Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Danijel Mišanović

Priporočilni sistem za nepremičnine na podlagi interesnih

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Aleksander Sadikov

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00521/2013
Datum: 02.09.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DANIJEL MIŠANOVIĆ**

Naslov: **PRIPOROČILNI SISTEM ZA NEPREMIČNINE NA PODLAGI INTERESNIH TOČK**

RECOMMENDER SYSTEM FOR REAL ESTATE BASED ON POINTS OF INTEREST

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Namen diplomske naloge je zgraditi prototipni priporočilni sistem, ki uporabniku svetuje pri nakupu nepremičnine glede na bližino interesnih točk (POI, angl. points of interest). Kandidat naj pridobi (v dogovoru s podjetjem Monolit d.o.o.) podatke o interesnih točkah in aktualni ponudbi nepremičnin za področje Ljubljane ter podatke prilagodi v primerno obliko za uporabo v spletnem priporočilnem sistemu. Kandidat se seznani z načinom hranjenja geografskih podatkov (PostGres podatkovna baza z razširitvijo PostGis). Obenem naj kandidat preuči primerno uporabo takšnih podatkov za priporočanje nepremičnin, podatke poveže s tipičnimi značilnostmi nepremičnin (npr. cena, velikost, ipd.) ter zasnuje ustrezen način uporabe in uporabniški vmesnik takšnega priporočilnega/odločitvenega sistema. Prav tako zasnuje ustrezen način predstavitve in analize rezultatov sistema.

Mentor:

viš. pred. dr. Aleksander Sadikov

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Danijel Mišanović, z vpisno številko **63080302**, sem avtor diplomskega dela z naslovom:

Priporočilni sistem za nepremičnine na podlagi interesnih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aleksander Sadikov
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 24. september 2013 Podpis avtorja:

Zahvalil bi se mentorju viš. pred. dr. Aleksanderu Sadikovu za ves čas in podporo, ki ga je namenil za čas izdelave diplomske naloge.

Prav tako bi se zahvali vsej ekipi podjetja Monolit, ki mi je od vsega začetka svetovala in pomagala pri postavitvi sistema. Brez nje ta diplomska naloga nebi bila možna. Hvala.

Kazalo

Povzetek

Namen diplomske naloge z naslovom Priporočilni sistem za nepremičnine na podlagi interesnih točk je bilo zgraditi prototipni priporočilni sistem, ki uporabniku svetuje pri nakupu nepremičnine glede na bližino interesnih točk POI(*Point of Interest*). Pri tem so bili podatki o POIih izposojeni iz strani podjetja Monolit d.o.o., podatki o nepremičninah pa pridobljeni iz spletnega mesta neprimicnine.net. Podatki se hranijo v podatkovni bazi PostGreSQL z razširitvijo PostGIS, ki jo potrebujemo za delo z geografskimi podatki. Podatki so bili nato servirani s pomočjo spletnih servisov tehnologije REST, preko katerih poteka komunikacija odjemalec-strežnik. Nato je bil za potrebe odjemalca razvit spletni odjemalec, ki temelji na standardnih tehnologijah HTML5, JQuery in CSS3. V spletnem odjemalcu lahko uporabnik filtrira nepremične, določi tipe interesnih točk, najde točno določene in nastavi uteži posameznim parametrom s pomočjo katerih se nato določi končna ocena vsake nepremičnine. Spletni odjemalec nam nato najbolj ocenjene rezultate predstavi in poda enostavno analizo rezultata. Najdene nepremičnine predstavi na karti skupaj z interesnimi točkami.

Ključne besede

priporočilni sistem, GIS, geografski informacijski sistem, spletni servisi, spletni odjemalec, nepremičnine, interesne točke

Abstract

The goal of the thesis was to prototype a recommendation system which would advise the user when purchasing immovable property based on vicinity of points of interest. The data used for determining POI's was provided by Monolit d.o.o. while the data for immovable property were obtained from the website nepremicnine.net. The data is then stored in an PostGIS extended PostGreSQL database, which is used for geographical work. The data is then presented using web services with REST technology in a client-server architecture. For the purpose of client we developed a simple web client based on HTML5, JQuery and CSS3 standards. The user can then filter real estate, set the desired type of points of interest, even set weighted parameters determining the final score for each real estate. The system then presents the best scoring real estates on a map together with the points of interest.

Keywords

recommendation system, GIS, geographic information system, web services, web client, real estate, points of interest, POI

Poglavje 1

Uvod

Cilj diplomske naloge je bil razviti prototipni priporočilni sistem, ki bo pomagal pri odločitvi nakupa nepremičnine. Pri tem so glavna oporna točka celotnega sistema POI(*Point of Interest*) oziroma interesne točke. Sistem namreč glede na izbrane tipe interesnih točk, ki jih želimo imeti poleg svoje bodoče nepremičnine in glede njihovih uteži, ki jih uporabnik nastavi, predlaga deset najbolje ocenjenih nepremičnin, ki naj bi zanimale uporabnika.

Ideja izvira iz podobnega sistema, ko smo zasledili podobno temo, kjer je cilj predlagati nepremičnino glede na bližino javnih povezav do službe. Odločili smo se, da sistem nadgradimo in razvijemo do te mere, da bo znal uporabiti vse interesne točke in ne le javne povezave.

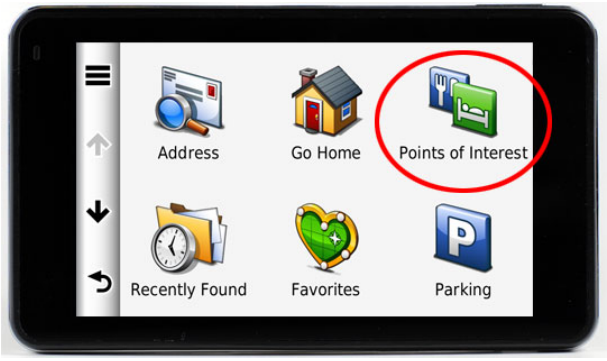
Glavni motiv dela je bil se spoznati s sistemi GIS (geografski informacijski sistem) in razviti delujoči prototip, ki bi se lahko z nekaj dodatnega dela hitro razširil in dodelal za dejansko produkcijsko okolje. Menimo tudi, da bi izdelek lahko služil kot specializirano orodje za posrednike nepremičnin ali kot enostavno orodje za vsakega uporabnika. Razvoj smo usmerili v enostavnost in preprostost sistema, ki ga bi znal uporabljati vsak, in se zato odločili za razvoj spletnega vmesnika za katerega uporabnik potrebuje le spletni brskalnik. Za potrebe odjemalca smo morali razviti tudi spletne servise, ki nam posredujejo podatke o nepremičninah in interesnih točkah ter skrbijo za posredovanje zahtev in odgovorov med odjemalcem in strežnikom. Za shra-

njevanje podatkov smo določili podatkovno bazo PostgreSQL, saj ima dobro podporo za delo s podatki GIS s svojo razširitvijo PostGIS. Vse podatke o interesnih točkah smo si sposodili pri podjetju Monolit d.o.o., testni vzorec podatkov o nepremičninah pa pridobili s spletnega mesta nepremičnine.net. Pri tem želimo poudariti, da smo se zaradi hitrosti razvoja prototipa in enostavnosti odločili, da uporabimo podatke le za Ljubljano.

Poglavje 2

Interesne točke

Interesne točke so hrbtenica našega sistema, zato jih bomo na kratko opisali. Predstavljajo lokacije, ki naj bi jih uporabnik ocenil kot koristne ali zanimive. Za opis interense točke sta potrebni zemljepisna širina in zemljepisna dolžina. Običajno so interesne točke opremljene še z imenom ali opisom, lahko pa vsebujejo tudi podatke o višini, telefonski številki, ulici, odpiralnem času itn. Za navigacijske naprave je tudi običajno, da je vsak tip interesne točke predstavljen s svojo ikono. Največkrat se izraz interesne točke uporablja v kartografiji, še posebej znotraj sistemov GIS in navigacijskih naprav (prikazano na Sliki 2.1), kjer se rada uporablja angleška kratica POI (*point of interest*).



