Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Bojan Hameršak

Podporna aplikacija za ornitologe

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu razvijte spletno aplikacijo, ki bo služila kot podpora ornitologom pri opazovanju, spremljanju in katalogiziranju ptic. Aplikacijo zasnujte tako, da bo podpirala administratorja, navadne uporabnike in goste. Administratorju omogočite urejanje uporabnikov, navadnim uporabnikom pa poleg registracije omogočite funkcionalnosti, ki so pomembne za ornitologe, kot so možnost dodajanja in urejanja relevantnih podatkov opažene ptice (npr. vrsta ptice, lokacija in datum opazovanja ptice), ter možnost dodajanja fotografije ptice. Za goste oziroma neregistrirane uporabnike pa omogočite prikaz lokacije ptičev in fotografij na zemljevidu, filtriranje glede na uporabnika, ki je dodal posamezen vnos, lokacijo in druge filtre glede na podatke o pticah, ter prikaz na zemljevidu glede na uporabljene filtre. V okviru diplome naredite načrt razvoja aplikacije in načrt podatkovne baze. Nato pa implementirajte zahtevane funkcionalnosti na strani strežnika in odjemalca. Pri tem sami izberite tehnologije na strani strežnika in na strani odjemalca. Poskrbite tudi za dobro uporabniško izkušnjo pri uporabi aplikacije na napravah z različnimi dimenzijami zaslonov, tako da naredite spletno aplikacijo odzivno.

Zahvaljujem se staršem, ki so me podpirali med študijem. Iskrena hvala mentorju doc. dr. Alešu Smrdelu za njegovo odzivnost in pomoč pri pisanju diplomske naloge. Hvala tudi vsem profesorjem na fakulteti, od katerih sem se veliko naučil.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronous JavaScript and XML	asinhroni JavaScript in XML
API	Application Programming Interface	aplikacijski programski vmesnik
ASP	Active Server Pages	aktivne strežniške strani
CIL	Common Intermediate Language	skupni vmesni jezik
CLI	Common Language Infrastructure	skupna jezikovna infrastruktura
CLR	Common Language Runtime	skupno izvajalno okolje
CRUD	Create, Read, Update, Delete	ustvari, beri, posodobi, izbriši
CSS	Cascading Style Sheets	kaskadne stilske podloge
DOM	Document Object Model	objektni model dokumenta
ECMA	European Computer Manufacturers Association	Evropsko združenje izdelovalcev računalnikov
EF	Entity Framework	ogrodje ORM
GPS	Global Positioning System	sistem globalnega pozicioniranja
HTML	Hypertext Markup Language	označevalni jezik za izdelavo spletnih strani
HTTPS	Hypertext Transfer Protocol Secure	varni komunikacijski protokl za splet

IDE	Integrated Development Environment	integrirano razvojno okolje
IMAP	Internet Message Access Protocol	internetni protokol za dostop do sporočil
IoT	Internet of Things	internet stvari
JIT	Just-In-Time	sprotni
JSON	JavaScript Object Notation	JavaScript objektna notacija
LINQ	Language Integrated Query	poizvedbe vgrajene v jezik
MVC	Model-View-Controller	model-pogled-krmilnik
NOAGS		Novi ornitološki atlas gnezdilk Slovenije
ORM	Object-Relational Mapping	objektno-relacijsko mapiranje
POP	Post Office Protocol	poštni protokol
RDBMS	Relational Database Management System	sistem za upravljanje z relacijskimi podatkovnimi bazami
SDK	Software Development Kit	zbirka orodij za razvoj programske opreme
SMTP	Simple Mail Transfer Protocol	enostavni protokol za prenos pošte
SQL	Structured Query Language	strukturiran poizvedovalni jezik
TLS	Transport Layer Security	protokol za varno povezovanje v omrežju
URL	Uniform Resource Locator	enolični naslov vira
WIC	Windows Imaging Component	Windows komponenta za prikaz slik
XML	eXtensible Markup Language	razširljiv označevalni jezik

Povzetek

Naslov: Podporna aplikacija za ornitologe

Avtor: Bojan Hameršak

Cilj diplomske naloge je bil razviti podporno spletno aplikacijo za ornitologe, ki je namenjena predvsem amaterskim opazovalcem ptic. Njeni najpomembnejši funkcionalnosti sta nalaganje in deljenje fotografij ptic z ostalimi uporabniki. Pri objavi fotografije lahko določimo tudi njeno lokacijo nastanka. Uporabniki lahko tako hitreje identificirajo lokacijo, kjer bi morda lahko opazili določeno vrsto ptice. Pregledovanje zbirke ptic je mogoče na zemljevidu ali pa po seznamu. Na voljo je več filtrov, s katerimi si lahko skrčimo iskanje. Aplikacija omogoča tudi pregledovanje redov in vrst ptic, s čimer si lahko uporabniki pomagajo pri določitvi vrste ptice. Aplikacija je izdelana tako, da se prikaz uporabniškega vmesnika prilagaja velikosti zaslona naprave. Strežniški del aplikacije je bil izdelan v jeziku C# z uporabo ogrodja ASP.NET Core MVC. Za odjemalski del so bile uporabljene standardne tehnologije HTML, CSS, JavaScript ter storitev Google Maps za prikaz lokacij nastanka fotografij.

Ključne besede: spletna aplikacija, ornitologija, slike, odzivni dizajn, model-pogled-krmilnik, Google Maps.

Abstract

Title: Support application for ornithologists

Author: Bojan Hameršak

The goal of this BSc thesis was to develop support web application for ornithologists intended primarily for amateur bird observers. Its most important functionalities are uploading and sharing photographs with other users. When uploading a photograph we can also set the place of origin, which enables users to more quickly identify the location where specific species of birds can be spotted. The application enables user to overview photographs using either a map or a list. Different filters are available, which can help in narrowing the search. The application also facilitates inspecting the orders and species of the birds, which helps users when determining the species. Application's user interface adapts to the size of the device screen. The server side of the application was developed in C# using the ASP.NET Core MVC framework. For the client side HTML, CSS and JavaScript were used along with the Google Maps service for displaying the location of the photographs' origin.

Keywords: web application, ornithology, pictures, responsive design, model-view-controller, Google Maps.

Poglavje 1

Uvod

Že nekaj časa se ukvarjam z opazovanjem ptic. Ker pa ornitologi ptic ne samo opazujemo, ampak jih tudi slikamo, se mi je nabral zajeten kup fotografij raznih ptic v elektronski obliki, ki pa je bil tudi precej neurejen. Tako se je pojavila potreba po možnosti katalogizacije fotografij oziroma slik ptic. Poleg tega se mi je včasih zgodilo, da sem hotel sliko kakšne redke ptice deliti s prijatelji. Tako sem prišel na idejo, da bi združil hobi s študijem.

1.1 Motivacija

V času študija sem spoznal kar nekaj spletnih tehnologij. Za diplomo sem želel narediti izdelek, kjer bi v celoti uporabil pridobljena znanja in ki bi ga v prihodnosti lahko še nadgradil. Odločil sem se za izdelavo spletne aplikacije, ki bi nadgrajevala osnovno idejo katalogizacije ptic z možnostjo deljenja slik s prijatelji. Želel sem izdelati uporabno spletno stran na to atraktivno temo. Ko sem premišljeval, kako bi združil prej omenjeni funkcionalnosti, sem pregledoval priljubljene spletne in tudi mobilne aplikacije. Spomnil sem se velikega zanimanja ljudi za mobilno aplikacijo Pokémon Go. Ljudje so hodili na določene lokacije samo zato, da bi videli neko navidezno bitje. Tako sem dobil idejo, da bi moja spletna aplikacija omogočala objavljanje slik ptic in njihovo lokacijo na zemljevidu.

Od divjih živali so ptice ena najlažjih skupin za opazovanje. Ker nam lahko hitro pobegnejo, se nam tudi precej približajo. Za opazovanje so mi zanimive, ker se med seboj zelo razlikujejo po barvah, oglašanju in obnašanju. Pomembno se mi zdi tudi osveščanje ljudi o naravi in našem vplivu nanjo, saj število živali v zadnjih letih pospešeno upada zaradi krčenja njihovega življenjskega prostora. Zato nekaterih vrst ptic, ki so jih še lahko opazovali moji starši, pri nas ni več mogoče videti. Kaj bodo lahko videli zanamci je odvisno tudi od nas, čeprav imamo omejen vpliv.

1.2 Cilji

Glavni cilj pri izdelavi diplomske naloge je bil narediti spletno aplikacijo, ki bi omogočala shranjevanje slik in njihovo katalogizacijo. Ker pa je zanimivo deliti slike tudi z drugimi, je bil sekundarni cilj tudi možnost objavljanja slik na internetu in deljenja z drugimi uporabniki. Zelo pomembna je tudi uporabniška izkušnja aplikacije, zato sta bili zahtevi tudi enostavnost uporabe in lep izgled aplikacije na vseh napravah, kar smo zagotovili z odzivnim dizajnom.

1.3 Pregled področja

Na spletu sem iskal, če že obstaja kakšna podobna aplikacija. V slovenščini sem našel NOAGS (Novi ornitološki atlas gnezdilk Slovenije) [11], ki je namenjen popisovanju pojavljanja ptic in prikazu podatkov na zemljevidu. Žal ta aplikacija ne omogoča tudi prikaza slik, kar bi lahko bilo v pomoč tudi ljubiteljskim opazovalcem ptic. Moja aplikacija prikazuje simbolične slike vrst ptic, omogoča pa tudi objavo slik uporabnikov, kar se mi zdi z vidika amaterskih oziroma ljubiteljskih opazovalcev bolj privlačno.

Med tujimi aplikacijami se mi zdi uporabna aplikacija eBird [3] inštituta Cornell Lab of Ornithology. Omogoča podobne funkcionalnosti kot moja aplikacija, to je deljenje slik in lokacije, možno pa je naložiti tudi avdio in

video posnetke. Dobra lastnost aplikacije je tudi ta, da že ima veliko zbirko slik. Njene slabosti so malo manj pregleden uporabniški vmesnik strani, na zemljevidu ni možno pregledovati slik vseh vrst ptic naenkrat poleg tega pa tudi ni na voljo predogleda slike, manjkajo pa ji tudi slovenske regije za boljše določanje lokacije.

Poleg namenskih aplikacij pa lahko slike delimo tudi na družbenih omrežjih, med katerimi sta na primer zelo priljubljeni aplikaciji Facebook in Instagram. Objave v teh družbenih omrežjih ponavadi niso v posebnih skupinah, zaradi česar jih je čez čas težje najti in so manj pregledne kot v namenskih aplikacijah, poleg tega pa zaradi splošnosti ne omogočajo enostavnega iskanja in pregledovanja ter katalogiziranja slik.

1.4 Struktura diplomske naloge

V prvem poglavju so predstavljeni uvod v diplomsko nalogo, motivacija, cilji, ki smo si jih zadali ob začetku izdelave diplomske naloge, ter pregled področja.

V drugem poglavju so opisane glavne tehnologije in orodja na strani strežnika ter na strani odjemalca, ki smo jih uporabili med razvojem aplikacije.

V tretjem poglavju je predstavljeno načrtovanje aplikacije od identifikacije funkcionalnosti preko definiranja strukture podatkov oziroma podatkovnega modela, do izgleda uporabniškega vmesnika aplikacije.

V četrtem poglavju je podrobno opisan razvoj aplikacije in njeno delovanje.

V zadnjem poglavju pa so predstavljeni zaključki, do katerih smo prišli med izdelavo diplomske naloge, predstavljene pa so tudi ideje za nadaljnje delo.

Poglavje 2

Uporabljene tehnologije

V tem poglavju predstavljamo tehnologije in orodja, ki smo jih uporabili pri izdelavi aplikacije v okviru diplomske naloge. Razdelili smo jih glede na stran uporabe.

2.1 Tehnologije na strani strežnika

V tem razdelku so predstavljene tehnologije, ki se uporabljajo na strani strežnika.

2.1.1 Microsoft Visual Studio

Microsoft Visual Studio [9] je integrirano razvojno okolje oziroma IDE (ang. *Integrated Development Environment*). Čeprav striktno gledano to ni strežniška tehnologija temveč razvojno okolje, smo ga med te tehnologije uvrstili, ker prvenstveno omogoča enostaven razvoj aplikacij na strani strežnika (delno pa tudi odjemalca). Microsoft je prvo različico izdal leta 1997, danes pa je aktualna različica Visual Studio 2017. Primarno je namenjen razvoju programov za operacijski sistem Microsoft Windows, spletnih strani in aplikacij, spletnih storitev ter mobilnih aplikacij. S svojimi orodji nam omogoča lažji in hitrejši razvoj v jezikih C, C++, C#, Visual Basic, F# in TypeScript. Z namestitvijo razširitev pa podpira tudi številne druge jezike. IDE vse-

buje urejevalnik izvorne kode, avtomatizirano prevajanje kode (ang. *build*), razhroščevalnik (ang. *debugger*) ter še druga koristna orodja.
Pri pisanju izvorne kode nam zelo pomaga orodje IntelliSense, s katerim dobimo večji vpogled v kodo, ki jo pišemo oziroma uporabljamo. Ko napišemo znak, ki loči elemente kode (v jeziku C# je to pika), nam prikaže seznam vseh naslednjih možnih elementov. Prikaže nam tudi informacije o elementih, na primer opis metode in njenih parametrov. Tako lahko namesto celotne kode napišemo samo nekaj začetnih črk, s pomočjo IntelliSensa pa dokončamo ukaze.

