stvoren novi zapis. Sustav korisniku također dojavljuje podatak jeli njegov ljubimac **dostignuo postavljen dnevni cilj za obroke** nakon novo unesenog zapisa o obroku.
* **Alternativni tijek izvođenja:** Uneseni podatak o količini unesenih kalorija je neispravan. Sustav ne stvara novi podatak. Sustav korisniku dojavljuje da je uneseni podatak neispravan. Korisnik može ponoviti postupak unosa novog zapisa obroku.

### UC10: Unos aktivnosti

* **Sažetak:** Korisnik može za svoje ljubimce unosit zapise o tjelesnoj aktivnosti.
* **Obrazloženje:** Kako bi korisnik mogao voditi dnevnik aktivnosti svojih ljubimaca, on mora unositi zapise o tjelesnoj aktivnosti svojih ljubimaca.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav, ljubimac za kojeg se unosi zapis o aktivnosti, je uspješno dodan u sustav (podatci o ljubimcu se nalaze u bazi podataka).
* **Osnovni tijek izvođenja:** Korisnik odabire ljubimca za kojega želi unijeti zapis tjelesnoj aktivnosti. Korisnik unosi broj minuta aktivnosti i vrstu aktivnosti ljubimca. Sustav stvara novi zapis i pohranjuje ga u bazi podataka. Sustav korisniku dojavljuje da je uspješno stvoren novi zapis. Sustav korisniku također dojavljuje podatak jeli njegov ljubimac dostignuo postavljen dnevni cilj za aktivnost nakon novo unesenog zapisa o aktivnosti.
* **Alternativni tijek izvođenja:** Uneseni podatak o količini minuta aktivnosti je neispravan. Sustav ne stvara novi podatak. Sustav korisniku dojavljuje da je uneseni podatak neispravan. Korisnik može ponoviti postupak unosa novog zapisa o aktivnosti.

### UC11: Izmjena unesenih zapisa o vaganju, obroku i aktivnosti

* **Sažetak:** Korisnik unesene podatke o vaganju, obroku i aktivnosti za svoje ljubimce može izmijeniti. Pritom korisnik može izmijeniti samo **zadnje** unesene zapise o obroku, vaganju i aktivnosti za **svakog ljubimaca**.
* **Obrazloženje:** Ukoliko je korisnik prilikom unosa novog zapisa o vaganju, obroku ili aktivnosti unio **ispravan**, ali **netočan** podatak, on ga naknadno može izmijeniti.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav. Zapis koji korisnik želi izmijenit je uspješno stvoren i dodan u sustav. Zapis koji korisnik želi izmijeniti je **posljednje** dodan zapis za pojedinog ljubimca.
* **Osnovni tijek izvođenja:** Korisnik odabire jedan od zadnje unesenih zapisa. Korisnik unosi nove podatke. Ukoliko su uneseni podatci ispravni sustav provodi i pohranjuje izmjene zapisa. Sustav korisniku dojavljuje da je uspješno provedena izmjena zapisa.
* **Alternativni tijek izvođenja:** Korisnik je unio neispravan podatak. Sustav mu dojavljuje da je unesen podatak neispravan i ne provodi izmjenu zapisa.

### UC12: Brisanje unesenih zapisa o vaganju, obroku i aktivnosti

* **Sažetak:** Korisnik može izbrisati unesene zapise o vaganju, obrocima i aktivnosti, pritom korisnik može izbrisati samo **zadnje unesene zapise** za svakog ljubimca.
* **Obrazloženje:** Ukoliko je korisnik prilikom unosa novog zapisa o vaganju, obroku ili aktivnosti unio **ispravan**, ali **netočan** podatak, on može naknadno izbrisati uneseni zapis.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav. Zapis koji korisnik želi izmijenit je uspješno stvoren i dodan u sustav. Zapis koji korisnik izmijeniti je **posljednje dodan** zapis za pojedinog ljubimca.
* **Osnovni tijek izvođenja:** Korisnik odabire jedan od zadnje unesenih zapisa. Sustav provodi i briše odabrani zapis iz baze podataka. Sustav korisniku dojavljuje da je uspješno obrisan zapis.
* **Alternativni tijek izvođenja:** -

### UC13: Tablični i grafički prikaz uneseni podataka o vaganju, obroku i aktivnosti

* **Sažetak:** Korisnik pregledava unesene zapise o vaganju, obrocima i aktivnosti. Zapisi su mu prikazani u obliku grafa i u obliku tablice.
* **Obrazloženje:** Kako bi korisnik mogao pratiti težinu, količinu unesenih kalorija i količinu aktivnosti svojih ljubimaca, on može u svakom trenutku pregledati sve unesene zapise za svoje ljubimce. Korisnik može unesene zapise dodatno filtrirati po datumu kada su oni nastali. Korisnik unesene zapise može pregledati u obliku grafa i u obliku tablice, što mu omogućuje da prati trend težine, količine unesenih kalorija i količine aktivnosti.
* **Preduvjeti:** Korisnik je uspješno prijavljen na sustav. Zapisi koje korisnik želi pregledati su uspješno uneseni u sustav.
* **Osnovni tijek izvođenja:** Korisnik odabire vrstu zapisa, ljubimca i interval datuma za koje želi pregledati zapise. Sustav dohvaća zapise odabrane vrste u odabranom vremenskom intervalu za određenog ljubimca. Sustav prikazuje zapise u obliku grafa i u obliku tablice.
* **Alternativni tijek izvođenja:** -

## Specifikacija API sučelja

U prošlom poglavlju su detaljno opisani funkcionalni zahtjevi registriranog korisnika, jedinog dionika sustava. Kako bi bilo moguće programski ostvariti ranije opisane funkcionalne zahtjeve, potrebno je izgraditi API sučelje koje će podržati korisnikove zahtjeve. U nastavku poglavlja je opisano API sučelje kojem je moguće ostvariti ranije navedene funkcionalne zahtjeve. Navedeno sučelje je moguće stvarno implementirati i ostvariti kroz mnoge tehnologije, te će se u nastavu dati implementacijski neovisna specifikacija API sučelja. Također, sama organizacija navedenih metoda po modulima i komponentama može biti različita ovisno o vrsti tehnologije i arhitekturnom stilu aplikacije.

### Registracija

* **Naziv metode:** Register
* **Opis:** Metoda koja provodi registraciju. Stvara novi korisnički račun u bazi podataka na temelju unesenih podataka.
* **Ulazni parametri:** E-mail i lozinka za korisnički račun koji se stvara.
* **Izlazni parametri:** Status koji označava jeli uspješnost registracije.

### Prijava

* **Naziv metode:** Login
* **Opis:** Metoda koja provodi prijavu korisnika na sustav. Provjerava postoji li korisnički račun s unesenim podatcima. Ukoliko postoji, provodi prijavu, ukoliko ne dojavljuje korisniku grešku.
* **Ulazni parametri:** E-mail i lozinka za prijavu na sustav.
* **Izlazni parametri:** Status koji označava uspješnost prijave.

### Odjava

* **Naziv metode:** Logout
* **Opis:** Metoda koja odjavljuje korisnika sa sustava.
* **Ulazni parametri:** E-mail korisnika koji se odjavljuje sa sustava.
* **Izlazni parametri:** Status koji označava uspješnost odjave.

### Dodavanje novog kućnog ljubimca

* **Naziv metode:** CreatePet
* **Opis:** Metoda koja stvara novog kućnog ljubimca. Metoda koja na temelju unesenih podatka stvara novog ljubimca i pohranjuje ga u bazu podataka.
* **Ulazni parametri:** podatci koji su potrebni da bi se stvorio novi ljubimac, detaljno opisani u poglavlju *UC4 Dodavanje novog kućnog ljubimca.*
* **Izlazni parametri:** Status koji označava jeli novi ljubimac uspješno stvoren ili nije.

### Pregled podataka o ljubimcima

* **Naziv metode:** DetailsPet
* **Opis:** Metoda dohvaća podatke o ljubimcima i baze podataka.
* **Ulazni parametri:** Ljubimac za koje korisnik želi pregledati podatke.
* **Izlazni parametri:** Svi pohranjeni podatci o ljubimcu u bazi podataka.

### Izmjena podataka o ljubimcima i postavljanje ciljeva

* **Naziv metode:** EditPet
* **Opis:** Metoda na temelju unesenih podataka od strane korisnika mijenja i pohranjuje izmijene podataka o ljubimcima u bazi podataka. Ovom metodom korisnici mijenjaju i **postavljene ciljeve** za pojedinog ljubimca
* **Ulazni parametri:** Ljubimac kojemu korisnik mijenja podatke, podatci koje korisnik želi izmijeniti (nova vrijednost podataka).
* **Izlazni parametri:** Status koji označava uspješnost izmijene podataka.

### Brisanje kućnog ljubimca

* **Naziv metode:** DeletePet
* **Opis:** Metoda koja iz baze podataka briše određenog ljubimca.
* **Ulazni parametri:** Ljubimac kojega korisnik želi izbrisati.
* **Izlazni parametri:** Status koji označava uspješnost postupka brisanja ljubimca.

### Unos vaganja

* **Naziv metode:** CreateWeightEntry
* **Opis:** Metoda koja na temelju unesenih podataka stvara novi zapis o vaganju u bazi podataka za određenog ljubimca.
* **Ulazni parametri:** Ljubimac za kojega se stvara zapis, podatci o vaganju.
* **Izlazni parametri:** Status koji označava uspješnost postupka stvaranja zapisa o vaganju.