Slika 2.1: Primer interesnih točke v navigacijski napravi

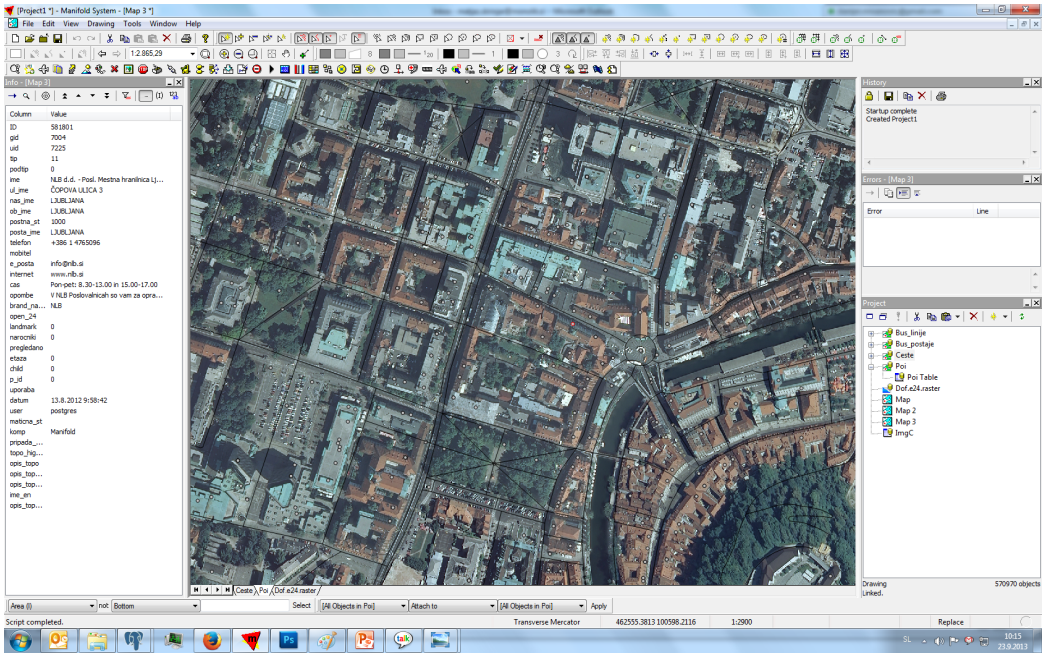
Vse interesne točke so pridobljene od podjetja Monolit d.o.o. in smo si jih izposodil za potrebe razvoja prototipa. O način zbiranja podatkov smo se pozanimali pri geodetu Matjažu Škrinjarju, zaposlenem v Monolitu [3].

Večina interesnih točk je bila zbranih s pomočjo internetnih virov, predvsem javnih virov, ki jih podjetja objavijo na svojih spletnih straneh ali v spletnih zbirkah interesnih točk. Naj podamo nekaj primerov tako dobljenih podatkov: bencinski servisi, bankomati, banke, trgovine itn. Podatki o avtokampih, gostilnah, lokalih itd. so največkrat pridobljeni na rumenih straneh ali preko ponudnikov podatkov o podjetjih, kjer glede na ulico in hišno številko ter s pomočjo evidence hišnih številk, ki je že umeščena v prostor, določimo lokacijo posamezne interesne točke. Seveda je večkrat potreben tudi kakšen "ročni"popravek, kjer si pomagamo s slojem digitalnih ortofoto posnetkov (DOF). Tako je največkrat treba popraviti lokacijo interesne točke, ker je ta prijavljena na drugem naslovu kot kjer se dejansko nahaja, ali pa je treba lokacijo zaradi praktičnosti spremeniti na primer z dejanske lokacije naslova na vhod (primer je avtokamp, pri katerem sta obe napaki tudi najbolj pogosti). Pri naravnih znamenitostih pa je včasih treba dobiti lokacijo s fizične karte, kjer si zopet pomagamo s slojem DOF, da točko čim-bolj natančno umestimo v prostor. Na sliki 2.2 je primer prikaza orodja, kako geodeti popravijo interesno točko.

Monolit je glavni ponudnik za cestno kartografijo in podatke o interesnih točkah v Sloveniji. Tako lahko njihove podatke o interesnih točkah najdemo pri vseh večjih ponudnikih navigacijskih naprav kot so Garmin, Mio itd., prav tako pa jih lahko najdemo v Googlovih zemljevidih in pri enem izmed največjih ponudnikov kartografije na svetu Navtequ, ki je po novem del Nokie.

Našteli smo še nekaj najbolj priljubljenih tipov interesnih točk, ki se v našem sistemu uporabljajo:

- bankomat
- banka



Slika 2.2: Primer orodja za delo z GIS sistemi.

- pošta
- lekarna
- zdravstveni dom
- fitnes
- bazen
- osnovna šola
- vrtec
- bar
- gostilna - splošno

Poglavje 3

Podatki, podatkovna baza PostgreSQL in razširitvi

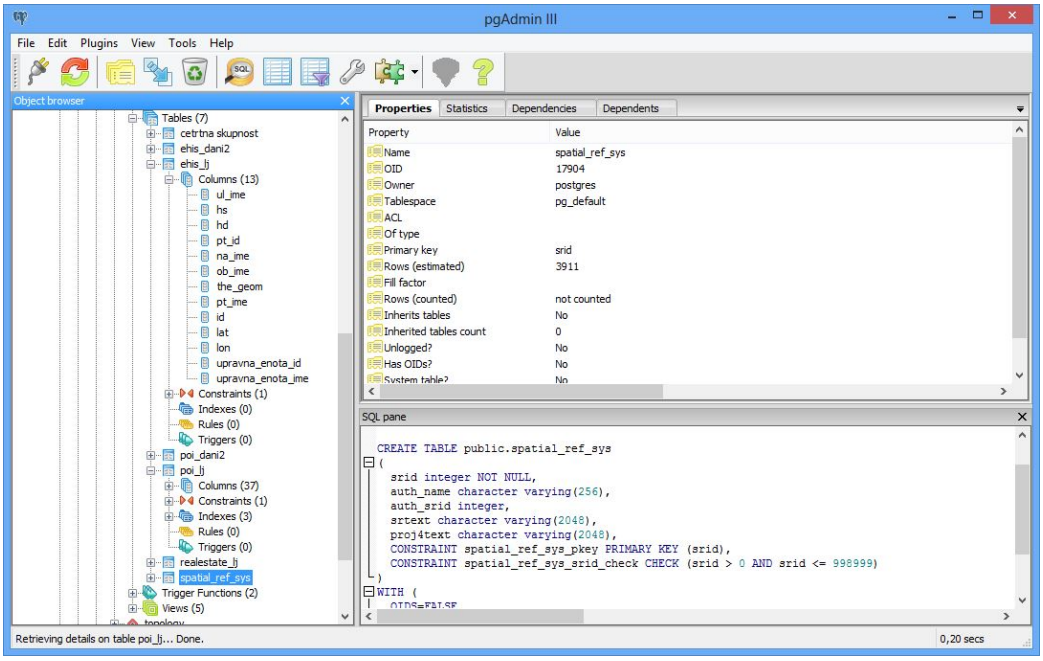
Za potrebe podatkovne baze smo se odločili za podatkovno bazo PostgreSQL [5]. Prvi razlog za to odločitev je, da je baza odprtokodna in kot taka nima nobenih omejitev proizvajalca. Drugi razlog pa je, da ima zelo dobro dokumentirano in podprto razširitev za delo s sistemi GIS, PostGIS. To odločitev so predlagali tudi strokovnjaki s področja GIS iz podjetja Monolit [3] in tako se po drugih alternativah nismo ozirali, saj je celotna baza že večkrat izkazala v produkcijskem okolju.

Pri namestitvi podatkovne baze ni bilo težav. Poleg tega smo namestili tudi koristno orodje za delo z bazo pgAdmin 3 (prikazano na sliki 3.1), s katerim smo med drugim testirali in izvajali stavke SQL in ga uporabili za pregled podatkov in kreiranje indeksov.

V bazi smo nato namestili naslednje dve razširitvi [4, 5]:

- PostGIS – je razširitev, ki doda funkcije za delo s prostorskimi podatki in nam omogoča pisanje prostorskih poizvedb v SQLu.

- pg_trgm – je razširitev, ki ponudi dostop do funkciji in operatorjev za določneje podobnosti besedila glede na ustreznost trigramov besed ter operacijske razrede za indeks (*indexs operator classes*), ki omogočajo



Slika 3.1: Orodje pgAdmina 3

hitro iskanje podobnih nizev. Zakaj smo to razširitev v praksi uporabili, razlagamo v poglavju 3.4.