2.1.2 C#

C# [2] je visokonivojski objektno orientiran programski jezik. Razvil ga je Microsoft v okviru ogrodja .NET. Prva različica je bila izdana leta 2002. Sintaksa je zelo podobna jeziku Java. Pri načrtovanju so uporabili dobre lastnosti jezikov C++ in Java ter odpravili nekatere pomanjkljivosti. Glavni cilji jezika C# kot jih navaja standard ECMA-334 [14]:

- preprost, sodoben, splošnonamenski, objektno orientiran jezik;
- preverjanje močne tipizacije, preverjanje meje polja, zaznavanje poskusa uporabe neinicializiranih spremenljivk, avtomatsko čiščenje pomnilnika;
- omogoča komponentno programiranje za porazdeljene sisteme;
- prenosljivost je zelo pomembna za izvorno kodo in programerje, ki že poznajo C in C++;
- podpora za internacionalizacijo;
- primeren za uporabo v vgradnih sistemih od zelo velikih, ki uporabljajo dovršene operacijske sisteme, do majhnih z le nekaj določenimi funkcijami;

- C# aplikacije učinkovito uporabljajo vire, kot sta pomnilnik in procesna moč, ni pa namen tekmovati z učinkovitostjo jezika C ali zbirnika.

Med nekaterimi pomembnimi lastnosti, ki jih je jezik pridobil z novimi različicami, je potrebno omeniti še generično programiranje, poizvedbe LINQ (ang. *Language Integrated Query*) in asinhrone metode.

C# uporablja veliko zbirko knjižnic ogrodja .NET. Programi se ne prevedejo v strojno kodo, ampak se izvajajo v posebnem navideznem izvajalskem okolju standarda CLI (ang. *Common Language Infrastructure*) kot upravljana koda CIL (ang. *Common Intermediate Language*). Dele take programske kode je lahko povezovati z deli, ki so napisani v katerem drugem jeziku skladnem s CLI [13]. Microsoftova implementacija standarda CLI se imenuje CLR (ang. *Common Language Runtime*). Izvajanje upravljane kode omogoča lahko prenosljivost kode za izvajanje na drugih okoljih. Eni takih implementacij CLR sta tudi .NET Core in pa Mono [15].

2.1.3 .NET Core

.NET Core [10] je odprtokodno splošnonamensko računalniško okolje, ki ga razvijata Microsoft in .NET skupnost. Predstavljamo si ga lahko kot okolje .NET Framework, samo da je namenjen različnim platformam. Deluje na operacijskih sistemih Windows, macOS in Linux. Primeren je za uporabo v aplikacijah v oblaku, na namiznih in prenosnih računalnikih, mobilnih napravah, vgradnih sistemih in napravah IoT (ang. *Internet of Things*). Okolje je lahko nameščeno v aplikaciji, na napravi oziroma sistemu ali pa samo pri posameznih uporabnikih. Sestavljajo ga:

- izvajalno okolje CoreCLR, osnovne knjižnice, čistilec pomnilnika (ang. *garbage collector*), sprotni prevajalnik (ang. *JIT compiler*);
- zbirka knjižnic CoreFX;
- zbirka orodij za razvoj programske opreme .NET Core SDK (ang. *Software Development Kit*);

- gostitelj za zagon aplikacij .NET Core in orodij SDK.

2.1.4 ASP.NET Core MVC

ASP.NET (ang. *Active Server Pages .NET*) je ogrodje za razvoj spletnih aplikacij, ki ga je razvil Microsoft. Nova generacija tega ogrodja je ASP.NET Core, ki združuje prej ločena ASP.NET MVC (ang. *Model-View-Controller*) in Web API (ang. *Web Application Programing Interface*, aplikacijski programski vmesnik za spletne storitve) v en programski model. Spletna aplikacija te diplomske naloge je izdelana z ASP.NET Core 1.1.

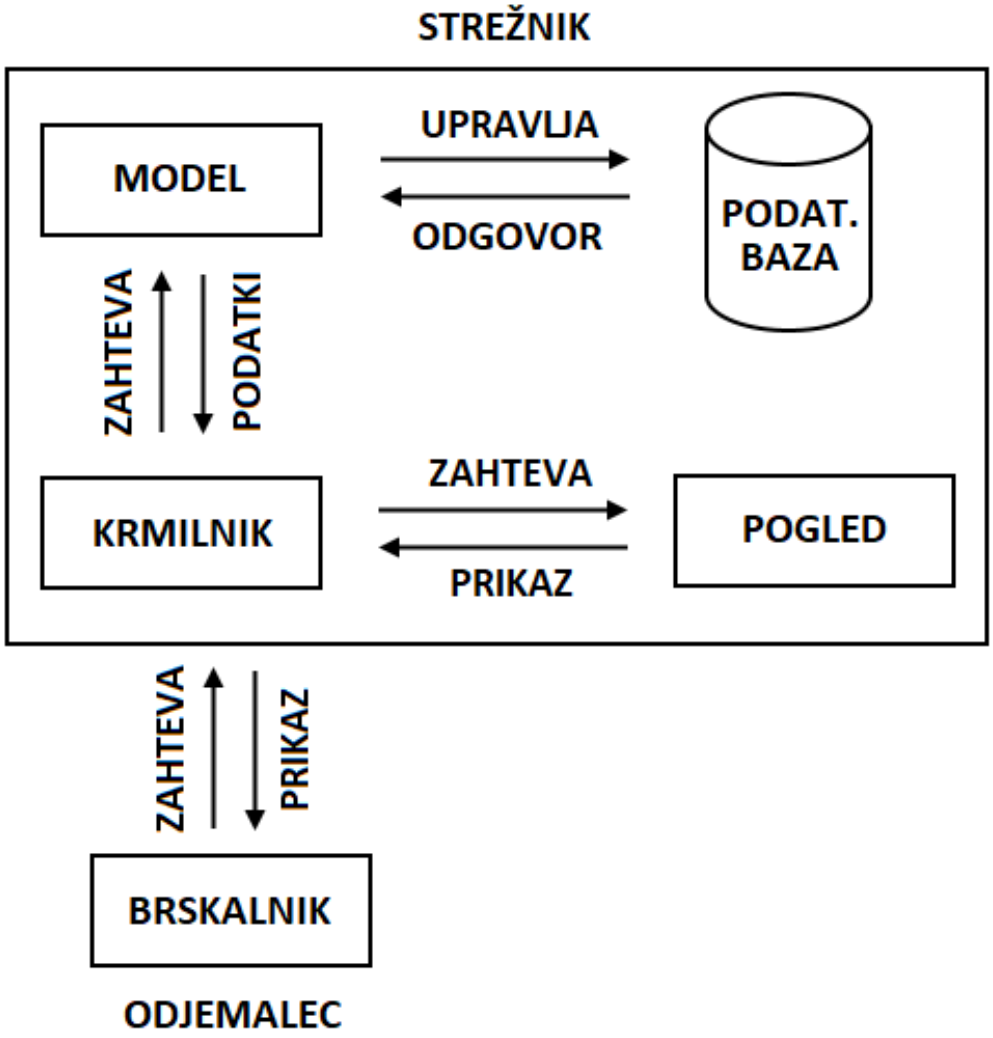
ASP.NET Core MVC je zgrajen na ASP.NET Core in omogoča izgradnjo spletnih aplikacij na podlagi arhitekture model-pogled-krmilnik, ki deli implementacijo na tri dele (slika 2.1):

- **model**: določa poslovno logiko in pravila, neposredno upravlja podatke;
- **pogled** (ang. *view*): zadolžen za uporabniški vmesnik, podatke mu poda krmilnik, ne komunicira neposredno z modelom;
- **krmilnik** (ang. *controller*): obravnava zahtevke odjemalca, prek modela upravlja s podatki, sporoči pogledu kaj naj prikaže, vrne prikaz odjemalcu.

Uporabnik izvaja interakcijo z uporabniškim vmesnikom, ki mu ga posreduje krmilnik, zgradi pa ga pogled. Krmilnik lahko na podlagi te interakcije pošlje zahtevo za podatke v model. Model pošlje zahtevo podatkovni bazi, ki vrne podatke. Model te podatke posreduje krmilniku, ta pa jih posreduje pogledu, ki zgradi prikaz za uporabnika.

Ta delitev omogoča učinkovito ponovno uporabo kode in vzporedni razvoj posameznih delov aplikacije.

Kateri krmilnik je izbran za obravnavo zahteve uporabnika, se določi glede na naslov URL (ang. *Uniform Resource Locator*), v katerem je lahko določena tudi metoda krmilnika oziroma akcija (ang. *action method*).



Slika 2.1: Diagram interakcije v modelu MVC.

2.1.5 Entity Framework Core

Entity Framework (EF) [4] je odprtokodno ogrodje za ORM (ang. *Object-Relational Mapping*). ORM povezuje objektno orientirano kodo z relacijsko podatkovno bazo oziroma razrede s tabelami. Tako v programskem jeziku uporabljamo nekakšno navidezno objektno podatkovno bazo, do katere dostopamo prek EF. Za dostop do podatkov lahko namesto jezika SQL (ang.

Structured Query Language) uporabljamo LINQ.

Za implementacijo aplikacije v okviru te diplomske naloge smo uporabili EF Core 1.1. Za izdelavo podatkovne baze smo uporabili pristop „najprej koda“ (ang. *code first approach*). Tako smo najprej naredili razrede v jeziku C#, iz katerih nam je potem EF ustvaril tabele v relacijski podatkovni bazi.

Za podatkovno bazo smo uporabili Microsoft SQL Server. Prednost uporabe EF je še ta, da aplikacija ne komunicira neposredno s samo podatkovno bazo, saj to stori EF namesto nje. Zaradi tega ne bi smeli imeti večjih težav z zamenjavo ponudnika podatkovne baze.

2.1.6 Microsoft SQL Server

Microsoft SQL Server je Microsoftov sistem za upravljanje z relacijskimi podatkovnimi bazami oziroma RDBMS (ang. *Relational Database Management System*). Prva različica je bila izdana leta 1989, danes pa je aktualna različica SQL Server 2017. Microsoft ponuja več različnih izdaj glede na zahteve uporabnikov. Ker je naša aplikacija še v fazi testiranja, smo uporabili kar minimalno različico SQL Server LocalDB, ki se uporablja za aplikacije v razvoju.

2.1.7 MagicScaler

MagicScaler [6] je knjižnica za spreminjanje ločljivosti slik. V primerjavi z ostalimi rešitvami je izjemno hitra in ohranja boljšo kvaliteto slik [8]. Za kodiranje in dekodiranje slik uporablja WIC (ang. *Windows Imaging Component*). Zato podpira vse kodeke, ki jih podpira operacijski sistem Windows, lahko pa se tudi dodatno naložijo. MagicScaler uporablja napredne optimizacijske tehnike. V primerjavi z ostalimi rešitvami, ki večinoma uporabljajo API GDI+, je MagicScaler hitrejši, porabi manj pomnilnika in ohranja lepše slike.

2.1.8 MailKit

MailKit je knjižnica za odjemalca e-pošte, ki razširja MimeKit. Podpira protokole SMTP (ang. *Simple Mail Transfer Protocol*), POP3 (ang. *Post Office Protocol*) in IMAP (ang. *Internet Message Access Protocol*). Uporabili smo jo za pošiljanje e-pošte, ki jo v okviru naše aplikacije pošiljamo ob registraciji v aplikacijo in v primeru, če je registriran uporabnik pozabil geslo. Ker ne pričakujemo velikega števila poslane e-pošte, za vsakega uporabnika po eno sporočilo ob registraciji in občasno kakšno sporočilo ob pozabljenem geslu, nam ta rešitev zadošča. V primeru, da se bo pokazala potreba po pošiljanju večjih količin sporočil, pa bomo razmislili o zamenjavi te knjižnice z uporabo naprednejših in zanesljivejših storitev za masovno obveščanje z e-pošto.

2.2 Tehnologije na strani odjemalca

V tem razdelku pa so predstavljene tehnologije, ki se uporabljajo pri spletni aplikaciji na strani odjemalca.

2.2.1 HTML

HTML (ang. *Hypertext Markup Language*) je označevalni jezik za izdelavo spletnih strani, ki jih predstavlja dokument HTML. Prvi predlog standarda je bil izdan leta 1993, danes pa je aktualna različica HTML 5.1, ki je bila izdana leta 2016. Elemente spletne strani gnezdimo med oznake, ki predstavljajo zgradbo in semantični pomen delov dokumenta. Dokument HTML lahko vsebuje tudi druge jezike, ki vplivajo na prikaz in obnašanje spletne strani. Poleg HTML se največkrat uporabljata skriptni jezik JavaScript, ki omogoča dinamično spreminjanje dokumenta, in CSS (ang. *Cascading Style Sheets*), ki določa izgled in postavitev vsebine spletne strani.

2.2.2 CSS

CSS oziroma kaskadne stilske podloge so preprost stilski jezik, s katerim opišemo, kako naj se elementi v dokumentu HTML prikažejo na spletni strani. Z uporabo CSS se je omogočilo ločevanje strukture in vsebine elementov HTML od njihovega izgleda in postavitve. Omogoča nam tudi, da lastnosti prikaza določimo na enem mestu.

Danes je aktualna različica 3, ki jo v veliki meri podpirajo najpomembnejši brskalniki, in ki poleg ostalega omogoča tudi animacije in medijske poizvedbe. Veliko stvari, ki so bile prej mogoče samo z JavaScriptom, se sedaj lahko implementira preko stilskih podlog. Medijske poizvedbe omogočajo implementacijo spletnih aplikacij, ki se lahko dobro prikažejo na različnih napravah z različnimi velikostmi zaslonov.