### Unos obroka

* **Naziv metode:** CreateMealEntry
* **Opis:** Metoda koja na temelju unesenih podataka stvara novi zapis o obroku u bazi podataka za određenog ljubimca
* **Ulazni parametri:** Ljubimac za kojega se stvara zapis, podatci o obroku.
* **Izlazni parametri:** Status koji označava uspješnost postupka stvaranja zapisa o obroku.

### Unos aktivnosti

* **Naziv metode:** CreateActivityEntry
* **Opis:** Metoda koja na temelju unesenih podataka stvara novi zapis o aktivnosti u bazi podataka za određenog ljubimca .
* **Ulazni parametri:** Ljubimac za kojega se stvara zapis, podatci o aktivnosti.
* **Izlazni parametri:** Status koji označava uspješnost postupka stvaranja zapisa o aktivnosti.

### Izmjena unesenih zapisa o vaganju

* **Naziv metode:** EditWeightEntry
* **Opis metode:** Metoda koja na temelju unesenih podataka od strane korisnika mijenja podatke o zapisu o vaganju, te provodi i sprema promjenu u bazi podataka.
* **Ulazni parametri:** Zapis kojemu se mijenjaju podatci, podatci koje korisnik želi izmijeniti.
* **Izlazni parametri:** Status koji označava uspješnost postupka izmjene podataka o zapisu vaganja.

### Izmjena unesenih zapisa o obroku

* **Naziv metode:** EditMealEntry
* **Opis metode:** Metoda koja na temelju unesenih podataka, od strane korisnika, mijenja podatke zapisa o obroku, te provodi i sprema promjene u bazi podataka.
* **Ulazni parametri:** Zapis kojemu se mijenjaju podatci, podatci koje korisnik želi izmijeniti.
* **Izlazni parametri:** Status koji označava uspješnost postupka izmjene podataka o zapisu obroka.

### Izmjena unesenih zapisa o aktivnosti

* **Naziv metode:** EditActivityEntry
* **Opis metode:** Metoda koja na temelju unesenih podataka, od strane korisnika, mijenja podatke zapisa o aktivnosti, te provodi i sprema promjene u bazi podataka.
* **Ulazni parametri:** Zapis kojemu se mijenjaju podatci, podatci koje korisnik želi izmijeniti.
* **Izlazni parametri:** Status koji označava uspješnost postupka izmjene podataka o zapisu aktivnosti.

### Brisanje zapisa o vaganju

* **Naziv metode:** DeleteWeightEntry
* **Opis metode:** Metoda koja iz baze podataka briše određeni zapis o vaganju.
* **Ulazni parametri:** Zapis koji korisnik želi obrisati.
* **Izlazni parametri:** Status koji označava uspješnost postupka brisanja zapisa o vaganju.

### Brisanje zapisa o obroku

* **Naziv metode:** DeleteMealEntry
* **Opis metode:** Metoda koja iz baze podataka briše određeni zapis o obroku.
* **Ulazni parametri:** Zapis koji korisnik želi obrisati.
* **Izlazni parametri:** Status koji označava uspješnost postupka brisanja zapisa o obroku.

### Brisanje zapisa o aktivnosti

* **Naziv metode:** DeleteActivityEntry
* **Opis metode:** Metoda koja iz baze podataka briše određeni zapis o aktivnosti.
* **Ulazni parametri:** Zapis koji korisnik želi obrisati.
* **Izlazni parametri:** Status koji označava uspješnost postupka brisanja zapisa o aktivnosti.

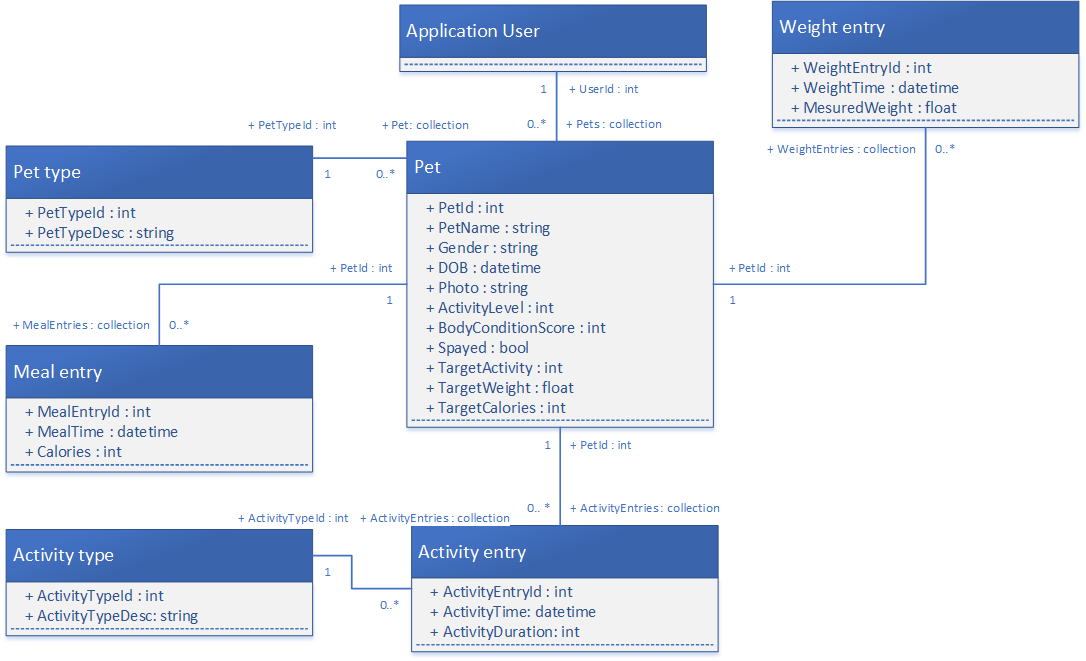
### Tablični i grafički prikaz uneseni podataka o vaganju, obroku i aktivnosti

* **Naziv metode:** GetEntries
* **Opis metode:** Metoda koja na temelju vremenskog intervala, odabranog ljubimca i odabrane vrste zapisa dohvaća zapise iz baze podataka koji zadovoljavaju ulazne parametre.
* **Ulazni parametri:** Vrsta zapisa, vremenski interval, ljubimac za kojega se prikazuju zapisi.
* **Izlazni parametri:** Lista zapisa koja zadovoljava ulazne parametre.

# Model domene

Model domene se sastoji od razreda kojim su modelirani koncepti iz stvarnog svijeta nužni za rad sustava. Bez ovih razreda ne bi bilo moguće ostvariti korisničke zahtjeve i samim time sustav ne bi imao nikakvu svrhu. U nastavku poglavlja je dan opis i dijagram razreda modela domene.

## Dijagram razreda



Slika 1: Dijagram razreda modela domene

## Opis dijagrama razreda

U nastavku je dan kratak opis dijagrama razreda prikazanog u prethodnom poglavlju.

Kako je ***Chubster*** aplikacija za praćenje tjelesne težine, aktivnosti i obroka **kućnih ljubimaca**, temeljni razred modela domene ovog sustava je razred ***Pet*** – razred kojim je modeliran koncept **kućnog ljubimca**. Ljubimci imaju svoje atribute – naziv, spol, datum rođenja, ciljeve (ciljana težina, ciljana dnevna količina aktivnosti i ciljana dnevna količina unesenih kalorija), sliku itd. Uz to, ljubimci imaju i vrstu koja je modelirana kroz razred ***Pet type***. Ovim razredom je opisana vrsta ljubimca koja može biti mačka ili pas (u trenutnoj verziji aplikacija). Ovakvim modelom je omogućeno jednostavno proširenje aplikacije i za druge vrste ljubimaca – u budućnosti bi tako korisnici mogli pratiti podatke i ostalim vrstama ljubimaca.

Uz to, ljubimac je vezan uz pojedinog korisnika – njegovog vlasnika. Vlasnik je korisnik aplikacije. Modeliran je razredom ***Application User***. Uz listu ljubimca, razred koji modelira korisnika sadrži osnovne podatke za pristup aplikaciji – **e-mail i kriptiranu lozinku**. *Application User* je u ovom sustavu ostvaren kao izveden razred razreda *Asp.Net.Core IdentityUsers* (Anderson, 2020). Ovaj razred je dio API-ja *Asp.Net.Core Identity* kojim je ostvareno upravljanje lozinkama, korisnicima, ulogama u sustavu itd. Zbog toga je na dijagramu razreda prikazan bez atributa, odnosno samo je prikazana veza između razreda *Pet* i *Application Users*. Svi ostali atributi i metode su metode i atributi razreda *Asp.Net.Core IdentityUsers* koji se mogu pronaći na sljedećim stranicama (Microsoft, 2020), te zbog toga nisu prikazani na dijagramu razreda, te se neće spominjati i obrađivati u daljnjem tekstu.

Praćenje tjelesne težine, aktivnosti i obroka ne bi bilo moguće bez razreda *Weight entry, Activity entry* i *Meal entry*. Ovim razredima su modelirani koncepti **zapisa** težine, aktivnosti i obroka. Svaki od zapisa sadrži vrijeme nastanka, ljubimca za kojega je vezan, te zabilježenu težinu, količinu aktivnosti ili količinu unesenih kalorija.

Uz vrijeme nastanka zapisa, količine aktivnosti u minutama, ljubimca i jedinstvenog identifikatora, zapis o aktivnosti (*Activity entry*) sadrži i vrstu aktivnosti – *Activity type*. Razredom *Activity type* je modelirana vrsta aktivnosti, koja može biti trčanje, igra itd.