Za vsa nadaljnja dela smo nato uporabili naslednje štiri tabele, ki smo jih uvozili v podatkovno bazo:

- `cetrna skupnost` – v tej tabeli so shranjeni in opisani poligoni mestnih četrti po Sloveniji.

- `ehis_lj` – v tej tabeli so shranjene in opisane vse ulice s hišnimi številkami za Ljubljano. Dodan je tudi podatek o zemljepisni širini in dolžini točke, ki označuje hišni naslov.

- `poi_lj` – v tej tabeli so shranjene in opisane vse interesne točke za Ljubljano. Dodan je tudi podatek o zemljepisni širini in dolžini točke, ki označuje lokacijo interesne točke.

- `realestate_lj` – v tej tabeli so shranjeni testni podatki in opisi vseh stanovanjskih nepremičnin za Ljubljano, ki smo jih pridobili iz spletnega mesta nepremicnine.net V naslednjih poglavjih bomo opisali strukturo vsake tabele, indekse, ki smo jih ustvarili v vsaki tabeli posebej, in njihov namen.

3.1 Vrste indeksov

Trije najpogostejši tipi indeksov, ki se jih uporablja znotraj podatkovne baze PostgreSQL [5]:

B-Tree – privzeti tip indeksa, saj se dobro obnese pri najbolj pogostih uporabah indeksa. Dobro se torej pri operacijah enakostih (<, <=, =, >=, >) ali od-do poizvedbah kot so funkcija `BETWEEN` in `IN`. Uporablja se tudi pri uporabi funkcije `LIKE`, a le v primeru, da je konstanta na začetku (npr. `LIKE('foo%')`)

GiST – ni samo tip indeksa, ampak je cela infrastruktura, znotraj katere se lahko uporabi mnogo različnih strategij indeksiranja. Glede na operacijski razred indeksa se uporabi ustrezna indeksna strategija. Ta tip indeksa je tudi prva izbira, če bi želeli narediti indeks nad geometrijo, saj se zelo dobro obnese pri problemih najbližjega soseda.

GIN – so inventirani indeksi, ki se dobro obnesejo pri vrednostih, ki vsebujejo več kot en ključ. V osnovi so podobni indeksu GiST in lahko uporabijo različne strategije glede na operacijski razred.

Obstaja tudi "pravilo palca", da so indeksi GIN hitrejši od indeksa GiST, a potrebujejo veliko več časa, da kreirajo indeks na velikih tabelah, kot indeksi tipa GiST. Prav tako indeksi GIN potrebujejo več časa, da izvedejo posodobitev indeksa, kar zna predstavljati težavo, če ga uporabimo nad tabelami, kjer se vrši veliko `INSERT` in `UPDATE` poizvedb.

V tem delu bi omenili še zelo koristen ukaz SQL EXPLAIN, s katerim lahko preverimo za dano poizvedbo SQL ali se je indeks uporabil ali ne.

Prilagam še primer ukaza in rezultata, iz katerega (prikazano v kodi 3.1 in kodi 3.2) je vidna uporaba indeksa.

Koda 3.1: Primer ukaza SQL EXPLAIN

```
1 EXPLAIN SELECT * FROM poi_lj WHERE LOWER(ime) LIKE('%fri%')
```

Koda 3.2: Primer rezultat ukaza SQL EXPLAIN ko je indeks uporabljen

```
1 Bitmap Heap Scan on poi_lj (cost=12.00..16.02 rows=1 width=457)
2   Recheck Cond: (lower((ime)::text) ~~ '%fri%'::text)
3   -> Bitmap Index Scan on poi_lj_ime_gin_trgm_idx (cost
       =0.00..12.00 rows=1 width=0)
4         Index Cond: (lower((ime)::text) ~~ '%fri%'::text)
```

V našem prototipnem sistemu uporaba indeksa ni bistveno pohitrila delovanja, saj je celotna baza podatkov razmeroma majhna, kar pomeni, da se baza pri stavkih SQL, v kombinaciji z ukazom `LIMIT` raje odločila za sekvenčno iskanje kot pa uporabo indeksa. Indekse smo vseeno opisali in uporabili zaradi morebitnih razširitev ter primere, ko bi se sistem uporabil na večjih podatkovnih bazah.

Poudarili bi še, da bi bilo potrebno za večja območja implementirati omejitev radija na glavni poizvedbi, s katero zajemamo interesne točke. V tem primeru bi morali implementirati še funkcijo v kodi 3.3. Da bi funkcija hitro delovala, bi nujno potrebovali indekse nad stolpci, kjer se hrani geometrija. Ker smo se omejili na Ljubljano te rešitve nismo implementirali. Za potrebe kreiranja indeksa nad geometrijo si lahko preberete več na uradnem spletnem mestu razširitve PostGIS.

3.2 Tabela četrtnih skupnosti

Tabelo četrtnih skupnosti smo uporabil zgolj za pred obdelavo tabele nepremičnin. Kako in zakaj smo tabelo uporabili, je opisano v poglavju 4. Na

Koda 3.3: Funkcija za omejitev radija

```
1 ST_DWithin(
2   st_transform(st_setsrid(the_geom, 4326), 3787),
3     st_transform(ST_GeomFromText('POINT(14.5844871350114
          46.0723657448294)', 4326), 3787),
4     3000 --radij v metrih
5 )
```

kratko bomo povzeli strukturo tabele in njene stolpce (opisano v tabeli 3.1). Opisali bomo samo tiste stolpce, ki smo jih uporabili.

Tabela 3.1: Opis tabele četrtnih skupnosti

Ime stolpca	**Tip stolpca**	**Opis**
d42_ime	varchar(30)	Ime mestne četrti
gid	integer	Identifikacija in primarni ključ
geom	geometry(MultiPolygon)	Geometrija, ki opisuje poligon mestne četrti
upravna_enota_id	integer	Id, ki nam pove h kateri upravni enoti sodi mestna četrt
upravna_enota_ime	character varying(30)	Ime upravne enote h kateri sodi mestna četrt

Stolpca upravna_enota_id in upravna_enota_ime smo dodali sami in ročno vpisali ustrezno upravno enoto, h kateri je mestna četrt pripadala. Pri tem smo se omejili le na mestne četrti Ljubljane.

3.3 Tabela hišnih naslovov

Tabelo hišnih naslovov smo prav tako uporabili le za pred-obdelavo podatkov nepremičnin. Na kratko bomo še povzeli strukturo tabele in njene stolpce

(opisano v tabeli 3.2). Opisali bomo samo tiste stolpce, ki smo jih uporabili.

Tabela 3.2: Opis tabele hišnih naslovov

Ime stolpca	**Tip stolpca**	**Opis**
ul_ime	character varying(30)	Ime ulice
hs	integer	Hišna številka
the_geom	geometry	Geometrija, ki opisuje točko lokacije hišne številke.
id	bigserial	Identifikacija in primarni ključ
lat	double precision	Zemljepisna širina
lon	double precision	Zemljepisna dolžina
upravna_enota_id	integer	Id, ki nam pove, h kateri upravni enoti sodi hišni naslov.
upravna_enota_ime	character varying(30)	Ime upravne enote, h kateri sodi hišni naslov.

Stolpec id smo dodali, saj uvožena tabela ni imela določenega primarnega ključa, ki ga potrebujemo, da lahko tabelo kot entitetni objekt uvozimo v naš program. Prav tako smo dodali tudi stolpca lat in lon. Programsko opisati, kaj predstavlja geometrija stolpca the_geom, bi bilo namreč zahtevno. Namreč po specifikaciji JPA moramo določen stolpec predstaviti kot nek objekt (več o tem v poglavju 5). Najlažje je če stolpce predstavimo v enem od primitivnih tipov (`Integer`, `String`, `Double` itn.). Ker je geometrija zapisana binarno in lahko predstavlja več vrst geometrije (točko, poligon, 3D poligon itn.), bi bilo napisati konverzijo v primeren primitiv zelo zahtevno, zato smo preprosto zapisali geometrijo s parom zemljepisna dolžina - zemljepisna širina. Ta rešitev deluje le za geometrijo, ki opisuje točke, a ker potrebe po opisu geometrije drugih tipov nismo imeli, je bila ta rešitev povsem primerna.

Vsako hišno številko smo naknadno opremili še s podatkom o upravni enoti. Podobno kot pri mestni četrti je to del, ki smo ga dodali, da smo

lahko obdelali tabelo nepremičnin (več o tem v poglavju 4).