2.2.3 JavaScript, jQuery

JavaScript [7] je skriptni programski jezik, ki nam pomaga ustvariti interaktivne in dinamične spletne strani. Prvenstveno se uporablja na strani odjemalca za dinamično manipulacijo komponent DOM (ang. *Document Object Model*) spletnih strani, v zadnjih časih pa postaja tudi vse bolj priljubljen na strežniški strani v strežniških izvajalnih okoljih, kot je na primer Node.js. Mi smo ga uporabili samo na strani odjemalca in sicer za preverjanje pravilnosti vnosa, interaktivni izbor datuma, prikaz slike čez celotno širino okna odjemalca in posodabljanje vsebine elementov HTML s pomočjo klicev AJAX (ang. *Asynchronous JavaScript and XML*). Uporabljali smo tudi knjižnico jQuery, ki poenostavlja sintakso. Klic jQuery vrne objekt jQuery, nad katerim lahko ponovno izvedemo kakšen klic in s tem močno poenostavimo pisanje aplikacij na odjemalskem delu.

2.2.4 Bootstrap

Bootstrap je odprtokodno ogrodje za oblikovanje spletnih strani in aplikacij. Razvil ga je Twitter z namenom zagotavljanja enotnega izgleda in lažjega

vzdrževanja svojih orodij. Vsebuje predloge podlog CSS in razširitve JavaScript za izgled elementov HTML. Bootstrap se zelo pogosto uporablja za zagotavljanje odzivnega dizajna.

2.2.5 AJAX

AJAX [1] je zbirka več spletnih tehnologij na strani odjemalca, ki nam pomaga narediti bogate spletne aplikacije, ki se po odzivnosti približujejo namiznim aplikacijam. Spletne aplikacije lahko pošiljajo in sprejemajo podatke od strežnika asinhrono oziroma v ozadju. Pri tem pa uporaba te tehnologije omogoča, da se prenaša samo del podatkov, s katerim lahko potem posodobimo le del in ne cele spletne strani, kar pohitri sam prikaz, obenem pa to pomeni tudi manj prometa po omrežju. Ker ta tehnologija omogoča asinhrone klice, ni potrebno čakati na rezultat zahtevka, aplikacija se lahko izvaja nemoteno naprej, v ozadju pa se izvaja vrnitvena funkcija, ki čaka na odgovor in ko ta prispe, posodobi samo del dokumenta, zato je izris hitrejši. Podatki se pošiljajo v obliki dokumenta XML (ang. *eXtensible Markup Language*) ali JSON (ang. *JavaScript Object Notation*). Da AJAX deluje, mora imeti odjemalec omogočen JavaScript.

2.2.6 Google API, Google Maps

Google API je spletni vmesnik, ki nam omogoča uporabo Googlovih storitev v naših aplikacijah. Mi smo ga uporabili za prikaz lokacij nastanka slik na zemljevidu Google Maps [5]. Za uporabo APIja moramo najprej pridobiti zastonj ključ za API, s čimer Google beleži uporabo svojih storitev. Zemljevid enostavno vgradimo v aplikacijo z naslovom Google Maps Embed API URL. Za dodajanje oznak na zemljevid uporabljamo JavaScript. V kodi spodaj je predstavljen primer vstavljanja zemljevida Google Maps v pogled **Objave/Create.cshtml**:

```
<script src="https://maps.googleapis.com/maps/api/js?key=
  API_KEY&callback=myMap&language=sl&region=SI"></script>
```

```
<script type="text/javascript">
  var marker;

  function myMap() {
    var geoss = new google.maps.LatLng(46.119944, 14.815333);
    var mapCanvas = document.getElementById("zemljevid");
    var mapOptions = { center: geoss, zoom: 8 };
    var map = new google.maps.Map(mapCanvas, mapOptions);

    google.maps.event.addListener(map, 'click',
      function (event) {
        $("#ZemljepisnaSirina").val(event.latLng.lat());
        $("#ZemljepisnaDolzina").val(event.latLng.lng());
        placeMarker(map, event.latLng);
    });
  }

  function placeMarker(map, location) {
    if (marker) {
      marker.setMap(null);
    }
    marker = new google.maps.Marker({
      position: location,
      map: map
    });
  }
 </script>
```

Poglavje 3

Načrtovanje aplikacije

V tem poglavju predstavljamo korake načrtovanja aplikacije, ki so potrebni za učinkovit razvoj in tudi za čim bolj učinkovito in uporabniku prilagojeno aplikacijo.

3.1 Zajem zahtev

Pri vsaki aplikaciji je zelo pomembno, da se naredi zajem zahtev, s katerim se evidentirajo potrebe uporabnika. S tem lahko v največji meri zagotovimo, da bodo uporabniki aplikacijo radi uporabljali ter jim bo v pomoč pri nalogah, za katere je namenjena. Ker sem tudi sam amaterski ornitolog, je bil ta korak sorazmerno enostaven, saj se moje potrebe dokaj dobro ujemajo s potrebami ostalih amaterskih ornitologov. Tako sem evidentiral funcionalnosti, ki jih mora ta aplikacija imeti z vidika ornitologa. Poleg tega pa sem evidentiral še funkcionalnosti, ki jih mora aplikacija nuditi za administracijo. V naslednjih razdelkih so predstavljene zahteve, grupirane glede na vloge uporabnikov in še dodatne splošne zahteve.

3.1.1 Funkcionalne zahteve za uporabnika

- nalaganje slik ptic in določanje lokacije nastanka;

- upravljanje z objavami slik;
- pregledovanje slik ptic na zemljevidu in po seznamu;
- pregledovanje redov in vrst ptic.

3.1.2 Funkcionalne zahteve za administracijo

- upravljanje z uporabniki;
- določanje moderatorjev;
- potrjevanje objave slik;
- upravljanje z redovi in vrstami ptic;
- varnostno kopiranje objav slik.

3.1.3 Dodatne zahteve

- privlačen uporabniški vmesnik;
- lep prikaz spletne strani na vseh napravah;
- varnost;
- učinkovitost;
- zanesljivost.

3.2 Načrtovanje funkcionalnosti

Ko so bile evidentirane zahteve, smo se lotili načrtovanja razvoja aplikacije. V okviru načrta razvoja smo najprej podrobneje definirali funkcionalne zahteve predstavljene zgoraj. Nato smo določili tudi, katere vloge uporabnikov ima naša aplikacija in katere funkcionalnosti so posamezni vlogi dodeljene. Ločimo med štirimi različnimi vlogami uporabnikov, ki imajo dovoljene različne funkcionalnosti oziroma akcije.

Neregistriran uporabnik je uporabnik, ki ni prijavljen (lahko tudi ni registriran) v aplikacijo. Ta uporabnik ima na voljo najmanjši nabor funkcionalnosti:

- pregledovanje redov in vrst ptic: Neregistriran uporabnik lahko pregleduje redove in vrste ptic po seznamu. Pomaga si lahko z iskanjem, filtri in razvrščanjem.
- pregledovanje slik ptic: Neregistriran uporabnik lahko pregleduje samo potrjene objave slik na zemljevidu ali pa po seznamu. Pomaga si lahko z iskanjem, filtri in razvrščanjem.
- registracija: Neregistriran uporabnik vnese svoj e-poštni naslov, ime, priimek, geslo in potrditev gesla. Če je postopek uspel, mu aplikacija pošlje povezavo za potrditev registracije na njegov e-poštni naslov.

Registriran uporabnik oziroma nalagalec je uporabnik, ki je prijavljen v aplikacijo, nima pa nobene privilegirane vloge. Na voljo pa ima večji nabor funkcionalnosti kot neregistriran uporabnik in sicer:

- vse akcije neregistriranega uporabnika (tudi registracijo, ker aplikacija dovoljuje uporabniku več uporabniških imen);
- pregledovanje slik ptic: Registriran uporabnik lahko pregleduje potrjene in svoje objave slik na zemljevidu ali pa po seznamu. Pomaga si lahko z iskanjem, filtri in razvrščanjem.
- prijava: Registriran uporabnik se prijavi v aplikacijo z vnosom e-poštnega naslova in gesla.
- nalaganje oziroma objavljanje slik ptic in določanje lokacije nastanka: Registriran uporabnik izbere željeno slikovno datoteko, izbere lokacijo nastanka s klikom na zemljevid Google Maps in izpolni ostale podatke o sliki. Preden je naložena slika vidna vsem, jo mora potrditi moderator ali administrator.

- upravljanje z naloženimi slikami: Registriran uporabnik lahko spreminja podatke o svojih naloženih slikah ter jih lahko tudi izbriše.
- ponastavitev gesla: Registriran uporabnik vnese svoj e-poštni naslov, na katerega mu aplikacija pošlje povezavo za ponastavitev gesla.

Moderator je uporabnik, ki ima že nekoliko višje privilegije in ima na razpolago tudi večji nabor funcionalnosti:

- vse akcije registriranega uporabnika;
- pregledovanje slik ptic: Moderator lahko pregleduje potrjene in nepotrjene objave slik po zemljevidu ali pa po seznamu. Pomaga si lahko z iskanjem, filtri in razvrščanjem.
- potrjevanje objav: Moderator lahko potrdi objavo slike, s čimer postane vidna vsem.
- upravljanje z objavami slik: Moderator lahko spreminja podatke o vseh naloženih slikah ter jih lahko tudi izbriše.

Administrator je uporabnik z največ pravicami in s tem tudi največjim naborom funcionalnosti, saj ima na voljo:

- vse akcije moderatorja;
- upravljanje z redovi in vrstami ptic: Administrator lahko dodaja, spreminja ter briše redove in vrste ptic.
- upravljanje z uporabniki: Administrator lahko spreminja podatke o uporabnikih, jih izbriše ter jim doda ali odvzame vlogo moderatorja. Poleg tega ima tudi vpogled v majhno statistiko objav posameznega uporabnika.

- upravljanje z varnostnimi kopijami: Administrator lahko ustvarja in briše varnostne kopije objav slik. Z varnostno kopijo lahko obnovi stanje objav slik v podatkovni bazi in datoteke slik.

3.3 Načrtovanje podatkovnega modela

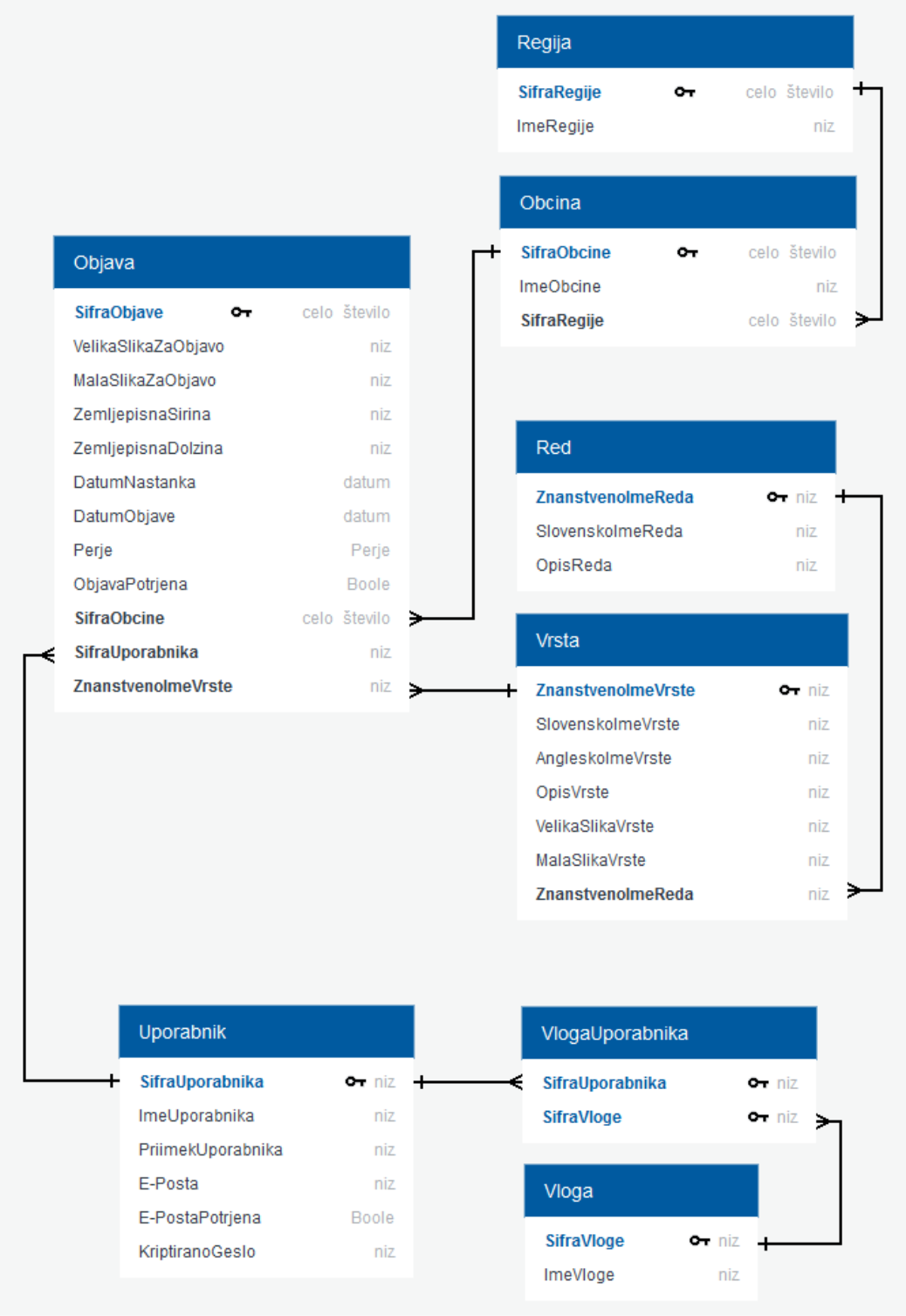
V naslednjem koraku načrtovanja smo izvedli eno najpomembnejših nalog, to je načrtovanje podatkovnega modela. Naredili smo konceptualni model v obliki entitetno-relacijske sheme. V njej smo z entitetnimi tipi predstavili podatke, ki jih želimo shranjevati in določili, kako so med seboj povezani. Na podlagi entitetnih tipov smo potem naredili razrede v programskem jeziku C#. S pomočjo EF so bile iz razredov ustvarjene tabele v relacijski podatkovni bazi.