# Implementacija

U nastavku poglavlja je dan prikaz implementacije sustava **Chubster**. Sustav **Chubster** je implementiran kao desktop i web aplikacija. Desktop aplikacija služi ako **pomoćna aplikacija** web aplikaciji, odnosno desktop aplikacija je implementirana kao **administratorsko sučelje**.

U nastavku poglavlja je prvo opisana podatkovna razina. Podatkovna razina sustava *Chubster* je ostvarena kao **relacijska baza podataka**. Ova razina sustava trajno čuva sve nužne podatke za ispravan rad sustava. Obje aplikacije koriste istu relacijsku bazu podataka.

S druge strane, poslovna logika i poslovni sloj obje aplikacije (*backend*) se oslanja na objektno-orijentiranu paradigmu. Cjelokupni kod poslovne logike obje aplikacije je organiziran u **razrede**. Razredi modeliraju koncepte iz stvarnog svijeta – točnije modeliraju koncepte iz domene problema. Razred sadrži i modelira podatke, te ponašanje entiteta iz stvarnog svijeta.

**Relacije** i **razredi** su dva koncepta koja je potrebno na neki način međusobno povezati. Dok relacije predstavljaju činjenice o povezanosti podataka, razredi su reprezentacije koncepata iz stvarnosti. Potrebno je na neki način ostvariti preslikavanje iz razreda u relaciju i obrnuto. Time je omogućeno povezivanje podatkovnog sloja u kojem se trajno čuvaju podatci i poslovnog sloja koji radi s objektima i izvršava poslovnu logiku nad njima.

Povezivanje ovih koncepata - relacije i razreda, je ostvareno objektno – relacijskim mapiranjem. Objektno – relacijsko mapiranje je tehnika kojom se ostvaruje automatska transformacija iz jedne u drugu reprezentaciju – iz relacije u razred i obrnuto. Time je omogućen rad s bazom podatka preko objekata.

U nastavku poglavlja je prvo opisan način na koji je ostvareno objektno – relacijsko mapiranje u sustavu *Chubster*. Zatim je prikazana baza podataka sustava kroz relacijski dijagram baze podataka.

Na kraju poglavlja je dan prikaz izrađene web i desktop aplikacije. Dana je struktura aplikacija, način na koji su pojedini dijelovi aplikacija povezani u cjelinu, te prikaz implementiranih korisničkih zahtjeva.

## Implementacija podatkovne razine i objektno – relacijsko mapiranje

U ovom poglavlju je opisana podatkovna razina sustava, te način na koji je podatkovna razina povezana s razinom poslovne logike (*backend*). Ova dva sloja su međusobno povezana objektno-relacijskom tehnikom.

Objektno-relacijsko mapiranje je ostvareno kroz sljedeća četiri koraka:

* Implementacija modela domene
* Definiranje pravila mapiranja
* Definiranje konfiguracije baze podataka
* Stvaranje baze podataka, te obavljanje osnovnih operacija nad bazom podataka

U desktop i web aplikaciji sustava *Chubster*, objektno-relacijsko mapiranje je ostvareno korištenjem radnog okvira *Entity Framework Core*. U nastavku poglavlja je dan detaljan prikaz načina na koji je ostvareno objektno – relacijsko mapiranje, kroz četiri navedena koraka. U poglavlju u kojem se opisuje stvaranje i rad s bazom podataka je također prikazan i relacijski model baze podatka sustava *Chubster*.

### Implementacija modela domene

Dijagram razreda modela domene, prikazan u poglavlju *Model domene*, je potrebno programski implementirati kako bi se prikazani razredi mogli koristiti unutar sustava. Ovo je ujedno i prvi korak objektno – relacijskog mapiranja – implementacija razreda iz modela domene. Iz dijagrama razreda je jednostavno i intuitivno implementirati razrede unutar sustava – atributi iz dijagrama razreda se preslikavaju u atribute razreda, a asocijacije su modelirane kao **reference** na pojedine objekte. Objektni model je ostvaren u programskom jeziku C#. U nastavku je detaljno opisano programsko ostvarenje razreda *Pet*. Ostali razredi iz modela domene su implementirani na jednak način, te se zbog toga njihova implementacija neće detaljno razmatrati u nastavku teksta.

#### Implementacija zareda Pet – kućnog ljubimca

public Pet()

{

MealEntries = new HashSet<MealEntry>();

WeightEntries = new HashSet<WeightEntry>();

ActivityEntries = new HashSet<ActivityEntry>();

}

[Key]

public int PetId { get; set; }

[Required]

[DisplayName("Pet name")]

[MaxLength(100, ErrorMessage = "Name is too long.")]

public string PetName { get; set; }

[Required]

[DisplayName("Gender")]

[MaxLength(50, ErrorMessage = "Gender is too long.")]

public string Gender { get; set; }

[Required]

[DisplayName("DOB")]

public DateTime DOB { get; set; }

public string Photo { get; set; }

[Required]

[Range(1, 3, ErrorMessage = "Invalid activity level input")]

public int ActivityLevel { get; set; }

[Required]

[DisplayName("Body condition score")]

[Range(1, 9, ErrorMessage = "Invalid BCS input")]

public int BodyCondtionScore { get; set; }

[Required]

[DisplayName("Spayed")]

public bool Spayed { get; set; }

[DisplayName("Daily activity goal")]

[Range(0, 500, ErrorMessage = "Activity goal is invalid! Must be in range {0, 500}")]

public int TargetActivity { get; set; }

[DisplayName("Daily calorie goal")]

[Range(0, 3000, ErrorMessage = "Daily calorie goal is invalid! Must be in range {0, 3000}")]

public int TargetCalories { get; set; }

[DisplayName("Weight goal")]

[Range(0, 200, ErrorMessage = "Weight goal is invalid! Must be in range {0, 200}")]

public float TargetWeight { get; set; }

public string UserId { get; set; }

public ApplicationUser User { get; set; }

[Required]

[DisplayName("Pet type")]

public int PetTypeId { get; set; }

public virtual PetType PetType { get; set; }

public virtual ICollection<MealEntry> MealEntries { get; set; }

public virtual ICollection<WeightEntry> WeightEntries { get; set; }

public virtual ICollection<ActivityEntry> ActivityEntries { get; set; }

}

Razred *Pet* modelira kućnog ljubimca. Atributi ljubimca su modelirani kao atributi razreda s pripadajućim svojstvima. Tako svaki atribut ima javnu metodu za dohvaćanje i promjenu vrijednosti atributa – svoj *getter* i *setter*. Osim toga, ljubimac ima jednu vrstu i jednog vlasnika – korisnika. Ove asocijacije su ostvarene na način da ljubimac ima referencu na razred *Application User*, te referencu na razred *Pet type*. Atributi PetTypeId i UserId modeliraju **strane ključeve**, te su nužni za ostvarenje mapiranja razreda u relacijsku bazu podataka. Svaki ljubimac ima više zapisa o težini, obrocima i aktivnosti. Zato svaki ljubimac ima **kolekciju** objekata *Activity entry*, *Meal entry* i *Weight entry*. Time je ostvarena asocijacija između ljubimca i pojedine vrste zapisa.

Razred Pet ima konstruktor kojim se inicijaliziraju i stvaraju odgovarajući objekti za čuvanje kolekcija zapisa.

Metapodatci o pojedinom atributu su opisani korištenjem .NET atributa (Microsoft, 2018). Ovim atributima je moguće definirati metapodatke koji će se naknadno koristiti pri mapiranju razreda u bazu podataka. Tako atribut [Key] označava da je navedeni podatak – atribut primarni ključ, te da ga tako treba i mapirati u relacijsku bazu podataka – kao primarni ključ relacije. Atribut [Required] označava da je pojedini atribut obavezan i da ne smije biti ostavljen kao NULL prilikom stvaranja ili izmjene objekta – mora imati određenu postavljenu vrijednost. Atribut [DisplayName] određuje ime atributa koje će se prikazivati korisnicima. Atribut [MaxLength] određuje maksimalnu duljinu tekstualnog podatka – string koji se sprema u atribut može sadržavati najviše max lenght znakova. Atribut [Range] određuje valjan raspon cijelih brojeva koji se može pohraniti u bazu.

### Konfiguracija i mapiranje

Nakon što je implementiran model domene, potrebno je definirati metapodatke o mapiranju. Radni okvir za objektno – relacijsko mapiranje mora dobiti podatke o tome kako definirane razrede treba preslikati u relacije. Metapodatci radnom okviru daju upute o imenima relacija, o imenima stupaca za svaki pojedini atribut razreda, te opisuju odnose među razredima. Iz navedenih metapodataka radni okviri mogu generirati relacijsku bazu podataka, te odgovarajuće SQL upite.