3.4 Tabela interesnih točk

Tabela interesnih točk je najbolj uporabljena tabela v našem sistemu in je opremljena z naslednjimi stolpci (opisano v tabeli 3.3). Opisali bomo samo tiste stolpce, ki smo jih dejansko uporabili.

Tabela 3.3: Opis tabele interesnih točk

Ime stolpca	**Tip stolpca**	**Opis**
gid	Integer	Identifikacija in primarni ključ
ime	character varying(70)	Ime interesne točke
ul_ime	character varying(70)	Hišni naslov interesne točke
the_geom	geometry	Geometrija, ki opisuje točko lokacije interesne točke.
lat	double precision	Zemljepisna širina
lon	double precision	Zemljepisna dolžina
poi_type_id	integer	Id, tipa interesne točke
opis_slo	character varying(50)	Opis ali naziv tipa interesne točke

Tudi tu smo dodali stolpca lat in lon iz enakih razlogov kot v tabeli hišnih naslovov, da smo lahko na enostaven način opisali podatek o lokaciji interesne točke v entitetnem objektu, ki je opisoval tabelo oziroma vrstico v tabeli interesnih točk (več o tem v poglavju 5).

V tej tabeli smo ustvarili tudi naslednje tri indekse, da smo pohitrili naše poizvedbe:

- poi_lj_gid_idx – indeks, narejen nad stolpcem gid. Uporabili smo indeks tipa B-tree, saj smo izvajali poizvedbe z operatorji enakosti (npr. `WHERE gid IN (1, 3, 121)`)
- poi_lj_poi_type_id_idx – indeks, narejen nad stolpcem poi_type_id. Uporabili smo indeks tipa B-tree, saj smo izvajali poizvedbe z operatorji

enakosti (npr. `WHERE poi_type_id IN (1, 3, 121)`)

- poi_lj_ime_gin_trgm_idx – indeks tipa GIN, narejen nad stolpcem ime. Pri tem indeksu smo uporabili razširitev pg_trgm, saj so v njej definirani operacijski razredi, ki znajo definirati ustrezno indeksno strategijo za problema tipa `WHERE ime LIKE('%foo%')`. Gre torej za primer, kjer je lahko za iskalno besedo in pred njo poljuben niz znakov.

V kodi 3.4 so napisani vsi tri stavki SQL potrebni za kreiranje zgoraj navedenih indeksov.

Koda 3.4: Ukazi za kreiranje indeksov na tabeli interesnih točk

```
1 CREATE INDEX poi_lj_ime_gin_trgm_idx ON poi_lj USING gin (LOWER(
    ime) gin_trgm_ops);
2 CREATE INDEX ON public.poi_lj (gid);
3 CREATE INDEX ON public.poi_lj (poi_type_id);
```

Pri indeksu za stolpec gid in poi_type_id ni treba navesti imena in tipa indeksa. PostgreSQL namreč privzeto uporabi B-tree indeks in generira ustrezno ime.

3.5 Tabela nepremičnin

Tabela nepremičnin je namenjena iskanju ustreznih nepremični in je sestavljena iz naslednjih stolpcev (opisano v tabeli 3.4).

Pri nepremičninah smo se omejili le na stanovanja znotraj Ljubljane, namenjena prodaji, vendar smo zgradili strukturo, ki jo je mogoče hitro prenesti tudi na ostale tipe nepremičnin, regije, lokacije, načine prodaje. Ustvarili smo tudi indeks tipa B-tree za vse stolpce, preko katerih filtriramo nepremičnine. V kodi 3.5 je napisan ukaz, potreben za kreiranje indeksa.

Tabela 3.4: Opis tabele nepremičnin

Ime stolpca	**Tip stolpca**	**Opis**
id	bigint	Identifikacija in primarni ključ
title	character varying	Naziv nepremičnine
location	character varying	Lokacija nepremičnine oziroma upravna enote
location_id	integer	Id lokacije
region	character varying	Regija nepremičnine
region_id	integer	Id regije
selltype	character varying	Način prodaje (prodaja, najem itd.)
selltype_id	integer	Id načina prodaje
size_m2	double precision	Kvadratura nepremičnine
price	bigint	Cena nepremičnine v centih
type	character varying	Tip nepremičnine (stanovanje, hiša itd.)
type_id	integer	Id tipa
subtype	character varying	Podtip nepremičnine (eno-sobno, garsonjera itd.)
subtype_id	integer	Id podtipa
build_year	integer	Leto, v katerem je bila nepremičnina zgrajena.
floor	character varying	Nadstropje nepremičnine
floor_max	character varying	Maksimalno nadstropje stavbe v kateri je nepremičnina.
ulica	character varying	Ulica nepremičnine
hisna_st	integer	Hišna številka nepremičnine
lat	double precision	Zemljepisna širina
lon	double precision	Zemljepisna dolžina

Koda 3.5: Ukaz za kreiranje indeksa na tabeli nepremičnin

```
1 CREATE INDEX ON public.realestate_lj (location_id, region_id,
    selltype_id, size_m2, type_id, price, subtype_id, build_year)
    ;
```

Poglavje 4

Obdelava nepremičnin

V produkcijskem okolju, bi bila najbolj primerna rešitev za dostop do podatkov o nepremičninah, če bi ponudniki nepremičnin ponudili specifikacijo in dostop do spletnih servisov. Ker do spletnih servisov dostopa nismo imeli, je bilo potrebno najti način, kako se dokopati do testnih podatkov. Odločili smo se, da bomo obdelali spletno mesto nepremicnine.net in si prenesli testni vzorec podatkov o vseh stanovanjih, ki se prodajajo v Ljubljani. Za ta namen smo uporabili razvojno orodje Eclipse IDE, ki je namenjeno razvoju javanskih aplikacij. Dodali smo še modul za podporo Mavenu. Maven omogoča dodajanje knjižnic in dokumentacije iz enega skupnega repozitorija, kreira pa tudi ustrezno strukturo glede na tip projekta, ki se ga bomo lotili.

Za obdelavo spletne strani smo se odločili za uporabo knjižnice jsoup. Če povzamem opis knjižnice z njihovega spletnega mesta [6], je jsoup javanska knjižnica, za delo z HTMLjem. Ponuja nam zelo priročen API za izvoz in upravljanje podatkov HTML ter pri tem uporablja najboljše metode znanih tehnologij, kot so DOM, CSS, jQuery. Za uvoz knjižnice je treba le dodati vrstice, napisane v kodi 4.1, v pom.xml datoteko znotraj `<dependencies>` značk. Maven nato poskrbi, da se ustrezna knjižnica prenese v naš projekt.

Podatke smo shranili v obliko datoteke csv, zato smo potrebovali še enostavno knjižnico, ki omogoča pisanje datotek CSV. Odločili smo se za knjižnico opencsv in jo dodali v naš projekt s kodo 4.2, podobno kot knjižnico

Koda 4.1: Dodajanje knjižnice jsoup v naš Maven podprt projekt

```
<dependency>
  <!-- jsoup HTML parser library @ http://jsoup.org/ -->
  <groupId>org.jsoup</groupId>
  <artifactId>jsoup</artifactId>
  <version>1.7.2</version>
</dependency>
```

jsoup.

Koda 4.2: Dodajanje knjižnice opencsv v naš Maven podprt projekt

```
<dependency>
   <groupId>net.sf.opencsv</groupId>
   <artifactId>opencsv</artifactId>
   <version>2.3</version>
</dependency>
```

4.1 Opis programa

Preden smo se lotili programa, smo morali analizirati kako spletno mesto nepremicnine.net sestavlja spletni naslov glede na izbrane prametre, kot so tip sobe in lokacijo. S pregledom HTMLja smo ugotovili tudi, v katerih poljih oziroma elementih so shranjeni posamezni podatki, da smo jih lahko kasneje ustrezno izluščili iz dokumenta HTML.

Delovanje programa najlažje povzamem, če ga opišem po korakih:

1. Nastavimo začetne parametre kot so začetni url, šifrante lokacij in tipov sob.

2. Začnemo s for zanko preko šifranta lokaciji in nato skočimo v ugnezdeno for zanko preko šifranta tipov sob. Glede na položaj obeh for zank v

šifrantih sestavimo ustrezni spletni naslov, ki ga moramo po novem obiskati.