Začeli smo z entitetno-relacijsko shemo, ki jo sestavljajo entitetni tipi **Uporabnik**, **Vloga**, **VlogaUporabnika**, **Objava**, **Vrsta**, **Red**, **Obcina** in **Regija**, ki je prikazana na sliki 3.1.

Entitetna tipa **Uporabnik** in **Vloga** sta v razmerju mnogo proti mnogo (ang. *many-to-many*). To pomeni, da je uporabnik lahko v več vlogah, posamezno vlogo pa ima lahko več uporabnikov. Zato potrebujemo še vmesni entitetni tip **VlogaUporabnika**, ki pove kateri uporabnik ima katero vlogo. **Uporabnik** in **Vloga** sta v razmerju ena proti mnogo (ang. *one-to-many*) z entitetnim tipom **VlogaUporabnika**.

Entitetni tip **Objava** oziroma tabela **Objava** v relacijski podatkovni bazi je namenjena shranjevanju novih objav slik in lokacij. **Objava** je v razmerju mnogo proti ena (ang. *many-to-one*) z entitetnimi tipi **Uporabnik**, **Vrsta** in **Obcina**. To pomeni, da je vsaka objava povezana z enim uporabnikom, eno vrsto ptice in eno občino. Preko entitetnega tipa **Uporabnik** je tako definirano, kdo je nalagalec in avtor slike, z entitetnim tipom **Vrsta** je definirano za katero vrsto ptice gre, entiteni tip **Obcina** pa definira občino, kjer je bila slika posneta.

Entitetni tip **Vrsta** je v razmerju mnogo proti ena z entitetnim tipom



Slika 3.1: Entitetno-relacijska shema, ki je bila načrtana za razvito aplikacijo.

Red. Ta relacija opisuje dejstvo, da je v vsakem redu lahko več vrst, pri tem pa je vsaka vrsta v enem redu. Pri objavi slike izberemo vrsto ptice. Z izborom vrste je posredno izbran tudi red ptice. Red ptice (npr. pevci) je širša skupina kot vrsta ptice (npr. kos) in nam lahko koristi kot filter pri prikazu vrst ptic ali prikazu objavljenih slik.

Entitetni tip **Obcina** je v razmerju mnogo proti ena z entitetnim tipom **Regija**. Ta relacija pa opisuje dejstvo, da je v vsaki regiji lahko več občin, pri tem pa je vsaka občina v eni regiji. Pri objavi slike izberemo občino lokacije. Z izborom občine je posredno izbrana tudi regija lokacije. Regija je širše področje kot občina in nam lahko koristi kot filter pri prikazu objavljenih slik.

Vrsta ptic in Red ptic sta poimenovana kot vrsta in red. Če bi želeli, bi lahko ta podatkovni model kasneje uporabili tudi za kakšno drugo aplikacijo, ki bi namesto ptic shranjevala kake druge živali ali rastline.

Za izdelavo logičnega in fizičnega modela smo sledili pristopu „najprej koda“, pri katerem se najprej ustvarijo razredi v programskem jeziku, nato pa se iz teh razredov ustvarijo tabele v relacijski podatkovni bazi. Nekateri razredi in tabele v podatkovni bazi imajo malo drugačno poimenovanje kot entitetni tipi, imajo pa tudi nekaj več lastnosti. Nekatere razrede za uporabnike nam je generiral Visual Studio. Razredi v jeziku C# imajo tudi povezovalne lastnosti (ang. *navigation property*). V spodnjih tabelah je predstavljena poenostavljena sestava razerdov:

Red

tip	**lastnost**	**opis**
niz	ZnanstvenoImeReda	znanstveno ime reda ptic, ključ
niz	SlovenskoImeReda	slovensko ime reda ptic
niz	CeloImeReda	celo ime reda ptic sestavljeno iz slovenskega imena in znanstvenega imena v oklepajih, dovoljeno samo branje

niz	OpisReda	opis reda ptic
kolekcija <Vrsta>	Vrste	vrste ptic, povezovalna lastnost

Vrsta

tip	**lastnost**	**opis**
niz	ZnanstvenoImeVrste	znanstveno ime vrste ptic, ključ
niz	SlovenskoImeVrste	slovensko ime vrste ptic
niz	AngleskoImeVrste	angleško ime vrste ptic
niz	CeloImeVrste	celo ime vrste ptic sestavljeno iz slovenskega imena in znanstvenega imena v oklepajih, dovoljeno samo branje
niz	OpisVrste	opis vrste ptic
niz	VelikaSlikaVrste	ime slike vrste ptic originalne ločljivosti
niz	MalaSlikaVrste	ime slike vrste ptic s pomanjšano ločljivostjo
niz	ZnanstvenoImeReda	znanstveno ime reda ptic, tuj ključ
Red	Red	red ptic, povezovalna lastnost
kolekcija <Objava>	Objave	objave, povezovalna lastnost

Regija

tip	**lastnost**	**opis**
celo število	RegijaID	šifra regije, ključ
niz	ImeRegije	ime regije
kolekcija <Obcina>	Obcine	občine, povezovalna lastnost

Obcina - občina

tip	**lastnost**	**opis**
celo število	ObcinaID	šifra občine, ključ
niz	ImeObcine	ime občine
celo število	RegijaID	šifra regije, tuj ključ
Regija	Regija	regija, povezovalna lastnost
kolekcija <Objava>	Objave	objave, povezovalna lastnost

ApplicationUser - uporabnik

tip	**lastnost**	**opis**
niz	Id	šifra uporabnika, ključ
niz	ImeUporabnika	ime uporabnika
niz	PriimekUporabnika	priimek uporabnika
niz	ImeInPriimekUporabnika	ime in priimek uporabnika, dovoljeno samo branje
niz	Email	e-poštni naslov
Boolov	EmailConfirmed	ali je e-pošni naslov potrjen
niz	PasswordHash	kriptirano geslo
kolekcija <Objava>	Objave	objave, povezovalna lastnost
kolekcija <UserRole>	Roles	vloge, povezovalna lastnost

UserRole - vloga uporabnika

tip	**lastnost**	**opis**
niz	RoleId	šifra vloge, skupaj s šifro uporabnika tvori ključ, tuj ključ

niz	UserId	šifra uporabnika, skupaj s šifro vloge tvori ključ, tuj ključ

Role - vloga

tip	**lastnost**	**opis**
niz	Id	šifra vloge, ključ
niz	Name	ime vloge
kolekcija <UserRole>	Users	uporabniki, povezovalna lastnost

Objava - objava slike

tip	**lastnost**	**opis**
celo število	ObjavaID	šifra objave, ključ
niz	VelikaSlikaZaObjavo	ime slike za objavo originalne ločljivosti
niz	MalaSlikaZaObjavo	ime slike za objavo s pomanjšano ločljivostjo
niz	ZemljepisnaSirina	zemljepisna širina lokacije
niz	ZemljepisnaDolzina	zemljepisna dolžina lokacije
datum	DatumNastanka	datum nastanka slike
datum	DatumObjave	datum objave slike
Perje	Perje	lastnosti perja ptic na sliki
Boolov	Potrjena	ali je slika potrjena za objavo s strani moderatorja ali administratorja
niz	UporabnikID	šifra nalagalca oziroma avtorja slike, tuj ključ
niz	ZnanstvenoImeVrste	znanstveno ime vrste ptic, tuj ključ

celo število	ObcinaID	šifra občine, tuj ključ
ApplicationUser	Uporabnik	nalagalec oziroma avtor, povezovalna lastnost
Vrsta	Vrsta	vrsta ptice, povezovalna lastnost
Obcina	Obcina	občina lokacije, povezovalna lastnost

Za opis perja ptic smo določili naštevanje (ang. *enumeration*) **Perje**. Vrednosti za naštevanje **Perje** pa so: jata, mladič, razno, samec, samica.

3.4 Načrtovanje dizajna aplikacije

Za dobro uporabniško izkušnjo je poleg funkcionalnosti aplikacije pomemben tudi njen dizajn. Odločili smo se za moderen minimalistični izgled, ki smo ga realizirali z uporabo knjižnice Bootstrap. Prevladujeta modra in bela barva, ki simbolizirata nebo in oblake. Zaradi boljšega kontrasta smo izbrali temnejšo modro. Pravtako smo z uporabo knjižnice Bootstrap zagotovili odziven dizajn, kar omogoča, da je uporabniški vmesnik spletne aplikacije prilagodljiv in ima lep prikaz na napravah z različno velikimi zasloni.

V naslednjem poglavju je predstavljen razvoj aplikacije. Kot zadnja koraka razvoja smo določili še testiranje in s tem odpravljanje napak.

Poglavje 4

Razvoj aplikacije

V tem poglavju je predstavljen razvoj strežniškega dela aplikacije ter prikazan izgled aplikacije na strani odjemalca.

4.1 Začetek projekta v Visual Studiu

V Visual Studiu smo uporabili predlogo začetnega projekta za spletne aplikacije (ang. *ASP.NET Core Web Application*) s predlogo individualnih uporabniških računov (ang. *Individual User Accounts*). Tako je bil avtomatsko ustvarjen del izvorne kode za zagon aplikacije in uporabo uporabniških računov.

4.2 Implementacija podatkovnega modela

Pri implementaciji podatkovnega modela smo sami naredili razrede **Red**, **Vrsta**, **Regija**, **Obcina** in **Objava**, ki opisujejo podatkovni model naše aplikacije. Z uporabo predloge individualnih uporabniških računov so bili ustvarjeni tudi drugi razredi, ki zagotavljajo delovanje uporabniških računov. Ob prvem zagonu aplikacije jih EF pretvori v tabele v relacijski podatkovni bazi SQL Server LocalDB. Atribute posameznega entitetnega tipa v konceptualnem modelu smo predstvili z lastnostmi v razredih C#. Lastnosti imajo

definiran podatkovni tip ter še dodatna pravila z uporabo anotacij. Tako so na enem mestu določena pravila modela, ki veljajo povsod v aplikaciji. Pri uporabi modela se glede na ta pravila izvede tudi verifikacija vnesenih podatkov na strani odjemalca, če ima ta omogočen JavaScript oziroma na strani strežnika, če je JavaScript onemogočen. V kodi spodaj je prikazan primer uporabe anotacij v razredu **Vrsta** za lastnost **ZnanstvenoImeVrste**:

```
[Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
[StringLength(50,
  ErrorMessage = "Dolžina je lahko največ 50 znakov.")]
[Display(Name = "Znanstveno ime vrste")]
public string ZnanstvenoImeVrste { get; set; }
```

Zatem smo naredili razred **ApplicationDbContext**, ki služi povezovanju s podatkovno bazo. Nato smo implementirali še razred **DbInitializer**, ki ob prvem zagonu aplikacije ustvari in napolni podatkovno bazo z redovi ptic, vrstami ptic, regijami, občinami, uporabniškimi vlogami in administratorskim uporabniškim računom. Ob prvem zagonu je dodanih 21 redov ptic, več kot 80 vrst ptic, 12 regij in 212 občin. Vrste ptic imajo za lažje določanje vrste tudi simbolično sliko. Večino slik vrst ptic smo dobili na spletni strani pixabay [12], ki ponuja slike za brezplačno uporabo.

4.3 Konfiguracija aplikacije

V razredu **Stratup** v metodi ***ConfigureServices*** so nastavljene naslednje storitve, lastnosti in zahteve:

- kontekst podatkovne baze iz **ApplicationDbContext**, prek katerega dostopamo do podatkov v bazi, Connection string je v datoteki **appsettings.json**;
- MVC;

- vedno zahtevaj protokol HTTPS (ang. *Hypertext Transfer Protocol Secure*);
- vedno zahtevaj avtentikacijo – tak je privzet dostop za vse krmilnike, drugačen način določimo z anotacijami nad krmilniki ali akcijami;
- Identity – uporabniški računi in vloge;
- pošiljanje e-pošte.

Vse zgoraj našteto je potem na voljo v celotni izvorni kodi prek injiciranja odvisnosti (ang. *dependency injection*). Kontekst podatkovne baze, upravitelja uporabniških računov in še nekatere druge storitve je potrebno injicirati eksplicitno, ostale storitve pa so injicirane avtomatsko. Za uporabo storitev v krmilnikih jih je potrebno navesti kot parametre v konstruktorju. Injiciramo jih lahko tudi v pogledih.

V kodi spodaj je prikazan primer injiciranja v krmilniku **ObjaveController**:

```
private readonly ApplicationDbContext _context;
private readonly IHostingEnvironment _environment;
private readonly UserManager<ApplicationUser> _userManager;

public ObjaveController(ApplicationDbContext context,
  IHostingEnvironment environment,
  UserManager<ApplicationUser> userManager)
  {
    _context = context;
    _environment = environment;
    _userManager = userManager;
  }
```

Spodnja koda pa prikazuje primer injiciranja v pogledih:

```
@inject UserManager<ApplicationUser> UserManager
```

V razredu **Stratup** v metodi ***Configure*** je med drugim določeno, kako strežnik obravnava zahtevke oziroma naslov URL in kako mora biti ta sestavljen. Določeno je, kako se navede krmilnik, njegove akcije in parametre. V kodi spodaj je prikazan primer sestavljanja naslova URL s pomočjo pomočnikov za ustvarjanje elementov HTML (ang. *tag helpers*) za razvrščanje seznama uporabnikov po priimku naraščajoče oziroma padajoče v pogledu **ApplicationUsers/Index.cshtml**:

```
<a asp-action="Index"
  asp-route-trenutnoIskanje=@Model.Iskanje
  asp-route-samoModeratorji=@Model.SamoModeratorji
  asp-route-razvrscanje=@Model.RazvrscanjePoPriimku
  asp-route-velikostStrani=@Model.VelikostStrani>
```

V razredu **MessageServices** smo na enem mestu definirali pošiljanje e-pošte z uporabo MailKita. Prek protokola SMTP in vrat 587 se z uporabo varne povezave TLS (ang. *Transport Layer Security*) povežemo z Gmailom. Storitev uporabljamo pri potrjevanju registracije in ponastavljanju pozabljenega gesla.