Radni okvir *Entity Framewrok Core* koristiti posebnu klasu za definiciju metapodataka o mapiranju, koja nasljeđuje osnovnu klasu *IdentityDbContext<TUser>*, gdje je TUser vrsta korisnika aplikacije. Kako je u ovoj aplikaciji korisnik aplikacije *Application User*, za definiciju metapodataka o mapiranju se koristiti klasa *ApplicationDbContext* koja nasljeđuje razred *IdentitiyDbContext<ApplicationUser>*. IdentityDbContext<TUser> je poseban izveden razred iz razreda *DbContext*, koji omogućava rad s korisnicima aplikacije – omogućava rad s lozinkama, ulogama korisnika itd. U nastavku je prikazana implementacija *ApplicationDbContext* razreda.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public virtual DbSet<ActivityEntry> ActivityEntries { get; set; }

public virtual DbSet<ActivityType> ActivityTypes { get; set; }

public virtual DbSet<MealEntry> MealEntries { get; set; }

public virtual DbSet<Pet> Pets { get; set; }

public virtual DbSet<PetType> PetTypes { get; set; }

public virtual DbSet<WeightEntry> WeightEntries { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

var config = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

.Build();

optionsBuilder.UseSqlServer(config.GetConnectionString("DefaultConnection"));

}

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Entity<ActivityEntry>(entity =>

{

entity.HasKey(e => e.ActivityEntryId);

entity.Property(e => e.ActivityEntryId).ValueGeneratedOnAdd();

entity.HasOne(p => p.Pet)

.WithMany(a => a.ActivityEntries)

.HasForeignKey(p => p.PetId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(p => p.ActivityType)

.WithMany(a => a.ActivityEntries)

.HasForeignKey(p => p.ActivityTypeId)

.OnDelete(DeleteBehavior.Cascade);

});

builder.Entity<ActivityType>(entity =>

{

entity.HasKey(e => e.ActivityTypeId);

entity.Property(e => e.ActivityTypeId).ValueGeneratedOnAdd();

});

builder.Entity<MealEntry>(entity =>

{

entity.HasKey(e => e.MealEntryId);

entity.Property(e => e.MealEntryId).ValueGeneratedOnAdd();

entity.HasOne(p => p.Pet)

.WithMany(a => a.MealEntries)

.HasForeignKey(p => p.PetId)

.OnDelete(DeleteBehavior.Cascade);

});

builder.Entity<Pet>(entity =>

{

entity.HasKey(e => e.PetId);

entity.Property(e => e.PetId).ValueGeneratedOnAdd();

entity.HasOne(p => p.PetType)

.WithMany(a => a.Pets)

.HasForeignKey(p => p.PetTypeId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(p => p.User)

.WithMany(u => u.Pets)

.HasForeignKey(p => p.UserId)

.OnDelete(DeleteBehavior.Cascade);

});

builder.Entity<PetType>(entity =>

{

entity.HasKey(e => e.PetTypeId);

entity.Property(e => e.PetTypeId).ValueGeneratedOnAdd();

});

builder.Entity<WeightEntry>(entity =>

{

entity.HasKey(e => e.WeightEntryId);

entity.Property(e => e.WeightEntryId).ValueGeneratedOnAdd();

entity.HasOne(p => p.Pet)

.WithMany(a => a.WeightEntries)

.HasForeignKey(p => p.PetId)

.OnDelete(DeleteBehavior.Cascade);

});

}

}

Razred poziva konstruktor razreda kojega nasljeđuje. Atributi tipa *DbSet* predstavljaju kolekciju svih entiteta u kontekstu, ili svih entiteta koji mogu biti dohvaćeni iz baze podataka, određenog tipa. U metodi *OnConfiguring* se definiraju konfiguracijski podatci. Ovdje se radnom okviru daju upute o tome gdje se nalazi baza podataka (*Connection string*), koju vrstu sustava za upravljanje bazama podataka treba koristiti (u primjeru ovog sustava se koristi SQL Server sustav za upravljanje bazama podataka). Podatak o lokaciji baze podataka se dohvaća iz konfiguracijske datoteke appsettings.json. U *Entity Framework Core*-u su podatci o mapiranju i konfiguraciji baze podataka povezani, te se **nalaze unutar istog razreda**.

U metodi *OnModelCreating* se definiraju metapodatci o tome kako pojedini razred preslikati u relaciju. Tako su definirani ključevi pojedinih relacija. U ovoj metodi su definirani primarni i strani ključevi, te odnosi između pojedinih razreda. Također je definirano da se vrijednost primarnog ključa generira prilikom dodavanja novog zapisa u relaciju. Uz definirane metapodatke u ovoj metodi*, Entity Framework Core* također koristi i .NET atribute prikazane u prethodnom poglavlju za potpunu definiciju i uputu mapiranja pojedinog razreda u relaciju.

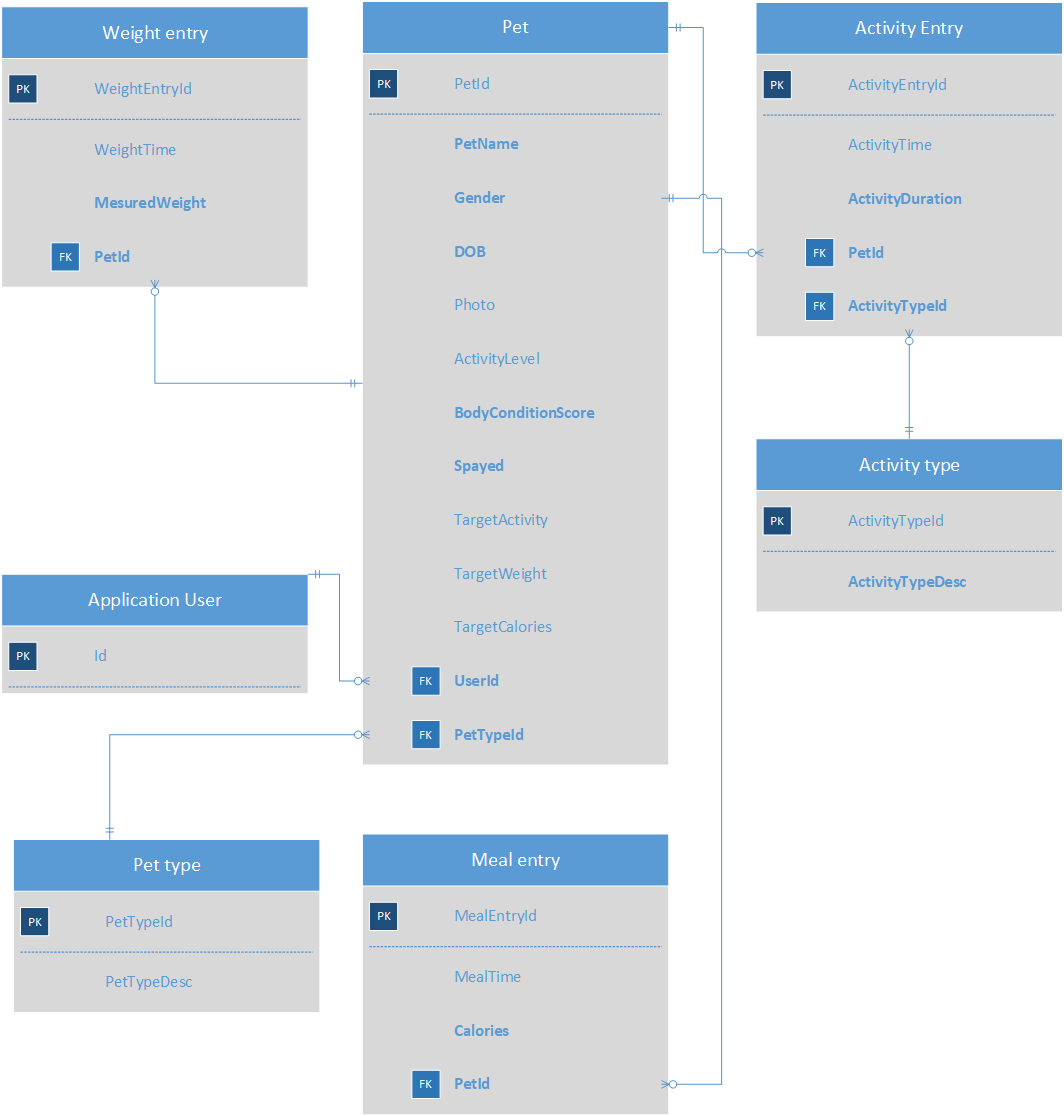
### Rad s bazom podataka

Nakon što su definirani podatci o mapiranju i konfiguraciji baze podataka, moguće je stvoriti relacijsku bazu podataka, te zatim preko objekata određenih razreda izvršavati osnovne operacije čitanja, pisanja i brisanja u bazu podataka.

#### Relacijski model baze podatka

U nastavku je prikazan relacijski model baze podataka sustava **Chubster**. Baza podataka je nastala izvršavanjem sljedećih naredbi u *Package Manager* konzoli unutar *Visual studio*-a:

* Add-Migration InitialCreate
* Update-Database



Slika 2: Relacijski model baze podataka sustava Chubster

#### Osnovne operacije nad bazom podataka

Nakon što je baza podataka stvorena, moguće je nad bazom izvršavati osnovne operacije čitanja, brisanja i pisanja podataka. Umjesto ručnog pisanja i izvršavanja SQL naredbi, ove operacije je moguće obaviti uporabom odgovarajućih metoda objektno-relacijskog radnog okvira nad određenim objektima. Time je rad s bazom podataka u potpunosti približen objektno-orijentiranoj paradigmi, osnovne operacije nad bazom podataka su izvedene kao pozivi odgovarajućih metoda nad pojedinim objektima.