3. Za dani naslov pridobimo dokument HTML in ga obdelamo.

4. Če opazimo, da ima dobljeni dokument HTML vsebovano navigacijo za premikanje po straneh, ker vsi rezultati niso mogli biti prikazani na eni strani, iz nje izluščimo zadnjo stran in se sprehodimo še preko preostalih strani ter pri tem vsako še ustrezno obdelamo.

5. Pri obdelavi spletne strani se sprehodimo preko vsakega elementa div znotraj strani, ki opisuje stanovanje, ga obdelamo in shranimo stanovanje v globalno tabelo stanovanj.

6. Na koncu, ko smo se sprehodili čez vse možne kombinacije šifranta lokacije in tipov sob nam preostane samo še, da shranimo vsa stanovanja iz globalne tabele stanovanj v datoteko CSV s pomočjo knjižnice opencsv.

Obdelava samega HTMLja je bila zelo preprosta z uporabo knjižnice jsoup. Najprej smo pridobili element div v katerem je bila vsebina, in se nato sprehodili z zanko while preko vseh notranjih elementov div, ki so opisovali stanovanja, ter pri tem izluščili ustrezne podatke. V kodi 4.3 je primer, kako smo izluščili ceno in velikost.

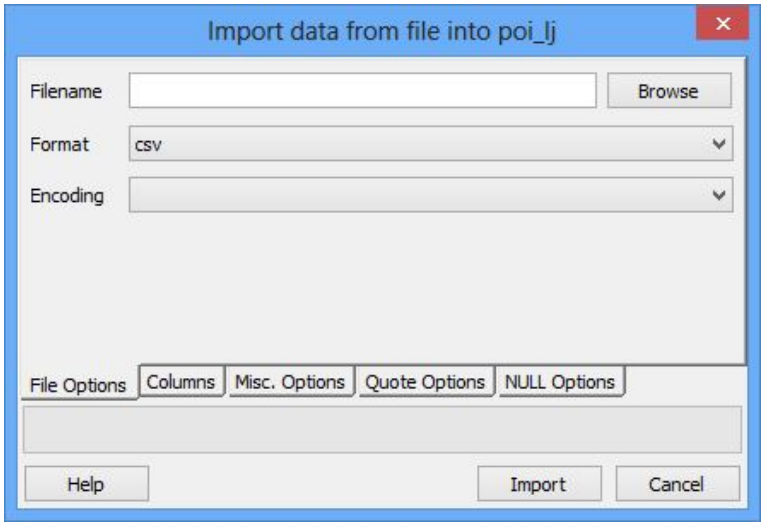
4.2 Obdelava podatkov nepremičnin

Nepremičnine smo imeli shranjene v formatu CSV in jih je bilo treba uvoziti v podatkovno bazo. To smo naredili z orodjem v pgAdmin 3, ki omogoča uvoz podatkov CSV v tabelo (prikazno na sliki 4.1). Tu je pogoj, da je vrstni red stolpcev enak, kot so podatki vpisani v datoteki CSV. Določiti je treba še ustrezno kodiranje, ločilni znak in narekovaje, če so podatki obdani z njimi.

Podatke smo zdaj imeli v podatkovni bazi, vendar smo naleteli na problem, da podatki o nepremičninah niso bili opremljeni z ulico ali geometrijo, ki bi lokacijsko opredelila nepremičnino. Najbolj natančen podatek o tem,

Koda 4.3: Koda za pridobitev podatka o velikosti in ceni nepremičnine iz dokumenta HTML

```
1  String size = div.select(".velikost").first().text();
2  size = size.substring(0, size.length() - 3).replace(',', '.');
3  flat.setSize(Float.parseFloat(size));
4
5  String price = div.select(".cena").first().ownText();
6  if (price.contains("m2")) {
7    price = price.substring(0, price.length() - 5).replaceAll("
       [\\.,]", "");
8    flat.setPrice((long)((float)Long.parseLong(price) * flat.
       getSize()));
9  }
10 else {
11   price = price.substring(0, price.length() - 2).replaceAll("
       [\\.,]", "");
12   flat.setPrice(Long.parseLong(price));
13 }
```



Slika 4.1: Orodja za uvoz podatkov iz dokumenta v formatu CSV

kje je lokacija, je bil podatek o upravni enoti. Odločili smo se za naslednjo rešitev:

1. Najprej smo v tabeli mestnih četrti vsako mestno četrt znotraj Ljubljane opremili s podatkom, h kateri upravni enoti sodi.

2. Nato smo pognali poizvedbo `UPDATE`, kjer smo vsako ulico v tabeli hišnih naslovov opremili s podatkom o upravni enoti glede na to, katera mestna četrt jo prekriva. Podano v kodi 4.4.

3. Nato smo spisali enostaven programček, ki je vrnil vsa stanovanja za posamezno upravno enoto.

4. Glede na število stanovanj v posamezni upravni enoti smo nato izvedli poizvedbo nad tabelo hišnih naslovov, ki je vrnila naključne hišne naslove iz ustrezne upravne enote.

5. Na koncu smo opremili vsako stanovanje z naslovom ter zemljepisno širino in dolžino iz iste upravne enote, kot je samo stanovanje.

Z zgoraj razloženim postopkom smo tako dobili kakovosten vzorec, kjer so nepremičnine iz določene upravne enakomerno raztresene po vsej upravni enoti glede na gostoto hišnih naslovov.

Koda 4.4: Poizvedba UPDATE za nastavitev upravne enote hišnih naslovov na podlagi prekrivanja z mestno četrtjo

```
1 UPDATE public.ehis_lj
2 SET upravna_enota_id = public."cetrtna skupnost".upravna_enota_id
    , upravna_enota_ime = public."cetrtna skupnost".
    upravna_enota_ime
3 FROM public."cetrtna skupnost"
4 WHERE ST_Within(public.ehis_lj.the_geom, public."cetrtna skupnost
    ".geom)
```

Poglavje 5

Strežnik, spletni servisi

Razvoj spletnih servisov smo se lotili v razvojnem orodju Netbeans. Netbeans so primarno namenjeni razvoju Java aplikaciji, hkrati vsebujejo tudi zelo dobro podporo za razvoj spletnih servisov in aplikacij. Prav tako imajo že ob namestitvi možnost namestitve tudi spletnega strežnika Apache Tomcat, ki pride že integriran znotraj Netbeansov in tako olajša testiranje spletnih servisov.

Pri razvoju spletnih servisov smo se odločili za uporabo naslednji tehnologij.

- JPA (Java Presistence API)
- Spletne servise tipa REST (Representational State Transfer)

JPA definira, kako se upravlja s podatki relacijskih baz v Java aplikacijah, medtem ko REST definira, kako naj se zahteve odjemalec strežnik prenašajo in obravnavajo. Opisa smo povzeli po [2, 8].

V naslednjih korakih bomo predstavili kako smo vzpostavili projekt.

1. Najprej smo kreirali spletni projekt, kjer smo izbrali, da bomo uporabili standard ee6 in strežnik Apache Tomcat
2. Nato je bilo treba vzpostaviti povezavo z našo podatkovno bazo PostgreSQL. Tu smo šli skozi enostaven čarovnik za vzpostavitev pove-

zave. Pri vzpostavitvi povezave je bilo treba prenesti še gonilnike, ki jih Java potrebuje za delo s podatkovno bazo tipa PostgreSQL. Gonilniki so zapakirani v datoteki tipa jar, ki jih je treba nato uvoziti tudi znotraj našega projekta.

3. Potem ko smo imeli povezavo s podatkovno bazo, smo s pomočjo čarovnika za uvoz entitete (prikazan na sliki 5.1) iz podatkovne baze uvozili entiteto za tabele interesnih točk, nepremičnin in hišnih naslovov.