4.4 Organizacija in dizajn spletne strani

S pomočjo knjižnice Bootstrap smo oblikovali uporabniški vmesnik. Najprej smo zasnovali menijsko vrstico, ki je prisotna na vrhu vsake strani aplikacije ter domačo stran. Menijska vrstica, ki je prikazana na sliki 4.1, vsebuje povezave na posamezne dele aplikacije.

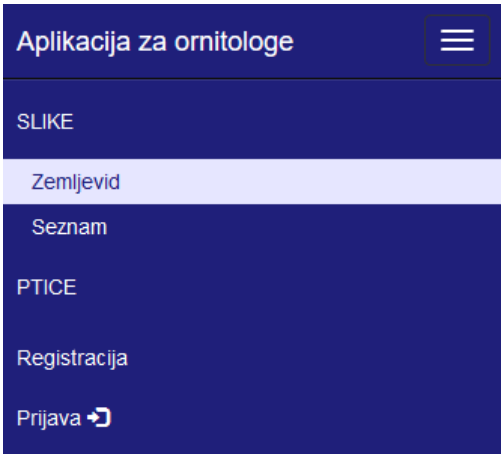
Slika 4.1: Menijska vrstica na vrhu vsake strani.

Elementi menijske vrstice od leve proti desni:

- Skrajno levo je povezava **Aplikacija za ornitologe**, ki vodi na domačo stran, kjer so kratka navodila za uporabo aplikacije.
- Meni **SLIKE** vsebuje izvlečni meni s povezavama **Zemljevid** in **Seznam**, ki vodita na pregledovanje objavljenih slik na zemljevidu in po seznamu.
- Meni **PTICE** vsebuje izvlečni meni s povezavama **Redovi** in **Ptice**, ki vodita na pregledovanje redov oziroma vrst ptic.
- Povezava **UPORABNIKI**, ki je vidna samo administratorju in vodi na stran, ki omogoča upravljanje z uporabniki.
- Povezava **OBNOVA**, ki je vidna samo administratorju in vodi na stran, ki omogoča upravljanje z varnostnimi kopijami objav slik.
- Povezava **Registracija**, ki vodi na registracijo novega uporabnika. Na sliki 4.1 ta povezava ni vidna, prikaže pa se v primeru, če obiskovalec strani ni prijavljen.
- Povezavi **Prijava** ali **Odjava**, ki sta namenjeni prijavi v aplikacijo oziroma odjavi.

Z uporabo predloge Bootstrap smo dosegli, da se menijska vrstica prilagodi trenutni velikosti zaslona, kar zagotavlja dobro uporabniško izkušnjo na različnih napravah. Prilagoditev menijske vrstice napravi z manjšim zaslonom prikazuje slika 4.2. Pri napravah z manjšimi zasloni, se elementi menijske vrstice skrijejo za tako imenovano ikono „hamburger“, ki je sestavljena iz več horizontalnih črt in je na sliki 4.2 vidna v zgornjem desnem kotu. S klikom na to ikono se odpre vertikalni meni.

Del dokumenta HTML, ki je na vseh straneh enak ter med drugim vsebuje menijsko vrstico in nogo strani, smo definirali na enem mestu v datoteki **_Layout.cshtml**.



Slika 4.2: Menijska vrstica na manjših zaslonih z izbranim menijem **SLIKE** in fokusom nad povezavo **Zemljevid**.

Pri pregledovanju podatkov so vedno na voljo različni filtri, s katerimi si lahko uporabnik pomaga skrčiti svoje iskanje. Seznami so razdelji po straneh (ang. *paging*) in imajo možnost razvrščanja po vseh stolpcih. Izbrano opcijo razvščanja nam pove puščica dol ali gor ob imenu stolpca za padajoče oziroma naraščajoče razvrščanje.

V aplikaciji smo obarvali gumbe, katerih akcije lahko sprožijo spremembo stanja podatkov ali pa je njihova akcija končna (npr. gumb za prenos slike). Ostali gumbi, ki šele vodijo na neko stran, so večinoma bele barve. Gumb, ki vodi na stran za nalaganje nove slike, pa je obarvan zato, da bolj izstopa.

4.5 Varnost

Pri ravnanju z večino podatkov potrebujemo operacije ustvari, beri, posodobi in izbriši oziroma operacije CRUD (ang. *Create, Read, Update, Delete*). Za krmilnike in poglede smo začetno izvorno kodo za operacije CRUD nad podatki generirali avtomatsko z uporabo tako imenovanega „odra“ (ang. *scaffolding*). Dodali smo ostale manjkajoče akcije in omejitve, s katerimi smo določili kdo lahko akcije izvede. Pri preverjanju ali ima uporabnik vlogo mo-

deratorja ali administratorja smo bili pozorni, da se pri negiranju tega izraza logični operator ALI spremeni v logični IN.

Spodaj je prikazan izraz za preverjanje ali ima uporabnik vlogo moderatorja ali administratorja:
`if(User.IsInRole("Administrator")||User.IsInRole("Moderator"))`

Spodaj pa je prikazan izraz za preverjanje ali uporabnik nima vloge moderatorja in tudi nima vloge administratorja:
`if(!User.IsInRole("Administrator")&&!User.IsInRole("Moderator"))`

Za omejitev dostopa do akcij v krmilniku, je potrebno v pogledih in tudi v krmilnikih preverjati kdo je prijavljeni uporabnik. Tako ni mogoče priti do poljubnih akcij z ročnim vnosom naslova URL. V primeru zahtevka na strežnik, za katerega uporabnik nima pravic, se mu prikaže opozorilo. V krmilnikih smo programsko kodo popravili tako, da so vse operacije varne:

- Validacija vnesenih podatkov na strani odjemalca in strežnika.
- Preverjanje vlog in šifer uporabnikov v krmilnikih in pogledih.
- Uporaba **ValidateAntiForgeryToken** pri akcijah tipa POST, kar preprečuje ponarejanje zahteve (ang. *cross-site request forgery*).
- Uporaba vezanja lastnosti (ang. *bind*) pri kreiranju novih entitet. S tem smo preprečili nedovoljeno nastavljanje lastnosti (ang. *overposting attack*). Z vezanjem določimo, katere lastnosti objekta lahko nastavljamo. Pri akcijah za posodabljanje entitet pa smo uporabili drugačen pristop, saj z vezanjem pobrišemo vrednosti lastnosti, ki niso bile posodobljene.
- Uporaba modelov pogledov (ang. *view model*, močno tipiziran razred za predstavitev podatkov v pogledu) ali pa metode ***TryUpdateMo-***

delAsync pri posodabljanju entitet, da se posobijo samo dovoljene lastnosti.

Za pošiljanje podatkov, ki spreminjajo podatkovni model aplikacije, se vedno uporabljajo metode POST. Za strani, ki pa samo prikazujejo podatke, na primer seznam objav slik, pa se uporabljajo metode GET. Zahtevke GET damo lahko med zaznamke. Tako si lahko med zaznamke shranimo na primer stran za prikaz objav slik z vsemi filtri, ki smo jih nastavili.

Krmilniki ponavadi posredujejo podatke pogledom v obliki modela ali pa modela pogleda, ker imajo ti močno tipizirane lastnosti in so zato bolj varni. Ponekod podatke posredujemo tudi v slovarju objektov ***ViewData***, ki ni tipiziran in moramo biti zato bolj pozorni pri uporabi.

Izvorno kodo, med katero bi lahko prišlo do napak, smo vstavili v blok ***try–catch*** in s tem ulovili napake.

V primeru, da pride do večjega izbrisa objav slik iz različnih razlogov, lahko vnose objav v podatkovni bazi in datoteke slik obnovimo z uporabo varnostne kopije.

4.6 Učinkovitost in zanesljivost

Strežnik ima omejeno število niti. Ponavadi mora čakati, da se zaključi vnos podatkov. Zato je koda krmilnikov asinhrona, da lahko med čakanjem streže druge zahtevke. Pri veliki zasedenosti strežnika nam to pride prav. Pri akcijah v krmilnikih smo za tip vrnjenih vrednosti (ang. *return type*) uporabili ***async Task<IActionResult>***. Znotraj akcij smo uporabili asihrone metode pri dostopanju do baze. Ker EF kontekst ni varen pri uporabi niti (ang. *thread safe*), je treba pred poizvedbami uporabljati klic ***await***, s katerim počakamo, da se kritična akcija konča.

EF implicitno uporablja transakcije. Ko kličemo metodo ***SaveChangesAsync*** se izvedejo vse narejene spremembe entitet ali pa nobena.

Boljše delovanje aplikacije smo dosegli tudi z zmanjševanjem ločljivosti slik, za kar smo uporabili MagicScaler. Pri nalaganju slike se shrani tudi

slika z zmanjšano ločljivostjo, katere velikost (v bytih) je ponavadi manjša za več kot 99%. Manjšo sliko uporabljmo za prikaz na zemljevidu in seznamu, s čimer zelo zmanjšamo promet po omrežju, ki bi nastal zaradi velikega števila velikih slik.

Pri branju in pisanju iz virov smo tokove definirali v izrazu ***using***, s čimer zagotovimo, da se po uporabi avtomatsko sprostijo s klicem metode ***Dispose***.

4.7 Registracija, prijava, odjava, ponastavitev pozabljenega gesla

Tukaj so opisane strani, ki omogočajo prijavo oziroma registracijo v aplikacijo.

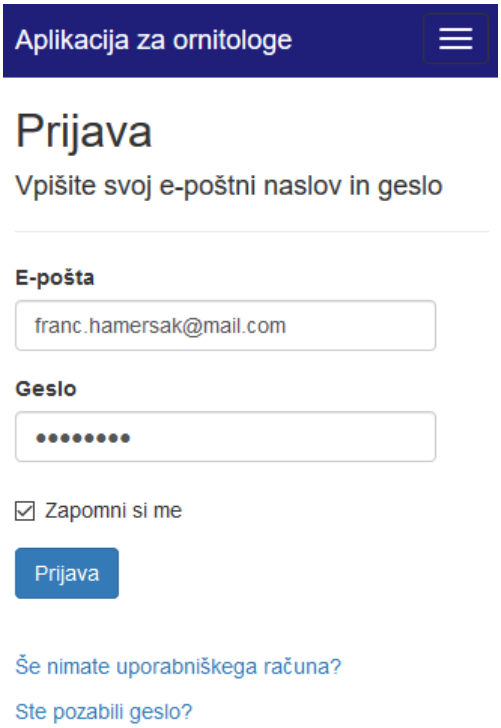
4.7.1 Registracija

Za registracijo mora uporabnik obiskati povezavo **Registracija**. Odpre se mu stran za registracijo, kjer izpolni zahtevane podatke ter potrdi registracijo s klikom na gumb **Registracija**. V primeru napačne oblike e-poštnega naslova, neustrezne dolžine in znakov gesla, napačne ponovitve gesla, vnosa že uporabljenega e-poštnega naslova ali katere druge napake se izpišejo opozorila. V primeru uspešne registracije, se uporabniku pošlje e-poštno sporočilo v katerem je povezava za potrditev registracije. S tem preprečimo, da bi nekdo za registracijo uporabil e-poštni naslov druge osebe.

4.7.2 Prijava in odjava

Za prijavo v aplikacijo mora uporabnik obiskati povezavo **Prijava**. Odpre se mu stran za prijavo, kjer vnese svoj e-poštni naslov in geslo (slika 4.3). Pred prvo prijavo se je potrebno registrirati in potrditi registracijo. Če uporabnik izbere možnost, da si ga aplikacija zapomni, bo ob zaprtju in ponovnem

odprtju brskalnika ostal prijavljen v aplikacijo. Za odjavo iz aplikacije mora uporabnik klikniti na povezavo **Odjava**.



Slika 4.3: Del strani za prijavo v aplikacijo.

4.7.3 Ponastavitev pozabljenega gesla

Za ponastavitev pozabljenega gesla mora uporabnik obiskati povezavo **Ste pozabili geslo?**, ki se nahaja na strani za prijavo (slika 4.3 spodaj). Odpre se mu stran, kjer lahko vpiše svoj e-poštni naslov. Če je naslov v bazi uporabnikov, se mu pošlje sporočilo s povezavo na stran, kjer lahko na novo nastavi svoje geslo.

4.8 Redovi in vrste ptic

V tem razdelku so predstavljene strani, ki omogočajo pregledovanje redov in vrst ptic.