U *Entity Framework Core*-u su osnovne operacije ostvarene kao metode nad instancom razreda *ApplicationDbContext*. Potrebno je stvoriti instancu ovog razreda, te je zatim pozivom odgovarajućih metoda moguće ostvariti izvršavanje pojedinih operacija nad bazom podataka. U aplikaciji *Chubster* su osnovne operacije dodavanja, brisanja, izmijene i čitanja ljubimca ostvarenje na sljedeći način:

private readonly ApplicationDbContext \_dbContext;

public async Task<Pet> ReadPetDetailsAsync(int petId)

{

return await \_dbContext.Pets

.Include(d => d.PetType)

.SingleOrDefaultAsync(p => p.PetId == petId);

}

public async Task<bool> CreatePetAsync(Pet pet, float weight)

{

var user = await GetPetUserAsync(pet.UserId);

pet.User = user;

pet.TargetActivity = CalculateTargetActivity(pet);

pet.TargetCalories = CalculateTargetCalories(pet, weight);

pet.TargetWeight = CalculateTargetWeight(pet, weight);

await \_dbContext.Pets.AddAsync(pet);

var created = await \_dbContext.SaveChangesAsync();

return created > 0;

}

public async Task<bool> UpdatePetAsync(Pet petToUpdate)

{

\_dbContext.Pets.Update(petToUpdate);

var updated = await \_dbContext.SaveChangesAsync();

return updated > 0;

}

public async Task<bool> DeletePetAsync (int petId)

{

var pet = await \_dbContext.Pets.SingleOrDefaultAsync(p => p.PetId == petId);

if (pet == null)

{

return false;

}

\_dbContext.Pets.Remove(pet);

var deleted = await \_dbContext.SaveChangesAsync();

return deleted > 0;

}

Tako je na primjer, za dohvaćanje svih podataka o ljubimcu korištena metoda *SingleOrDefaultAsync*, kojoj je kao argument predan lambda izraz koji određuje kriteriji filtriranja. Tako se dohvaćaju samo ljubimci s određenom vrijeđanošću atributa PetId. Metoda je izvršena nad atributom *DBSet* koji sadrži kolekciju ljubimaca instance razreda *ApplicationDbContext*. Prije rada s bazom podataka je dakle **potrebno stvoriti instancu razreda** *ApplicationDbContext*, nad kojom će se, pozivom odgovarajućih metoda obavljati osnovne operacije nad bazom podataka.

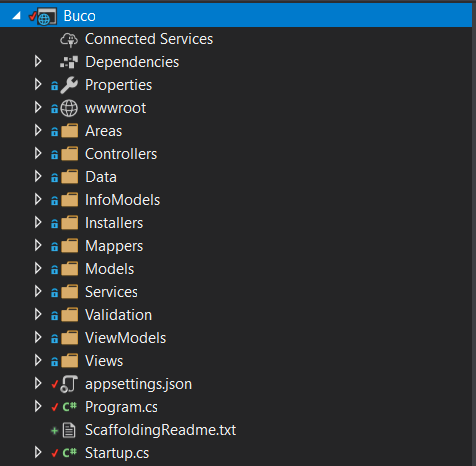
## Implementacija web aplikacije

U ovom poglavlju je opisana implementacija **web aplikacije** sustava ***Chubster***. Prvo je dan opis organizacije repozitorija, te način na koji su pojedini dijelovi aplikacije međusobno povezani. Zatim je prikazan način na koji su ključni funkcionalni zahtjevi iz poglavlja *Funkcionalni zahtjevi* ostvareni.

### Organizacija repozitorija i ostvarenje aplikacije

Web aplikacija sustava Chubster je ostvarena kao MVC .NET Core web aplikacija. Za implementaciju aplikacije je dakle korišteno radno okruženje .NET Core verzije 3.1. Aplikacija slijedi MVC strukturu, odnosno pogledi, kontroleri i modeli su jasno okruženi s jasnim odvajanjem odgovornosti. Aplikacija je implementirana u C# programskom jeziku. U nastavku je dan prikaz organizacije repozitorija web aplikacije. U svakom od prikazanih repozitorija se nalaze datoteke s izvornim kodom aplikacije. Dakle, izvorni kod aplikacije je podijeljen u repozitorije, pri čemu datoteke s izvornim kodom unutar pojedinog repozitorija imaju istu određenu ulogu u sustavu.

U nastavku je prikazana organizacija repozitorija web aplikacije sustava Chubster.



Slika : Organizacija repozitorija web aplikacije Chubster

U nastavku je dan opis pojedinih repozitorija prikazanih na slici 3.

Wwwroot: Sadrži statičke dokumente, poput slika, datoteka s JavaScript izvornim kodom, datoteka s CSS kodom itd.

Areas: Repozitorij sadrži MVC poglede za rad s korisnicima. Kako je rad s korisnicima ostvaren pomoću Asp.Net.Core IdentityUser-a, sustav koristi unaprijed stvorene poglede za prijavu i registraciju na sustav.

Controllers: Sadrži kontrolere, odnosno razrede koji ostvaruju sloj poslovne logike web aplikacije (backend). Kontroleri primaju korisničke zahtjeve, obrađuju ih i vraćaju korisnicima odgovarajuće odgovore.

Data: U ovom repozitoriju se nalazi ApplicationDbContext razred koji služi kao izvor metapodataka o mapiranju i konfiguraciji baze podataka.

InfoModels: Pomoćni razredi koji na koje se oslanjaju kontroleri. Sadrže implementacijske detalje o straničenju.

Installers: Posebni razredi koji osiguravaju registraciju odgvarajućih servisa pomoću Dependecy Injectiona.

Mappers: Razredi za preslikavanje modela u view models. Modele iz modela domene preslikavaju u view modele koje kontroleri šalju pogledima.

Models: Razedi koji implementiraju model domene.

Services: Servisi, posebni razredi koji služe za rad s bazom podataka. Obavljaju osnovne operacije nad bazom podataka, te vraćaju rezultate operacija kontrolerima.

Validation: Razredi i metode za validaciju podataka.

ViewModels: Modeli koje kontroleri predaju pogledima. Pogledi ne rade s modelima iz modela domene, već njihovim posebnim reprezentacijama. U ovom repozitoriju se nalaze view model-i s kojima rade pogledi.

Views: Pogledi kojima je ostvareno korisničko sučelje (frontend).

Web aplikacija Chubster je dakle izrađena kao MVC aplikacija. U osnovnu MVC strukturu su dodani pomoćni repozitoriji i razredi koji omogućuju podjelu posla i rasterećenje kontrolera. Tako kontroleri ne rade izravno s bazom podataka, već da bi pristupili podatcima u bazi pozivaju odgovarajuće metode razreda iz repozitorija Services.

Grafičko sučelje je ostvareno kroz .cshtml datoteke koje se nalaze u repozitoriju Views. Određene datoteke iz ovog repozitorija korisnicima prikazuje njihov web preglednik. Korisnik interakcijom s pogledima šalje određene zahtjeve kontrolerima.

Kontroleri, dakle prihvaćaju korisničke zahtjeve koje im na obadu šalju pogledi. Korisnici tako mogu tražiti prijavu na sustav, stvaranje novog korisničko računa, stvaranje novog ljubimca, odnosno mogu zahtijevati da sustav za njih obave sve ono što je opisano u poglavlju Funkcionalni zahtjevi.

Kada kontroler primi korisnički zahtjev, prvo mora ispitati njegovu ispravnost. Kontroler se oslanja na validacijske metode razreda iz repozitorija Validation. Nakon što je ispitana ispravnost zahtjeva, ukoliko je zahjtev ispravan, kontroler poziva metode iz repozitorija Services kako bi obavio odgvarajuću metodu nad bazom podataka.

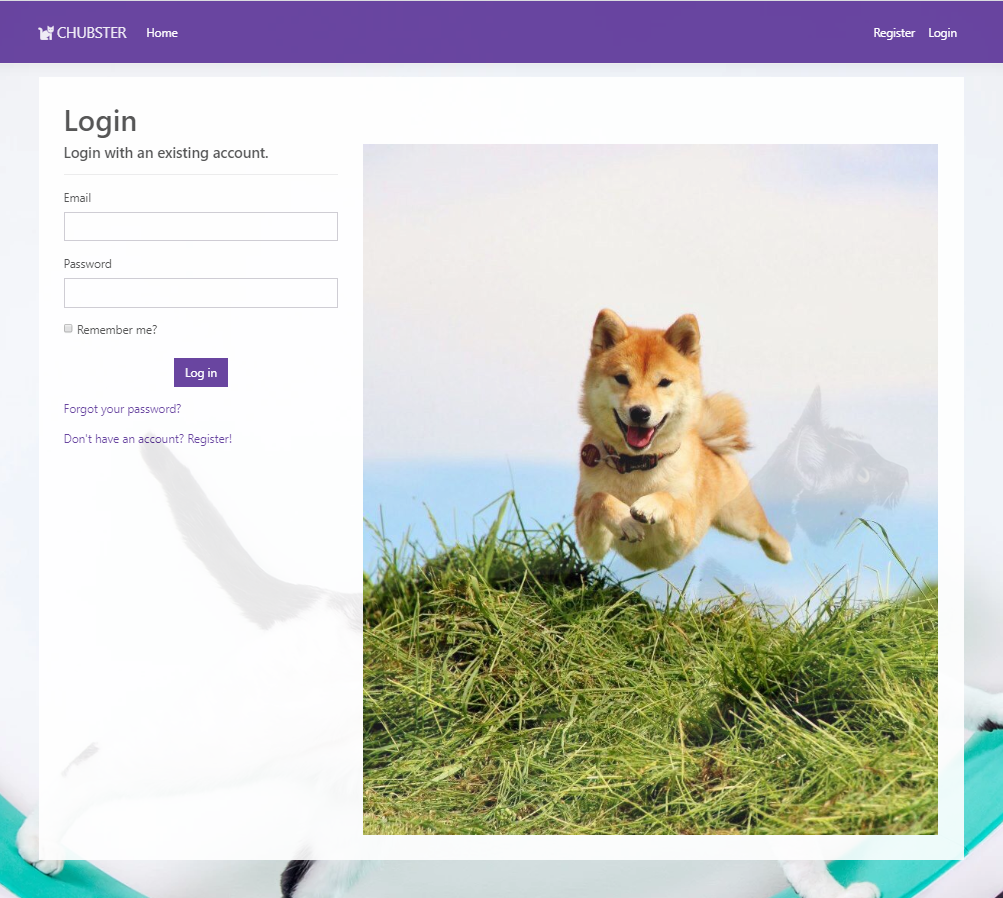
Razredi iz repozitorija Services pristupaju bazi podataka i vraćaju odgvarajuće odgovore. Npr. ako je zatražen dohvat svih ljubimaca, odgvarajuća metoda će vratiti listu svih ljubimaca u sustavu.