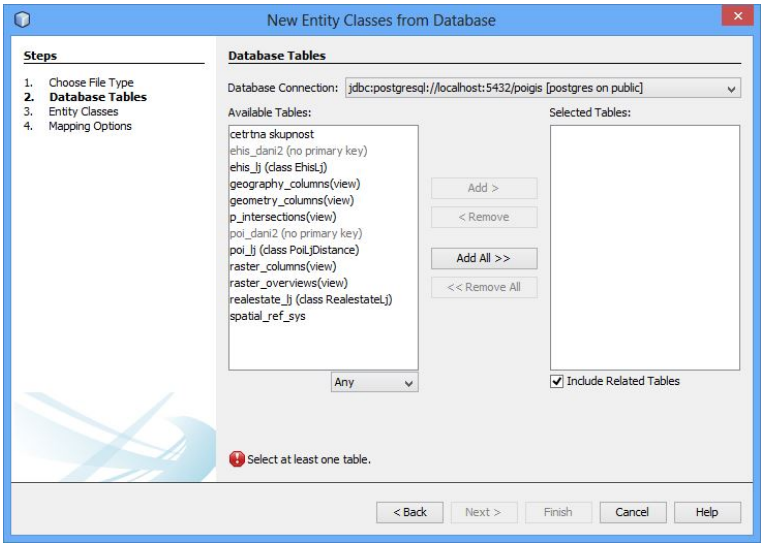
4. Nato smo s čarovnikom ustvarili enostavne kontrolerje za naše entitete. Kontrolerji so razredi, kjer dodamo kodo, ki sestavlja poizvedbe za posamezno entiteto nad bazo. Privzeto je v kontrolerjih že napisanih nekaj preprostih poizvedb.

5. Na koncu dodamo še razred, v katerega pišemo zahteve REST, torej spletne servise. Tu si zopet pomagamo s čarovnikom (prikazano na sliki 5.2), kjer označimo še, da želimo uporabiti knjižnico Jersey. Čarovnik nam doda še vse ustrezne nastavitvene datoteke, ki jih potrebujemo za delovanje spletnih servisov.

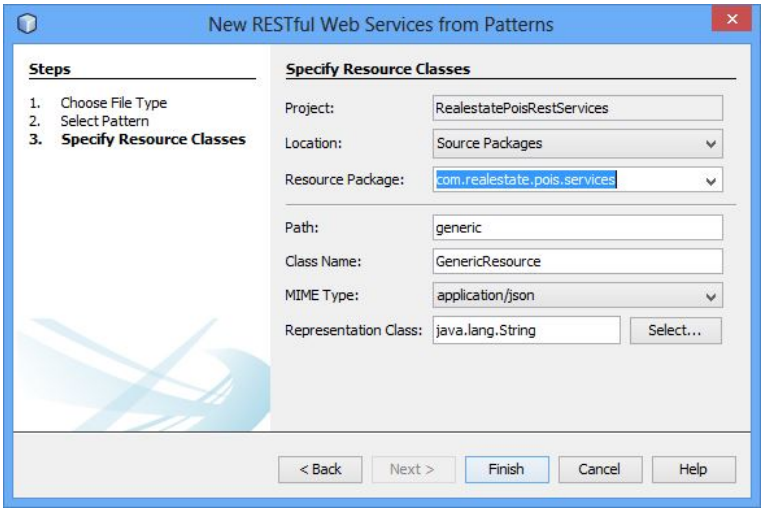
5.1 Entitete in poizvedbe nad njimi

Entiteta v našem primeru predstavlja enostaven javanski objekt, ki opisuje zapis v tabeli [2]. Z entiteto lahko torej predstavimo eno vrstico podatkovne baze. Primer entitete imamo podan v kodi 5.1. Vidimo, da gre v osnovi za razred z getterji in setterji ali za POJO (*Plain Old Java Object*), kot se poimenuje tak razred v javanskih krogih.
V našem programu imamo tako naslednje entitetne razrede:

- EhisLj – entitetni razred tabele hišnih naslovov
- PoiLj – entitetni razred tabele interesnih točk



Slika 5.1: Čarovnik za uvoz entitete



Slika 5.2: Čarovnik za kreiranje spletnih servisov

Koda 5.1: Primer entitete

```
1  @Entity
2  @Table(name = "ehis_lj", catalog = "poigis", schema = "public")
3  @XmlRootElement
4  public class EhisLj implements Serializable {
5     private static final long serialVersionUID = 1L;
6     @Column(name = "ul_ime")
7     private String ulIme;
8     @Column(name = "hs")
9     private Integer hs;
10    @Column(name = "hd")
11    private String hd;
12 //itd.
13
14    public EhisLj() {
15    }
16
17    public String getUlIme() {
18       return ulIme;
19    }
20
21    public void setUlIme(String ulIme) {
22       this.ulIme = ulIme;
23    }
24
25    public Integer getHs() {
26       return hs;
27    }
28
29    public void setHs(Integer hs) {
30       this.hs = hs;
31    }
32 //itd.
33 }
```

- PoiLjDistance – entitetni razred interesnih točk z dodanim stolpcem razdalje
- RealestateLj – entitetni razred nepremičnine.

Poizvedbe nad entitetami izvajamo v ustreznih kontrolnih razredih. Specifikacija JPA [2] ponuja več možnosti, kako napisati poizvedbo.

1. Napišemo poizvedbo s pomočjo graditelja kriterijev. Tu uporabimo metode, podobne SQLovim ukazom. Priporočen način za sestavljanje dinamičnih poizvedb.
2. Napišemo poizvedbo v načinu JPQL. Zelo podobno je kot pisanje stavkov SQL, le da ima jezik nekaj posebnosti.
3. Napišemo poizvedbo v izvirnem načinu. V primeru ko poizvedba vsebuje funkcije, ki se lahko izvajajo samo nad določeno podatkovno bazo. V našem primeru moramo torej uporabiti ta način, ko izvajamo poizvedbe SQL, ki uporabljajo funkcije razširitve PostGIS.

V kodi 5.2 je primer za vsakega odnačinov. Največja prednost prvih dveh načinov je, da so poizvedbe kompatibilne z vsemi podatkovnimi bazami in sta načina pisanja poizvedb priporočena, medtem ko je pri tretjem odvisno od dejanskih imen in funkciji, ki jih uporabljamo.

Koda 5.2: Primeri načinov pisanja poizvedb

```
1 // Primer graditelja kriterijev
2 public Integer getRealestateLjBuildYearMax() {
3     EntityManager em = getEntityManager();
4     try {
5         CriteriaBuilder cb = em.getCriteriaBuilder();
6         CriteriaQuery<Integer> cq = cb.createQuery(Integer.class)
              ;
7         Root<RealestateLj> rt = cq.from(RealestateLj.class);
8         cq.select(cb.max(rt.get(RealestateLj_.buildYear)));
9         TypedQuery<Integer> q = em.createQuery(cq);
```

```
10          Integer result = q.getSingleResult();
11          return result;
12      } finally {
13          em.close();
14      }
15  }
16  // Primer JPQL
17  public Integer getRealestateLjBuildYearMax() {
18      EntityManager em = getEntityManager();
19      try {
20          StringQuery sq = "SELECT MAX(re.buildYear) FROM
                RealestateLj re";
21          TypedQuery<Integer> q = em.createQuery(sq, Integer.class)
                ;
22          Integer result = q.getSingleResult();
23          return result;
24      } finally {
25          em.close();
26      }
27  }
28  // Primer izvirnega nacina
29  public Integer getRealestateLjBuildYearMax() {
30      EntityManager em = getEntityManager();
31      try {
32          StringQuery sq = "SELECT MAX(build_year) FROM public.
                realestate_lj";
33          Query q = em.createNativeQuery(sq, Integer.class);
34          Integer result = q.getSingleResult();
35          return result;
36      } finally {
37          em.close();
38      }
39  }
```

Sledi opis naše najbolj zahtevne poizvedbe na bazi, ki je hkrati tudi jedro naše aplikacije. Poizvedba nam vrne najbližje interesne točke vsakega tipa skupaj z razdaljo v metrih od dane koordinate nepremičnine. Pri tem se

omejimo na tipe, ki nas oziroma uporabnika zanimajo. Poizvedba si lahko pogledamo v kodi 5.3.

1. Najprej v najbolj ugnezdenem stavku `SELECT` izberem vse interesne točke želenih tipov in jim izračunamo razdaljo od nepremičnine.

2. Nato iz rezultatov stavka `SELECT` v prvi točki vzamemo minimalno razdaljo ter tip. Dobimo tabelo parov tip-razdalja.

3. Ponovimo stavek iz prve točke in ga združimo z rezultati iz druge, tako da dobimo celosten opis ustreznih tipov z minimalno oddaljenostjo za vsako interesno točko. Ker pa je možno, da je minimalna razdalja enaka za več interesnih točk istih tipov, naredimo še poizvedbo, kjer vzamemo interesno točko z najmanjšo identifikacijo. Tako dobimo rezultat tip interesne točke z minimalno razdaljo in minimalno identifikacijo.

4. Na koncu še združimo rezultat iz tretje točke z vsemi interesnimi točkami, kjer se identifikacija ujema. Dobimo torej celosten opis interesne točke skupaj z njeno razdaljo. Zaradi prejšnjih stavkov vemo, da dobimo le interesne točke vsakega iskanega tipa, ki so najbližje nepremičnini.