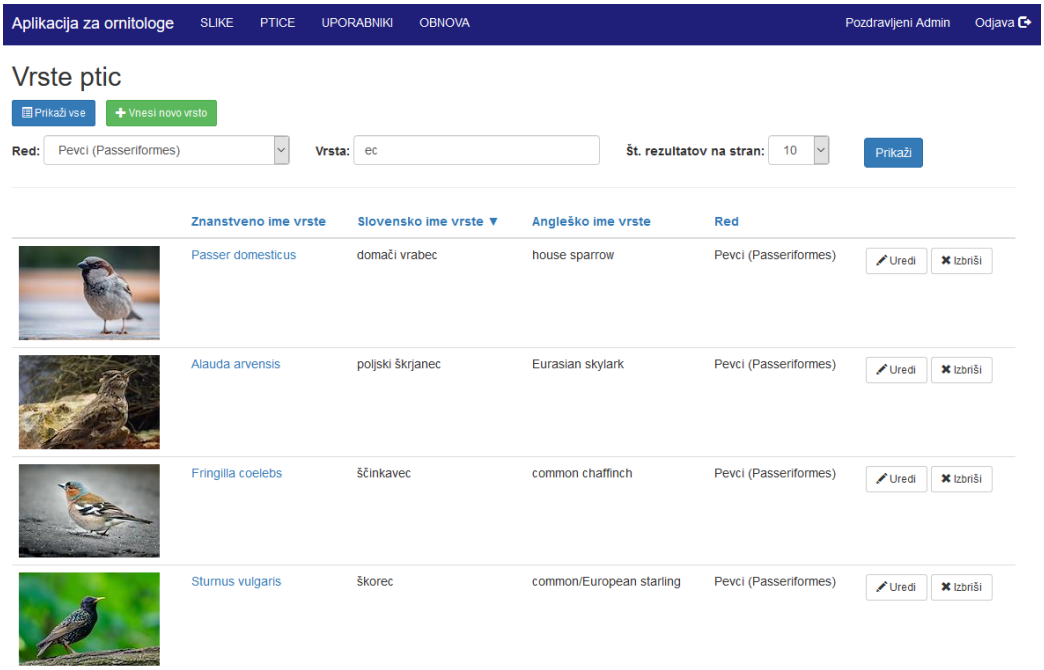
4.8.1 Redovi ptic

Za pregledovanje redov ptic, mora uporabnik obiskati povezavo **Redovi**, ki se nahaja v meniju **PTICE**. Odpre se mu stran, kjer je prikazan seznam redov ptic. Za vsak red ptic je prikazano znanstveno in slovensko ime reda. Seznam se lahko razvršča po vseh stolpcih. Za prikaz seznama so na voljo različni filtri. Uporabnik lahko išče redove ptic po znanstvenem ali slovenskem imenu in izbira število prikazanih rezultatov na stran. Filtre potrdi z gumbom **Prikaži**, ponastavi pa z gumbom **Prikaži vse**. S klikom na znanstveno ime posameznega reda ptic se mu odpre stran z vsemi podatki o tem redu. Če je uporabnik administrator, ima prikazan gumb **Vnesi nov red**, ki vodi na stran za vnos novega reda ptic. Administrator ima na skrajnem desnem robu seznama tudi gumba **Uredi** in **Izbriši**, ki vodita na stran za urejanje podatkov o redu ptic oziroma na stran za brisanje reda ptic.

4.8.2 Vrste ptic

Za pregledovanje vrst ptic, mora uporabnik obiskati povezavo **Vrste**, ki se nahaja pod zavihkom **PTICE**. Odpre se mu stran, kjer je prikazan seznam vrst ptic (slika 4.4). Za vsako vrsto ptic je prikazana simbolična slika, znanstveno ime vrste, slovensko ime vrste, angleško ime vrste ter slovensko ime reda z znanstvenim imenom reda v oklepajih. Seznam se lahko razvršča po vseh stolpcih, razen po sliki. Za prikaz seznama so na voljo različni filtri. Uporabnik lahko izbira red ptic, išče vrste ptic po znanstvenem, slovenskem ali angleškem imenu in izbira število prikazanih rezultatov na stran. Filtre potrdi z gumbom **Prikaži**, ponastavi pa z gumbom **Prikaži vse**. S klikom na znanstveno ime posamezne vrste ptic ali pa simbolično sliko se odpre stran z vsemi podatki o tej vrsti. Če je uporabnik administrator, ima prikazan gumb

Vnesi novo vrsto, ki vodi na stran za vnos nove vrste ptic. Administrator ima na skrajnem desnem robu seznama tudi gumba **Uredi** in **Izbriši**, ki vodita na stran za urejanje podatkov o vrsti ptic oziroma na stran za brisanje vrste ptic.



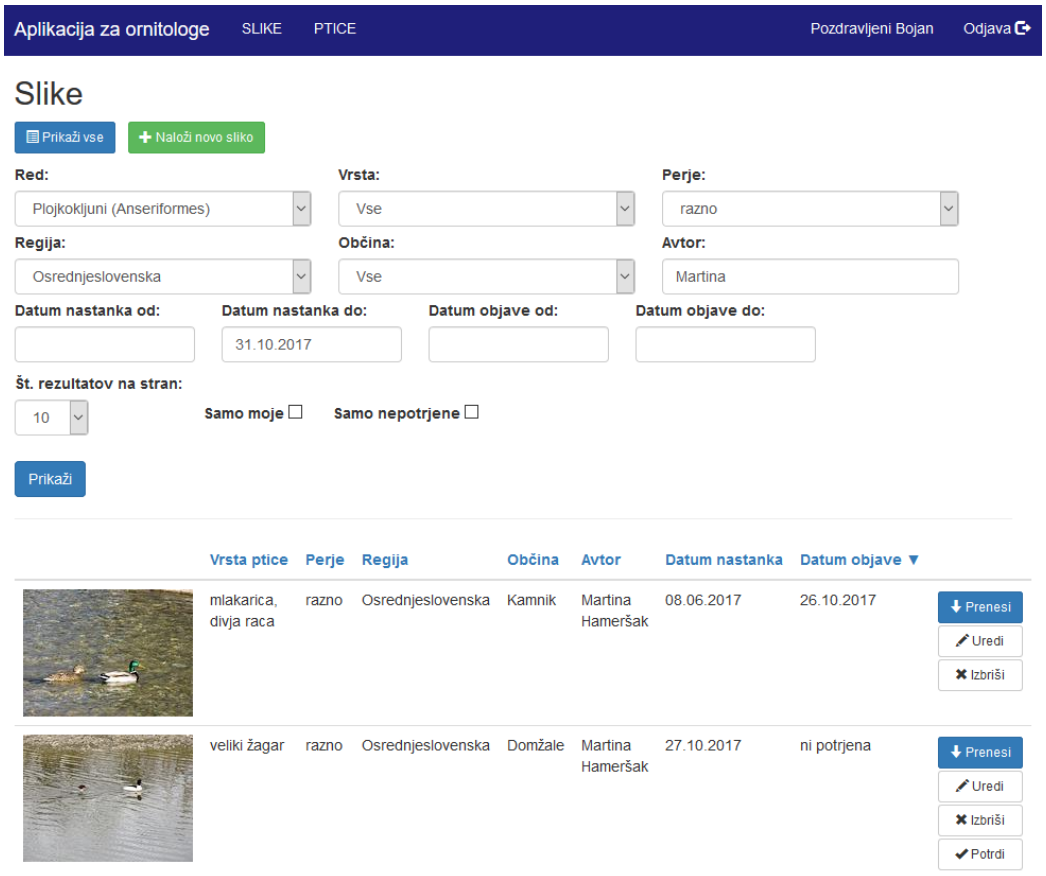
Slika 4.4: Del strani seznama vrst ptic z razvrščanjem po slovenskem imenu vrste ptic.

4.9 Objave slik

Za pregledovanje objav slik, mora uporabnik obiskati povezavo **Zemljevid** ali pa **Seznam**, ki se nahajata v meniju **SLIKE**. Odpre se mu stran, kjer so prikazane objave slik na zemljevidu ali pa po seznamu.

4.9.1 Pregledovanje objav slik po seznamu

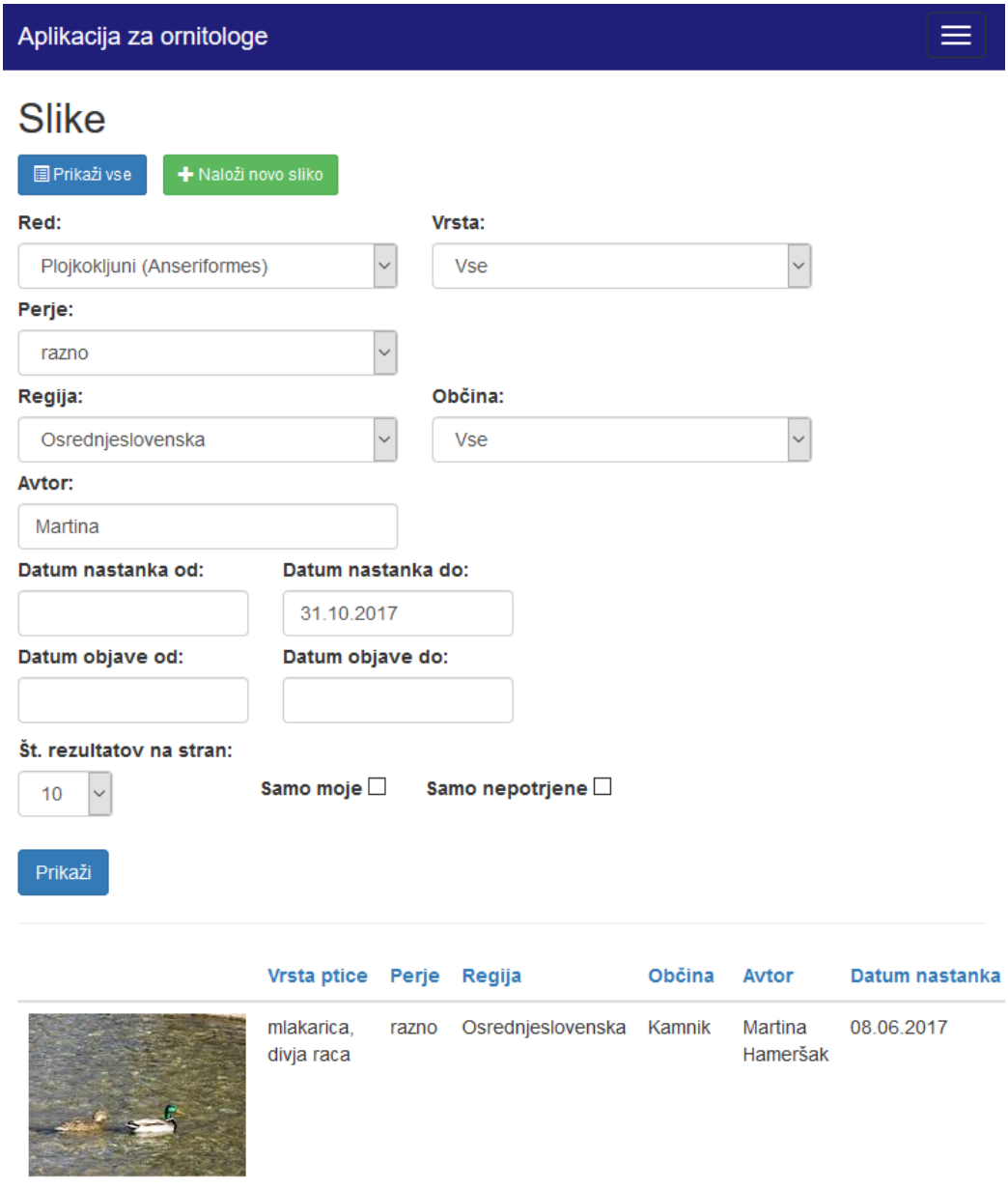
Pri pregledovanju objav slik po seznamu je za vsako objavo prikazana slika, slovensko ime vrste ptic, lastnost perja, regija, občina, avtor, datum nastanka ter datum objave. Primer takega seznama je prikazan na sliki 4.5.



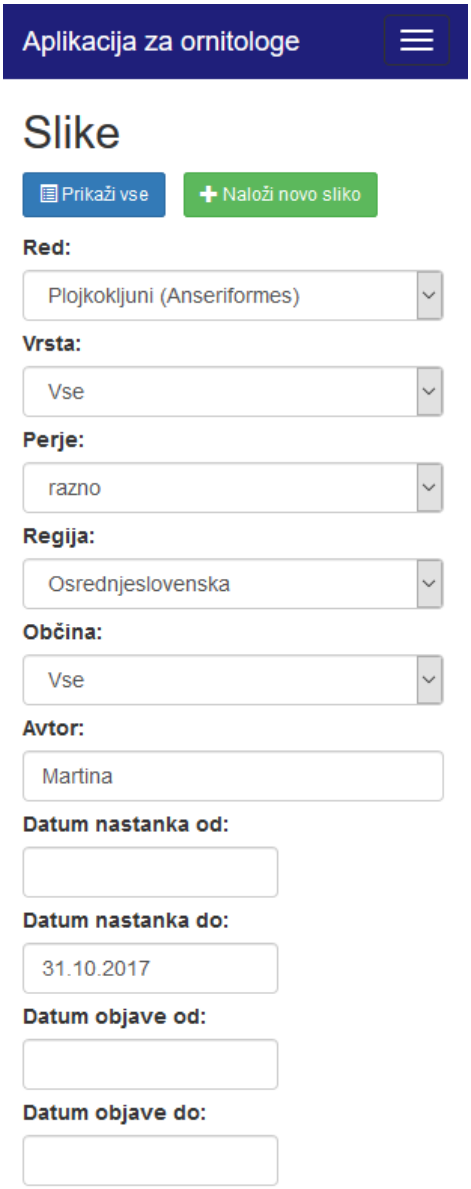
Slika 4.5: Del strani seznama objav slik prikazan na velikem zaslonu.