Kontroler će odgovor od strane servisa prihvatiti. Kako servisi rade s modelima implementiranim u repozitoriju models, a pogledi s view models implementiranim u repozitroju ViewModels, kontroler mora obaviti pretvorbu modela u view model. Ovu pretvorbu obavljaju razredi iz repozitorija Mappers. Nakon što je stvren odgvarajući view model, kontroler može odgovor, u obliku view modela, proslijediti pogledu. Pogled korisniku prikazuje rezultat tražene operacije i odgovor kontrolera na njegov zahtjev.

### Korisničko sučelje i implementacija funkcionalnih zahtjeva

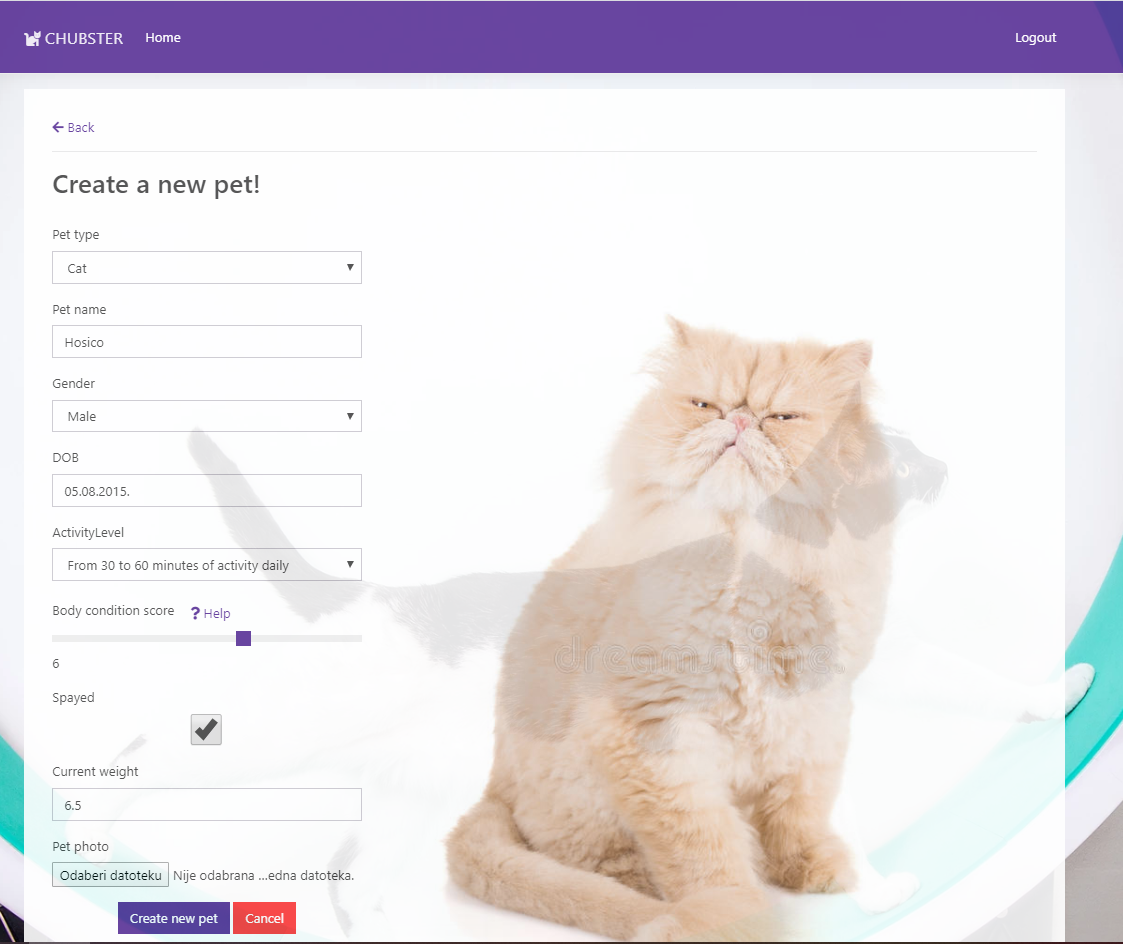
U nastavku je prikazano korisničko sučelje, te sučelja kojima su ostvareni neki od ključnih funkcionalnih zahtjeva opisanih u poglavlju Funkcionalni zahtjevi.

#### UC1: Registracija



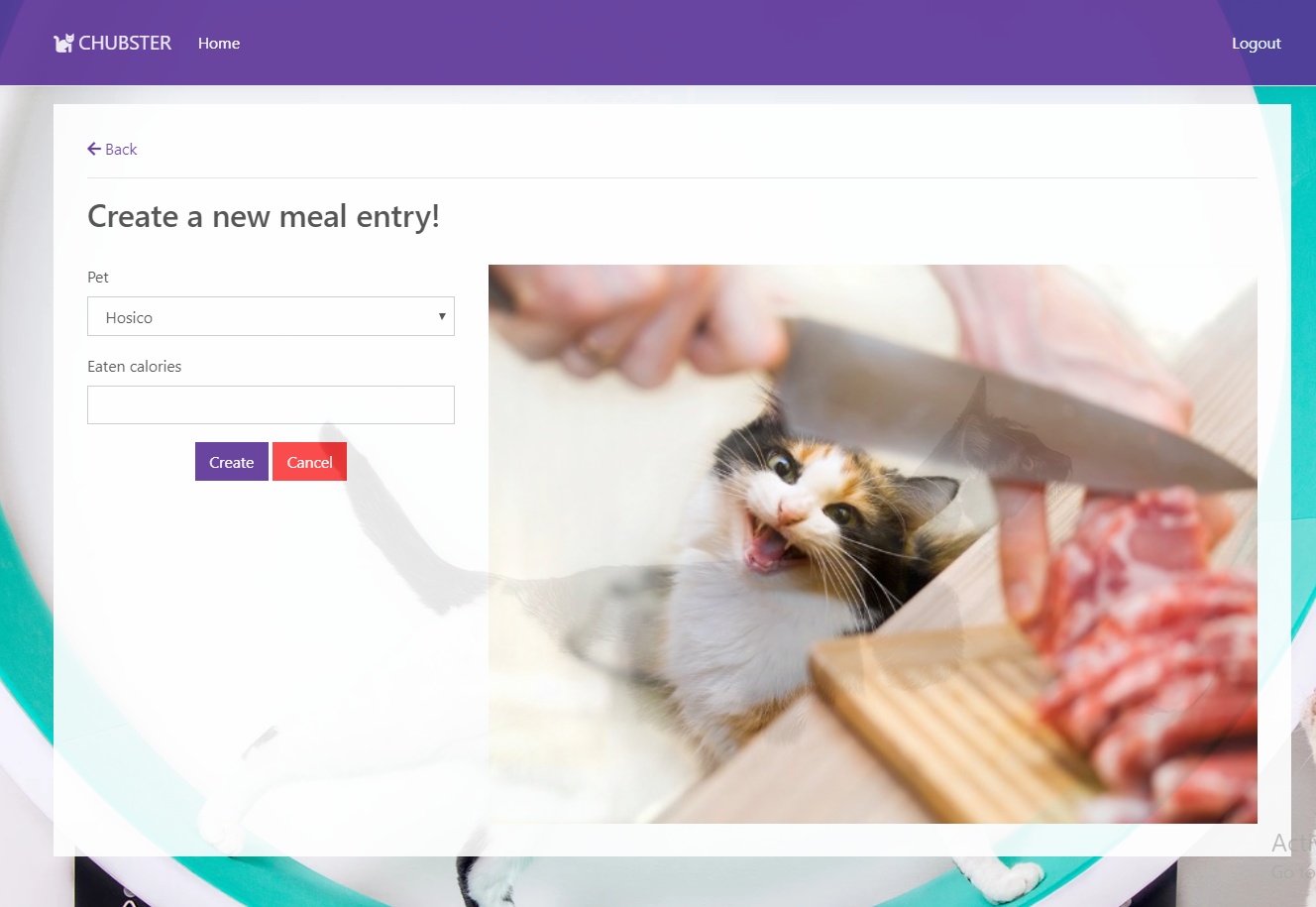
Slika 4: Korisničko sučelje za registraciju