Koda 5.3: Koda poizvedbe za pridobitev najbližje interesne točke vsakega tipa

```
1     public List<PoiLjDistance> getClosestPOIS (double lon,
         double lat, Integer[] poiType) {
2       EntityManager em = getEntityManager();
3       Cache cache = em.getEntityManagerFactory().getCache();
4       cache.evict(PoiLjDistance.class);
5       try {
6          String customSringQuery = "SELECT\n"
7                + " *\n"
8                + "FROM\n"
9                + " poi_lj\n"
10               + "INNER JOIN\n"
11               + "(\n"
```

```
12                  + " SELECT \n"
13                  + "  POIs_B.poi_type_id,\n"
14                  + "  POIs_B.min_distance,\n"
15                  + "  min(POIs_C.gid) as min_gid\n"
16                  + " FROM\n"
17                  + " (\n"
18                  + "  SELECT\n"
19                  + "   POIs_A.poi_type_id,\n"
20                  + "   min(distance) as min_distance\n"
21                  + "  FROM \n"
22                  + "  (\n"
23                  + "   SELECT\n"
24                  + "    poi_lj.*,\n"
25                  + "    ST_distance(\n"
26                  + "     st_transform(st_setsrid(the_geom, 4326),
                        3787), \n"
27                  + "     st_transform(ST_GeomFromText(?1, 4326),
                        3787)\n"
28                  + "    ) as distance\n"
29                  + "   FROM\n"
30                  + "    poi_lj\n"
31                  + "   WHERE\n"
32                  + "    poi_type_id in (" + JPAUtil.implodeArray("
                        ,", poiType) + ")\n"
33                  + "  ) POIs_A\n"
34                  + "  GROUP BY \n"
35                  + "   poi_type_id\n"
36                  + " ) POIs_B\n"
37                  + " INNER JOIN\n"
38                  + " (\n"
39                  + "  SELECT\n"
40                  + "   poi_lj.*,\n"
41                  + "   ST_distance(\n"
42                  + "    st_transform(st_setsrid(the_geom, 4326),
                        3787), \n"
43                  + "    st_transform(ST_GeomFromText(?1, 4326),
                        3787)\n"
44                  + "   ) as distance\n"
```

```
45                  + "  FROM\n"
46                  + "   poi_lj\n"
47                  + "  WHERE\n"
48                  + "   poi_type_id in (" + JPAUtil.implodeArray(",
                        ", poiType) + ")\n"
49                  + " ) POIs_C ON (POIs_B.min_distance = POIs_C.
                        distance) and (POIs_B.poi_type_id = POIs_C.
                        poi_type_id)\n"
50                  + " GROUP BY \n"
51                  + "  POIs_B.poi_type_id, POIs_B.min_distance\n"
52                  + ") RESULT ON poi_lj.gid = RESULT.min_gid";
53
54          Query customQuery = em.createNativeQuery(
                customSringQuery, PoiLjDistance.class);
55          customQuery.setParameter(1, "POINT(" + lon +" " + lat +
                ")");
56
57          List<PoiLjDistance> list = customQuery.getResultList();
58          return list;
59      } finally {
60          em.close();
61      }
62  }
```

Poizvedbo smo napisali v izvirnem načinu, saj vsebuje funkcije, ki so lastne le podatkovni bazi PostgreSQL z razširitvijo PostGIS. V kodi 5.4 je še funkcija, ki izračuna razdaljo.

S funkcijo st_setsrid določimo standard, v kateri je zapisana geometrija. Naša je zapisana v standardu WGS84 s kodo srid (Spatial Reference System Identifier) 4326. WGS84 [9] je standard, ki se uporablja globalno za opis lokacije na zemlje. Prav tako je to standard, ki ga uporablja GPS (Global Positioning System).

Ker je to stopinjski način zapisa koordinate, ki zemljo predpostavi kot sferoid, in ni primeren za računanje razdalje v metrih, moramo koordinate projicirati na ravnino. Ravninska projekcija, najboljša za Slovenijo, je Gauss Kruger projekcija s kodo srid 3787. Šele ko imamo koordinati transformi-

rani v ustrezno projekcijo, lahko izračunamo razdaljo med njima v metrih (o sistemih GIS in nekaj malega tudi o projekcijah ter standardih si lahko preberete v [1]).

Koda 5.4: Funkcija za izračun razdalje med dvema točkama

```
1 ST_distance(
2   st_transform(st_setsrid(the_geom, 4326), 3787),
3   st_transform(ST_GeomFromText('POINT(14.472435 46.044365)',
        4326), 3787)
4 ) as distance
```

5.2 Spletni servisi

Pri spletnih servisih smo uporabljali zahteve tipa GET in POST [7]. GET smo uporabili pri zahtevah, kjer smo želeli poslati samo en parameter. Te zahteve smo nato klicali tako, da smo parameter pripeli v spletni naslov. Primer take zahteve je v kodi 5.5. Zahteve POST smo uporabili v primerih, kjer smo želeli poslati večje število parametrov, tako da so bili vsi podatki znotraj telesa zahteve POST. Primer take zahteve je v kodi 5.6. Podatki se pošiljajo v formatu JSON. Knjižnice znotraj našega programa same skrbijo, da so vsi podatki pravilno pretvorijo v format JSON in ponovno nazaj v ustrezen objekt. Že arhitektura spletnih servisov namreč poskrbi, da se za vse POJO objekte ter vse liste in tabele uporabijo ustrezne knjižnice za pravilno pretvorbo. Priporočljivo je, da vsak POJO objekt, ki ga bo treba pretvoriti v format JSON, opremimo z anotacijo `@XmlRootElement`.

5.3 Obdelava glavne zahteve in sestava ustreznega odgovora

Glavna zahteva, je zahteva, na katero odgovor je računsko najbolj potraten in nam hkrati poda odgovor na problem, ki je temelj celotne diplomska na-

Koda 5.5: Primer zahteve GET

```
1 //Zahteva GET
2 @GET
3 @Path("locations/{regionId}")
4 @Produces("application/json")
5 public List<ValuePairHolder> getDistinctLocations(@PathParam("
      regionId") int rid) {
6   RealestateLjJpaController rljc = new RealestateLjJpaController
        (JPAUtil.getFactory());
7   return rljc.getRealestateLjDistinctLocations(rid);
8 }
```

Koda 5.6: Primer zahteve POST

```
1 //Zahteva POST
2 @POST
3 @Path("getResults")
4 @Produces("application/json")
5 @Consumes("application/json")
6 public MasterResponse getResults(MasterRequest request) {
7   MasterResponseController mrc = new MasterResponseController(
        request);
8   return mrc.processResponse();
9 }
```

loga. V tej zahtevi se torej obdelajo vsi poslani prametri in uteži, na kar nam strežnik poda odgovor, ki vsebuje deset najbolje ocenjenih nepremičnin glede na uteži in bližino interesnih točk. Obdelavo glavne zahteve najlažje obrazložimo po korakih.

V obrazložitvi programa bomo pisali o tipih interesnih točk in najdenih interesnih točkah. Tu gre za tipe, ki jih uporabnik izbere in uteži na odjemalcu, medtem ko gre pri najdenih interesnih točkah za interesne točke, ki jih uporabnik najde s pomočjo iskalnika in so točno določene interesne točke, ki jih hoče imeti uporabnik v bližini. Najdene interesne točke prav tako uteži z neko utežjo. Več o tem v opisu odjemalca v poglavju 6.2.

1. Spletni servis prejme zahtevo POST, kjer se podatki v formatu JSON samodejno pretvorijo v POJO objekt tipa MasterRequest. Dobljeni objekt nato pošljemo na obdelavo h kontrolerju, ki je zadolžen za izdelavo glavnega odgovora.

2. Najprej inicaliziramo kontrolerja za delo z nepremičninami in interesnimi točkam ter pokličemo poizvedbo, ki vrne vse nepremičnine, ki ustrezajo danim parametrom, ki smo jih prejeli v glavni zahtevi. Dobimo torej nepremičnine, ki ustrezajo filtrom, ki jih je uporabnik nastavil na odjemalcu.