Seznam se lahko razvršča po vseh stolpcih razen po sliki. Za prikaz seznama so na voljo različni filtri. Uporabnik lahko izbira red ptic, glede na izbor reda lahko izbira tudi vrsto ptic, izbira lahko lastnost perja, regijo, glede na izbor regije lahko izbira tudi občino, išče lahko avtorja, določa datum nastanka ter objave slik in izbira število prikazanih rezultatov na stran. Filtre potrdi z gumbom **Prikaži**, ponastavi pa z gumbom **Prikaži vse**. Tako kot vse

strani v naši aplikaciji ima tudi seznam objav slik odziven dizajn. Sliki 4.6 in 4.7 prikazujeta izgled aplikacije na napravah s srednjim oziroma z majhnim zaslonom. S klikom na sliko se odpre stran z vsemi podatki o tej objavi.



Slika 4.6: Del strani seznama objav slik prikazan na srednjem zaslonu.



Slika 4.7: Del strani seznama objav slik prikazan na majhnem zaslonu.

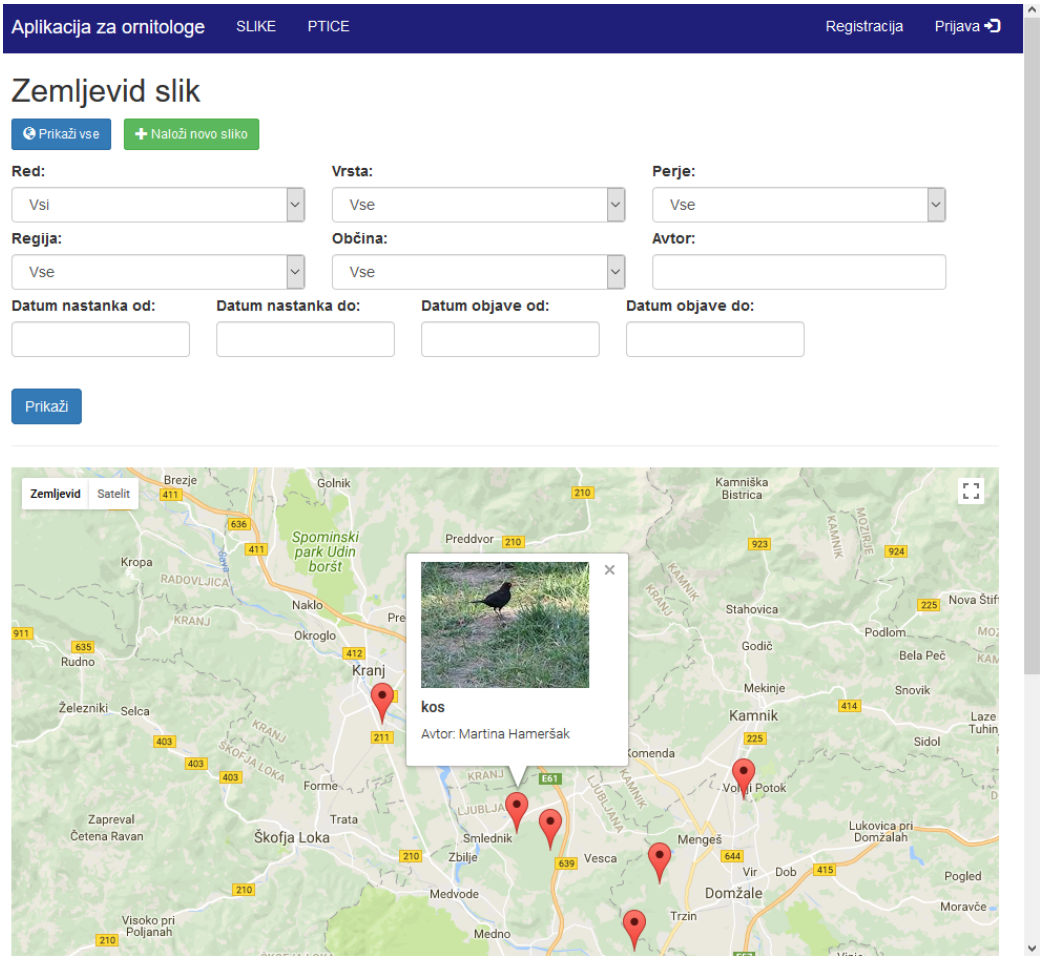
Vedno je prikazan gumb **Naloži novo sliko**, ki vodi na stran za nalaganje nove slike za objavo (stran za nalaganje je opisana spodaj). Za nalaganje slik mora biti uporabnik prijavljen v aplikacijo. Na desnem robu seznama je gumb **Prenesi**, s katerim lahko vsak prenese sliko polne ločljivosti na svojo

napravo. Če je uporabnik prijavljen v aplikacijo, ima na voljo dodaten filter, s katerim lahko prikaže samo svoje objave slik. Če je uporabnik v vlogi moderatorja ali administratorja, ima tudi filter za prikaz samo nepotrjenih objav. Prijavljen uporabnik ima na skrajnem desnem robu seznama tudi gumba **Uredi** in **Izbriši**, ki vodita na stran za urejanje podatkov o objavi slike oziroma na stran za brisanje objave slike. Moderator in administrator imata tudi gumb **Potrdi**.

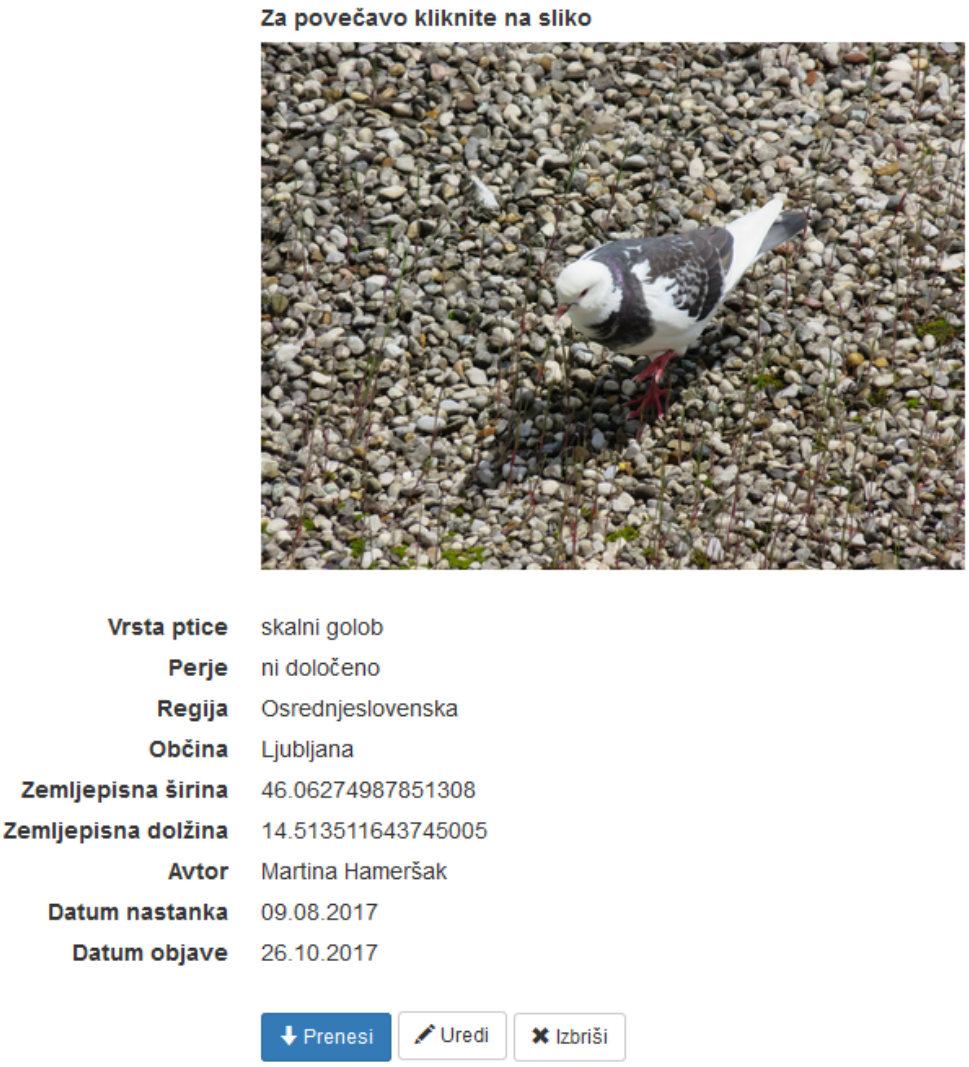
Preden je objava slike na ogled vsem obiskovalcem strani, jo mora potrditi moderator ali administrator. S tem preprečimo zlorabo strani za objavljanje slik z drugačnimi vsebinami. Dokler objava slike ni potrjena, je vidna samo avtorju, moderatorjem in administratoju.

4.9.2 Predogled objav slik na zemljevidu

Pri predogledu objav slik na zemljevidu, so na zemljevidu Google Maps prikazane točke nastanka slik, kot lahko vidimo na sliki 4.8. Za prikaz točk na zemljevidu so na voljo enaki filtri kot za prikaz po seznamu, razen števila rezultatov na stran. S klikom na točko na zemljevidu se uporabniku prikaže mala slika s podatki o vrsti ptice in avtorju slike, s klikom na sliko pa se mu odpre stran z vsemi podatki o tej objavi, ki jo prikazuje slika 4.9. Tudi na tej strani so lahko prikazani gumbi **Prenesi**, **Uredi**, **Izbriši** in **Potrdi**.



Slika 4.8: Del strani za predogled objav na zemljevidu.

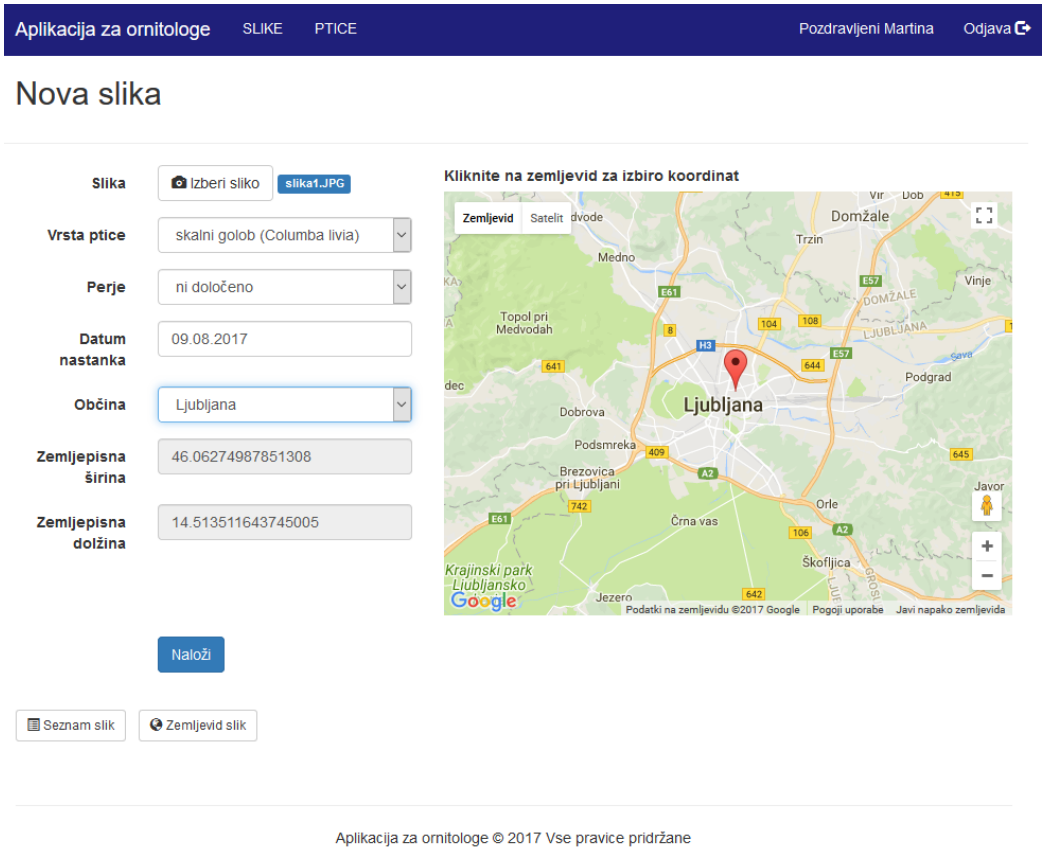


Slika 4.9: Del strani s podrobnostmi objave slike.

4.9.3 Nalaganje slik za objavo

Na sliki 4.10 je prikazana stran za nalaganje slik, ki jih želi uporabnik objaviti. Uporabnik z raziskovalcem izbere slikovno datoteko, s klikom na zemljevid določi koordinate lokacije nastanka slike ter izpolni ostale podatke. V primeru pravilnega vnosa podatkov se s klikom na gumb **Naloži** ustvari nova

objava, ki vsebuje tudi sliko z zmanjšano ločljivostjo. Slike vsakega uporabnika se shranjujejo v svojo mapo, pred imenom pa se jim doda časovni žig (ang. *timestamp*). Tako vemo, kdaj je bila slika naložena na strežnik, preprečimo pa tudi prekrivanje imen datotek.