#### UC4: Dodavanje novog kućnog ljubimca



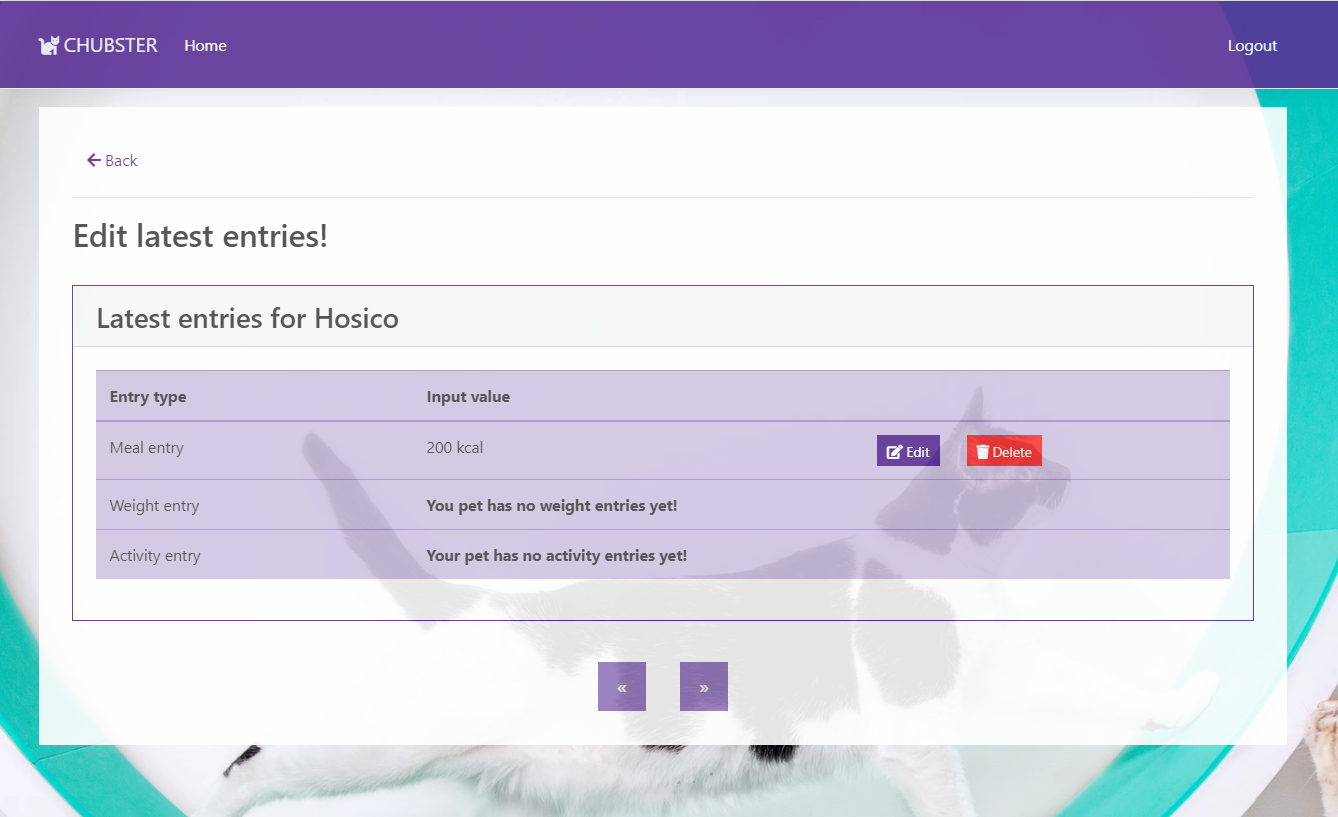
Slika 5: Korisničko sučelje za dodavanje novog kućnog ljubimca

#### UC9: Unos obroka



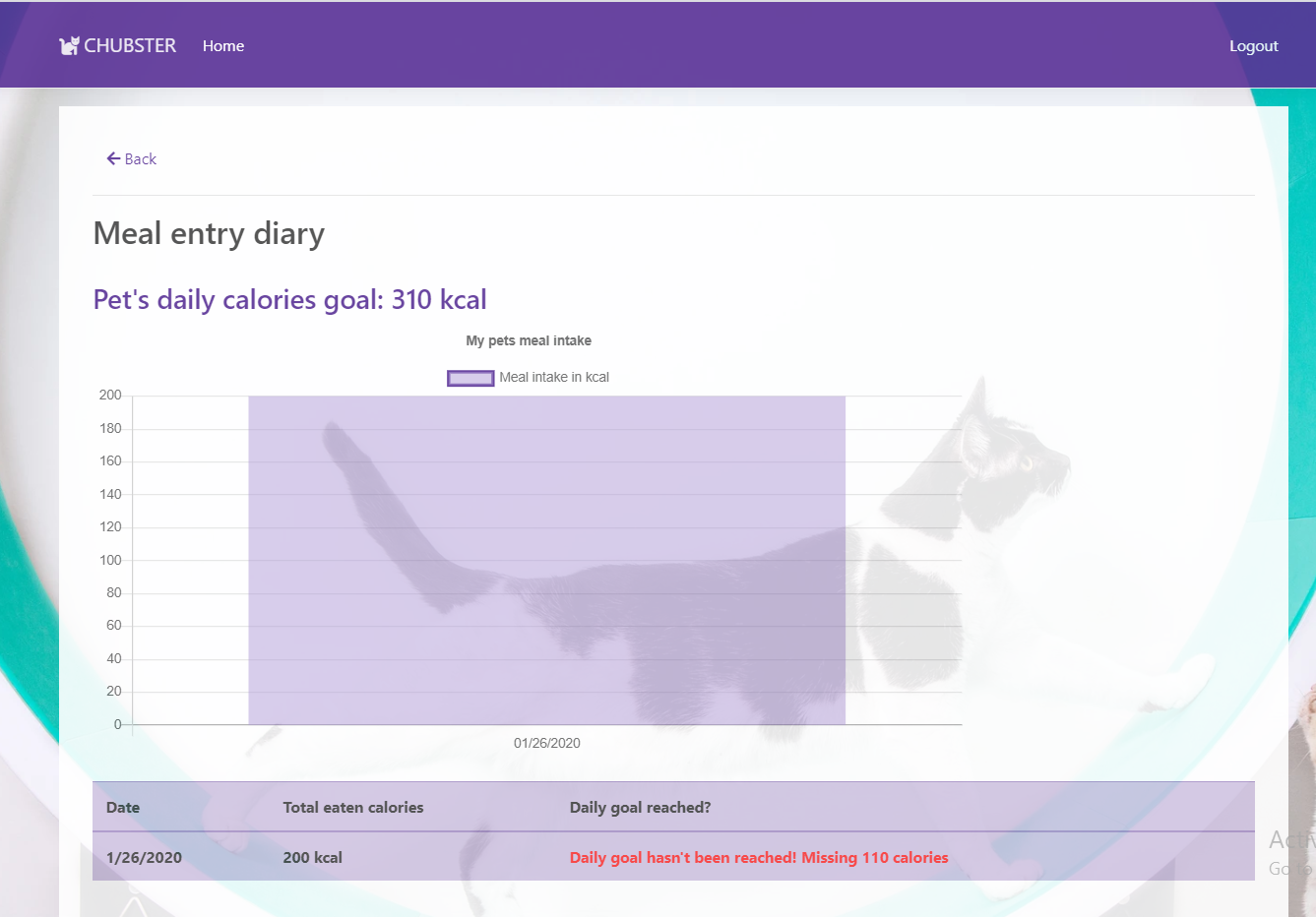
Slika 6: Korisničko sučelje za unos novog obroka

#### UC11: Izmjena unesenih zapisa o vaganju, obroku i aktivnosti



Slika 7: Korisničko sučelje za izmjenu i brisanje zapisa o vaganju, obrocima i aktivnosti

#### UC13: Tablični i grafički prikaz uneseni podataka o vaganju, obroku i aktivnosti



Slika 8: Korisničko sučelje za prikaz unesenih podataka o obrocima

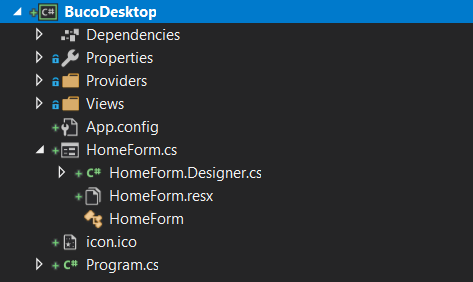
## Implementacija desktop aplikacije

Desktop aplikacija sustava Chubster služi kao pomoćna aplikacija web aplikacije. Desktop aplikacija služi kao korisničko sučelje za administratore sustava. Tako administratori imaju pristup svim podatcima u sustavu nad kojima mogu obavljati osnovne operacije čitanja, pisanja, izmijene i brisanja.

U nastavku je dan opis organizacije repozitorija, te je dan prikaz korisničkog sučelja desktop aplikacije.

### Organizacija repozitorija i ostvarenje aplikacije

Chubster desktop aplikacija je ostvarena kao .NET Core WinForms aplikacija. Aplikacija je implementirana uporabom C# programskog jezika.



Slika 9: Organizacija repozitorija desktop aplikacije sustava Chubster

Providers: Opskrbljivači imaju istu ulogu kao kontroleri u web aplikaciji, ostvaruju sloj poslovne logike aplikacije (backend).

Views: Pogledi koji imaju istu ulogu kao pogledi u web aplikaciji, ostvaruju korisničko sučelje.

Desktop aplikacija se oslanja i koristi modele i servise implementirane unutar web aplikacije. Tako desktop aplikacija sadrži reference na sve modele i razrede iz repozitorija Services.

Korisnici svoje zahtjeve šalju opskrbljivačima na obradu preko pogleda. Pogledi implementiraju korisničko sučelje i omogućuju korisnicima interakciju sa sustavom. Opskrbljivači provjeravaju ispravnost korisničkih zahtjeva. Ukoliko je zahtjev ispravan, pozivaju odgovarajuću metodu servisa. Servisi obavljaju operaciju nad bazom podataka. Servisi se oslanjaju na modele iz repozitorija Models web aplikacije. Servisi vraćaju odgvarajući odgovor opskrbljvačima, koji taj odgovor zatim prosljeđuju pogledima. Pogledi prezentiraju odgovor sustava korisniku.