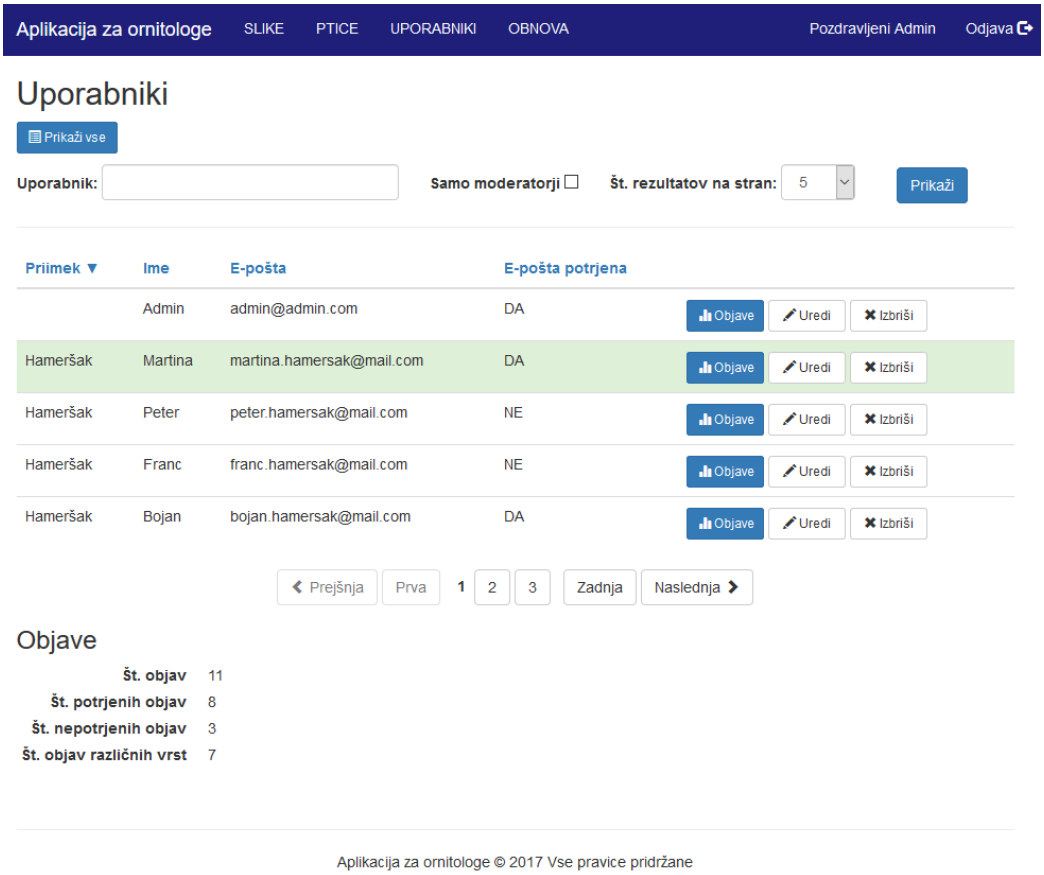


Slika 4.10: Stran za nalaganje nove slike za objavo s fokusom na oknu za občino, ki je zato osvetljeno.

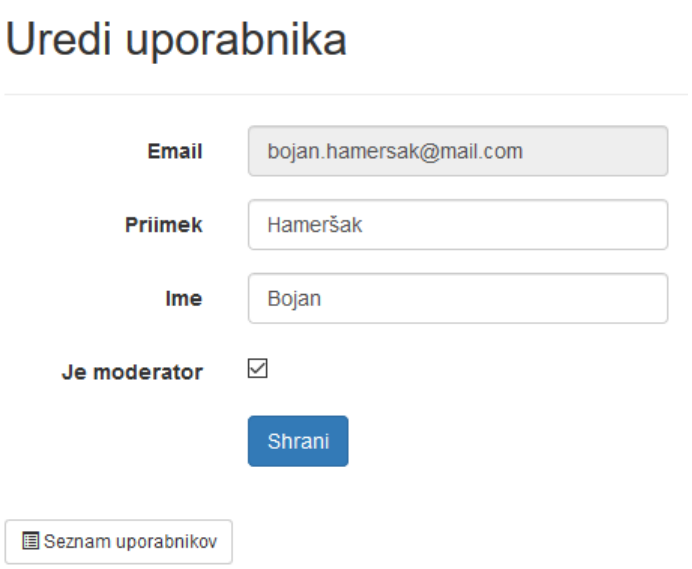
4.10 Upravljanje z uporabniki

Uporabnike lahko pregleduje in z njimi upravlja samo administrator. Seznam uporabnikov se nahaja na strani **UPORABNIKI**. Za vsakega uporabnika je prikazan priimek, ime, e-poštni naslov ter ali je naslov potrjen (slika 4.11).

Seznam se lahko razvršča po vseh stolpcih. Na voljo so filtri za iskanje uporabnika, prikaz samo moderatorjev in izbiro števila rezultatov na stran. Na skrajnem desnem robu seznama so gumbi **Objave**, **Uredi** in **Izbriši**. Gumb **Objave** na isti strani prikaže statistiko objav slik posameznega uporabnika. Gumb **Uredi** vodi na stran za urejanje podatkov o uporabniku, ki je prikazana na sliki 4.12. Na tej strani lahko uporabniku tudi dodelimo vlogo moderatorja. Gumb **Izbriši** pa vodi na stran za izbris uporabnika.



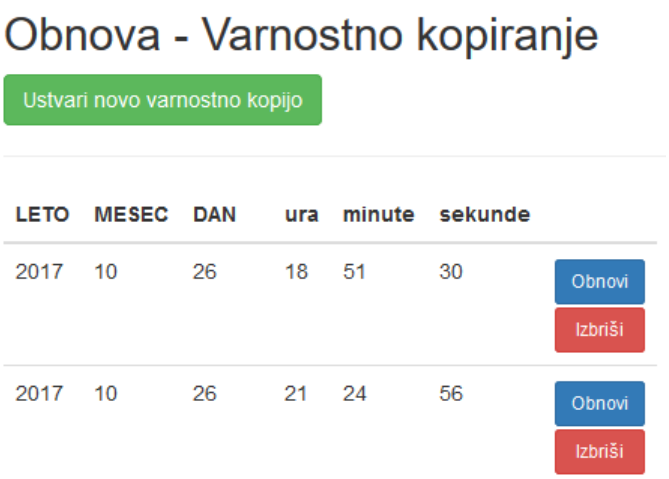
Slika 4.11: Stran za upravljanje uporabnikov.



Slika 4.12: Del strani za upravljanje uporabnika.

4.11 Varnostno kopiranje

Z varnostnimi kopijami objav slik lahko upravlja samo administrator. Seznam varnostnih kopij se nahaja na strani **OBNOVA** (slika 4.13). S klikom na gumb **Ustvari novo varnostno kopijo** se ustvari mapa s časovnim žigom, v katero se skopirajo vse datoteke objav slik, ustvari pa se tudi tekstovna datoteka, v katero se shranijo stavki SQL s katerimi obnovimo stanje objav v podatkovni bazi. Z gumbom **Obnovi** obnovimo stanje objav na stanje v varnostni kopiji. Z gumbom **Izbriši** pa izbrišemo varnostno kopijo.



Slika 4.13: Del strani za obnovitev stanja objav slik.

4.12 Testiranje

Zadnja faza razvoja je testiranje in nato še seveda odprava napak. Napak ni bilo veliko, saj smo aplikacijo že sproti podrobno testirali in odpravljali ugotovljene napake. Aplikacijo smo pokazali tudi nekaj prijateljem, ki so pokomentirali delovanje in izgled. Na podlagi njihovih mnenj smo jo še dokončno dopolnili.

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge smo razvili aplikacijo, ki je namenjena ljubiteljem opazovanja ptic. Glavne funkcionalnosti, ki smo jih želeli imeti v okviru naše aplikacije, so možnost objavljanja in deljenja slik z ostalimi, prikaz slik skupaj z lokacijo nastanka na zemljevidu, iskanje z uporabo raznih filtrov, tudi glede na določeno regijo, in enostavna uporaba. K razvoju nas je vzpodbudilo to, da med različnimi aplikacijami, ki bi jih lahko uporabili v ta namen, nismo zasledili nobene, ki bi v popolnosti zadostila našim potrebam in zahtevam. Obstajajo razne aplikacije, ki jih je moč uporabiti za namene opazovanja ptic, vendar ponavadi ne vključujejo vseh želenih funkcionalnosti, prav tako pa so lahko tudi neprimerne oziroma težke za uporabo.

5.1 Težave

Trenutno je aplikacija še v testni fazi ter še ni javno objavljena in dostopna. Vendar pričakujemo, da bo po odpravi težav zaživela in služila svojemu namenu. Trenutno je najbolj pereča težava prenašanje slik med uporabnikom in strežnikom. Ker prenašanje slik povzroča več prometa po omrežju, je potrebno še poiskati ustrezno rešitev za namestitev aplikacije v produkcijsko okolje. Vsekakor pa je želja že od samega začetka, da je aplikacija javno dostopna in da jo uporablja čim večje število uporabnikov.

Med razvojem ni bilo nepremostljivih težav. Zasluga za to gre med drugim tudi programerski skupnosti zbrani okrog uporabljenega ogrodja, saj je na spletu dovolj primerov uporabe ogrodja ASP.NET Core MVC. Nekaj več težav je bilo s pošiljanjem e-pošte, ker je bilo potiho blokirano s strani antivirusnega programa, verjetno z namenom preprečevanja širjenja neželene e-pošte. Nekaj zapletov je bilo tudi z datumi, ki jih shranjujemo v slovenskem formatu. Pri ustvarjanju varnostne kopije smo morali ta format prilagoditi standardu SQL Serverja, zato da pri obnavljanju stanja objav slik ni prišlo do napake formata datuma.

5.2 Možne nadgradnje

Razvito aplikacijo je možno v prihodnje še nadgraditi in ji dodati nove funcionalnosti, na primer:

- Avtomatsko branje lokacije iz slik, če je ta informacija na voljo. Poleg mobilnih telefonov tudi nekateri fotoaparati zapišejo koordinate lokacije GPS (ang. *Global Positioning System*).

- Če je več slik na točno istih koordinatah, nam pri pregledovanju na zemljevidu prikaže samo zadnjo. Izboljšava bi bila, da nam prikaže seznam.

- Možnost nalaganja avdio in video posnetkov.

- Razširitev aplikacije tudi za druge države, dodanje regij sosednjih držav.

- Nadgradnja aplikacije na ta način, da bi omogočala uporabo tudi za kakšno drugo tematiko, ki bi potrebovala podobne funkcionalnosti. Lahko bi namesto ptic beležili kake druge živali ali rastline.

- Zanimiva funkcionalnost, vendar tudi težka za implementiranje, bi bila tudi zmožnost avtomatskega prepoznavanja vrste ptic.

V okviru diplome smo razvili spletno aplikacijo namenjeno ornitologom, postavlja pa se vprašanje, če bi bila smiselna implementacija aplikacije v obliki mobilne aplikacije. Med drugim obstajajo tovrstne aplikacije tudi v trgovini za telefone. Izkaže pa se, da je trenutno stanje tehnologij pri mobilnih napravah tako, da je odgovor zaenkrat ne. Za prave ljubitelje ptic morajo biti slike, ki so posnete, zares kakovostne. Za dobre slike ptic imajo fotoaparati na pametnih telefonih preslabo optično približanje. Ker so ptice majhne, so ob povečavi take slike slabo vidne oziroma niso dobro vidne podrobnosti ptic na slikah. Samo aplikacijo pa je mogoče uporabljati tudi na mobilnih napravah, saj je uporabniški vmesnik odziven, tako da se prilagaja različno velikim zaslonom in je primeren tudi za uporabo na mobilnih telefonih.

V primeru, da pa se pokaže potreba po mobilni aplikaciji, pa bi lahko implementirali funkcionalnosti slikanja in avtomatskega pošiljanja slik na strežnik ter dodajanja lokacije na podlagi pozicije GPS.

Literatura

[1] Ajax (programming). Dosegljivo: https://en.wikipedia.org/wiki/Ajax_(programming). [Dostopano: 20. 10. 2017].

[2] C Sharp (programming language). Dosegljivo: https://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Dostopano: 20. 10. 2017].

[3] eBird. Dosegljivo: http://ebird.org/content/ebird/. [Dostopano: 20. 10. 2017].

[4] Entity Framework. Dosegljivo: https://en.wikipedia.org/wiki/Entity_Framework. [Dostopano: 20. 10. 2017].

[5] Google Maps Embed API. Dosegljivo: https://developers.google.com/maps/documentation/embed/guide. [Dostopano: 20. 10. 2017].

[6] Introducing MagicScaler. Dosegljivo: http://photosauce.net/blog/post/introducing-magicscaler. [Dostopano: 20. 10. 2017].

[7] JavaScript. Dosegljivo: https://en.wikipedia.org/wiki/JavaScript. [Dostopano: 20. 10. 2017].

[8] Bertrand Le Roy. .NET Core Image Processing. Dosegljivo: https://blogs.msdn.microsoft.com/dotnet/2017/01/19/net-core-image-processing/. [Dostopano: 20. 10. 2017].

[9] Microsoft Visual Studio. Dosegljivo: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Dostopano: 20. 10. 2017].

[10] .NET Core Guide. Dosegljivo: https://docs.microsoft.com/en-us/dotnet/core/. [Dostopano: 20. 10. 2017].

[11] NOAGS. Dosegljivo: http://atlas.ptice.si/atlas/index.php?r=site/page&view=about. [Dostopano: 20. 10. 2017].

[12] Pixabay - Free Images. Dosegljivo: https://pixabay.com/. [Dostopano: 20. 10. 2017].

[13] Programski jezik C sharp. Dosegljivo: https://sl.wikipedia.org/wiki/Programski_jezik_C_sharp. [Dostopano: 20. 10. 2017].

[14] Standard ECMA-334. Dosegljivo: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf. [Dostopano: 20. 10. 2017].

[15] What is managed code? Dosegljivo: https://docs.microsoft.com/en-us/dotnet/standard/managed-code. [Dostopano: 20. 10. 2017].