### Korisničko sučelje

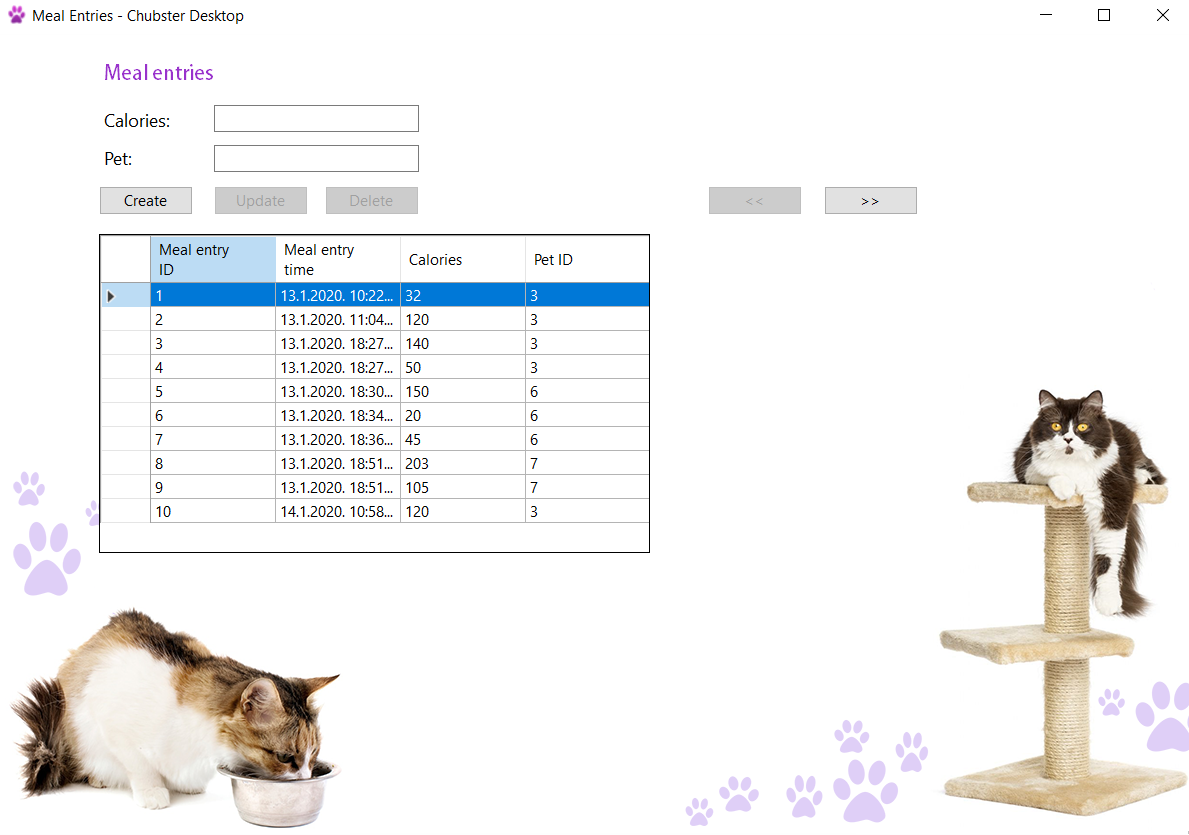
U nastavku je dan prikaz korisničkog sučelja desktop aplikacije.

#### Upravljanje ljubimcima u sustavu



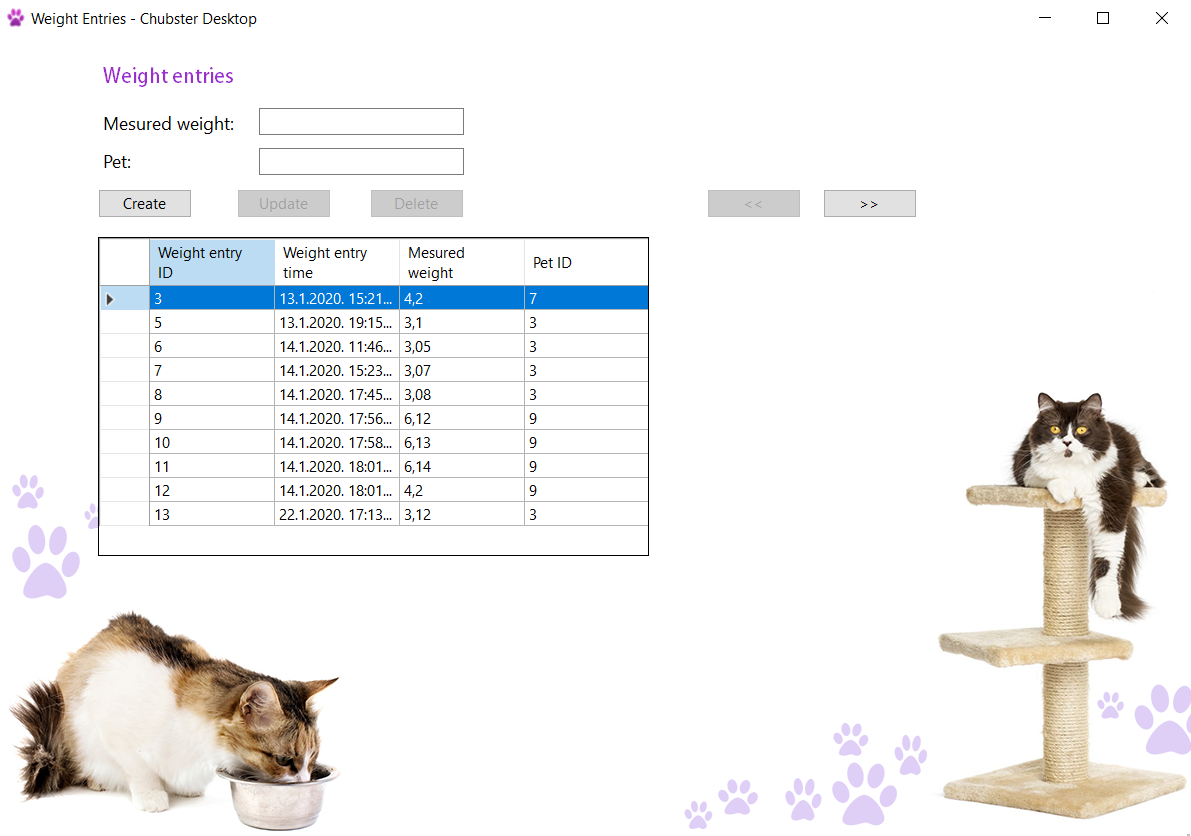
Slika 10: Korisničko sučelje za upravljanje podatcima o ljubimcima

#### Upravljane zapisima o obrocima u sustavu



Slika 11: Korisničko sučelje za upravljanje podatcima o obrocima

#### Upravljanje zapisima o težini u sustavu



Slika 12: Korisničko sučelje za upravljanje zapisima o težini

# Zaključak

Sustav Chubster je sustav koji svojim korisnicima omogućuje vođenje evidencije o težini, obrocima i dnevnoj aktivnosti kućnih ljubimaca. Sustav je implementiran kao web i desktop aplikacija. Korisnici sustava su vlasnici ljubimaca, koji se moraju registrirati kako bi mogli koristit sustav. Korisnici imaju niz zahtjeva koje sustav mora podržati kako bi korisnici imali određenu korist od sustava. Model domene sadrži razrede koji su nužni za ostvarenje funkcionalnih zahtjeva. Bez tih razreda ne bi bilo moguće ispuniti korisničke funkcionalne zahtjeve, pa samim time sustav ne bi imao nikakvu svrhu. Web i desktop aplikacije koriste istu implementaciju modela domene. Obje aplikacije također koriste i pristupaju istoj bazi podataka.

Baza podataka sustava Chubster je relacijska baza podataka. Sloj podatkovne razine obje aplikacije je ostvaren korištenjem objektno-orijentirane paradigme. Zbog toga su korištene tehnike objektno-relacijskog mapiranja koje omogućuju povezivanje ova dva sloja. Uporaom objektno-relacijske tehnike se rad s bazom sustava svodi na rad s objetima koji predstavljaju odgovarajuće relacije, pri čemu je rad s bazom prilagođen objektno-relacijskoj paradigmi.

Web aplikacija implementira sve funckionalne zahtjeve korisnika. Korisnici korištenjem web aplikacije mogu voditi evidenciju o težini, obrocima i aktivnosti svojih ljubimaca. Web aplikacija je ostvarena kao MVC .NET Core aplikacija.

Desktop aplikacija služi kao korisničko sučelje za rad s bazom podataka. Namijenjena je administratorima sustava. Ostvarena je kao WinForms .NET Core desktop aplikacija.

Svi navedeni korisnički zahtjevi su implementirani, no to ne znači da nije moguće dodatno proširiti sustav. Tako je u budućnosti moguće i implementirati mobilnu aplikaciju, koja bi usluge sustava korisnicima nudila i preko pametnih mobitela.

Uz to, u budućnosti bi se mogla popraviti implementacija modela domene uporabom NHibernate radnog okvira. Model domene ne bi trebao sadržavati nikakve podatke o mapiranju i ostvarenju baze podataka, već bi se samo trebao baviti modeliranjem poslovne domene. *DataSet*-ovi koji su atributi razreda *ApplicationDbContext* narušavaju pravilo da model domene **samo modelira poslovnu domenu**. Zbog ovih atributa dolazi do „curenja“ brige o podatkovnom sloju u model domene, što nije povoljno svojstvo. *NHibernate* omogućava transparento čuvanje podataka. Uporabom ovog radnog okvira model domene nema nikakvih podataka o klasama za pohranu podatka. Model domene je u potpunosti odvojen od ostvarenja perzistencije podataka.

Ovo su samo neke od mogućnosti proširenja i budućeg rada na sustavu.

Literatura