3. V tretjem koraku izračunamo delež, ki ga ima starost stanovanja in razmerje cena/velikost pri končni oceni.

$$\text{delež}_{\text{starosti}} = \frac{\text{utež}_{\text{starosti}}}{\text{utež}_{\text{starosti}} + \text{utež}_{\text{cena na m}^2}} \cdot \text{delež uteži}_{\text{nepremičnin}}$$

$$\text{delež}_{\text{cena na m}^2} = \frac{\text{utež}_{\text{cena na m}^2}}{\text{utež}_{\text{starosti}} + \text{utež}_{\text{cena na m}^2}} \cdot \text{delež uteži}_{\text{nepremičnin}}$$

4. Sprehodimo se preko uteži, določenih za tipe interesnih točk; te seštejemo in ko imamo vsoto vseh uteži, še vsako utež normaliziramo po formuli.

$$\text{delež uteži}_i = \frac{\text{utež}_i}{\sum \text{utež}_i} \cdot \text{delež uteži}_{\text{tipov interesnih točk}}$$

5. Enako kot v četrti točki naredimo tudi za uteži za skupino najdenih interesnih točk.

6. Sprehodimo se preko nepremičnin, ki smo jih dobili v drugi točki, in za vsako nepremičnino kličemo poizvedbo, ki vrne najbližje interesne točke določenih tipov, in poizvedbo, ki vrne razdalje najdenih interesnih točk. Rezultat teh dveh poizvedb pripnemo na posamezno nepremičnino. Pri tem si še shranimo maksimum in minimum za starost, razmerje cena/velikost, razdaljo od nepremičnine za vsak tip interesnih točk in razdaljo najdenih interesnih točk.

7. Nato se zopet sprehodimo preko nepremični in tokrat izračunamo končno oceno ter oceno posameznih atributov. Pri tem uporabimo enostavno linearno formulo, kjer manjša kot je vrednost atributa večja je ocena. Končna ocena je le vsota vseh ocen atributov za vsako nepremičnino.

$$\text{ocena} = \frac{\text{atribut}_{max} - \text{atribut}}{\text{atribut}_{max} - \text{atribut}_{min}} \cdot \text{atribut}_{\text{delež}}$$

8. Na koncu še razvrstimo nepremične po končni oceni, uteži tipov interesnih točk in uteži najdenih interesnih točk po njihovem deležu. Nato shranimo vse potrebne parametre, uteži in deset najboljših nepremičnin v objekt MasterResponse, ki ga posredujemo kot odgovor odjemalcu, ki je posredoval zahtevo. Objekt se avtomatsko pretvori v objekt formata JSON.

5.4 Nastavitve strežnika

Brskalnik v privzetem načinu ne dovoli, da kličemo spletne servise v razvojnem okolju zaradi tako imenovane zavrnitve *Access-Control-Allow-Origin*. Na spletu je več rešitev kako se izogniti temu problemu. Izbrali smo rešitev, kjer na naš strežnik dodamo zmožnost konfiguriranja CORS (*Cross-Origin Resource Sharing*) filtrov. Iz vira [10] lahko razberemo, da je CORS rezultat

truda po standardiziranju mehanizma za omogočanje *cross-domain* zahtev v brskalnikih in strežnikih.

Tako smo s spletnega mesta [10] prenesli dve datoteki tipa jar, ki smo jih skopirali v mapo lib, v kateri so ostale knjižnice našega spletnega strežnika Apache Tomcat. Nato je bilo treba ugasniti strežnik, ga osvežiti in potem ponovno zagnati, da je strežnik zaznal dve novi dodani knjižnici. Preostalo nam je samo še, da smo napisali konfiguracijo v kodi 5.7 v našo nastavitveno datoteko web.xml in ponovno zagnali strežnik.

Koda 5.7: Konfiguracija CORS filtra

```
<filter>
  <filter-name>CORS</filter-name>
  <filter-class>com.thetransactioncompany.cors.CORSFilter</
     filter-class>
      <init-param>
            <param-name>cors.allowGenericHttpRequests</param-name
               >
            <param-value>true</param-value>
      </init-param>

      <init-param>
            <param-name>cors.allowOrigin</param-name>
            <param-value>*</param-value>
      </init-param>
      <init-param>
            <param-name>cors.supportedMethods</param-name>
            <param-value>GET, HEAD, POST, OPTIONS</param-value>
      </init-param>

      <init-param>
            <param-name>cors.supportedHeaders</param-name>
            <param-value>*</param-value>
      </init-param>
</filter>
<filter-mapping>
      <filter-name>CORS</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

Poglavje 6

Odjemalec

Za odjemalca smo naredili nov projekt znotraj orodja Netbeans in ga zastavili kot spletno aplikacijo, ki ustreza standardom HTML5. Cel projekt temelji na naslednjih tehnologijah [7, 12, 11]

- CSS3 – standard, ki določa kako so naši elementi html prikazani
- HTML5 – jezik za strukturiranje in predstavitev vsebine na globalnem omrežju
- jQuery – je knjižnica, ki olajša ter poenostavi delo z JavaScriptom
- jQuery UI – knjižnica, bazirana na jQueryu, ki ponudi na razpolago mnogo elementov, animacij in pripomočkov, ki nam olajšajo sestavljanje uporabniškega vmesnika.
- Google Maps JavaScript API v3 – zbirka funkcij za prikaz GoogleMap karte.

Celoten uporabniški vmesnik je zasnovan tako, da se vse dogaja asinhrono na eni strani, podobno kot spletna aplikacija GoogleMaps. To pomeni, da odjemalec nima podstrani in se vse dogaja na eni glavni strani. Za komunikacijo s strežnikom uporabljamo ajax klice. Primer enega takih klicev imamo v kodi 6.1.

Koda 6.1: Primer ajax klica

```
1  $.ajax({
2    url: "http://localhost:8084/RealestatePoisRestServices/
         webresources/realestates/poiSearch",
3    type: "POST",
4    contentType: "application/json; charset=utf-8",
5    dataType: "json",
6    data: request.term,
7    success: function( data ) {
8      response( $.map( data, function( item ) {
9        return {
10       label: item.ime + (item.ulIme ? ", " + item.ulIme : ""),
11       value: item.ime + (item.ulIme ? ", " + item.ulIme : ""),
12       gid: item.gid,
13       ulica: item.ulIme
14       };
15     }));
16   }
17 });
```

Celoten uporabniški vmesnik je bil zasnovan tako, da uporablja elemente HTML ter elemente knjižnice jQuery UI. Zasnovan je okoli teme knjižnice jQuery UI, kar pomeni, da smo elemente HTML opremili z ustreznimi razredi, da jih je nato CSS teme prikazal v ustreznem stilu in tematiki. Temo si lahko vsak zasnuje sam na strani jQueryUI [11] s pomočjo izbirnika teme (*ThemeRoller*). Vse, kar mora storiti potem, je da zamenjamo CSS sklic za temo na sklic za svojo temo.

Uporabili smo kar nekaj elementov in zmožnosti knjižnice jQuery UI, ki jih v nadaljevanju povzemamo:

- *Sortable* – naredi skupek elementov sotrabilnih.
- *Accordion* – naredi vertikalen interaktivni meni.
- *Button* – ustrezno oblikuje gumb, da ustreza izgledu teme.
- *Progressbar* – ustvari vrstico napredka.
- *Slider* – ustvari drsnik.
- *Tabs* – naredi interaktiven meni v zavihkih.
- *Tooltip* – naredi namig za določene elemente.
- *Hide, show* – skrije ali pokaže nek element, če želimo lahko uporabimo tudi animacijo.

Vsi zgoraj omenjeni elementi delujejo po podobnem principu, zato podajamo enostaven primer izdelave drsnika. Na strani jQuery UI [11] so odlični primeri implementacije vseh teh elementov po katerih smo se zgledovali, zato se v podrobnejšo razlago implementacije nekaterih funkcionalnost na odjemalcu ne bom spuščali.

1. Najprej v naš html dokument napišemo div element in ga opremimo z lastnostjo id, da se bomo lahko nato v JavaScriptu sklicevali na ustrezni element. `<div id="drsnik"></div>`

2. V naši datoteki JavaScript pokličemo ustrezno funkcijo s pravilno nastavljenimi nastavitvami, ki nam element opremi v drsnik. Spodaj je primer, kako naredimo drsnik, ki ima minimalno vrednost 0, maksimalno 100, drsnik premika vrednost po +/-2 naenkrat, začetna vrednost je 20.

```
$( "#drsnik" ).slider({
     step: 2,
  min: 0,
  max: 100,
     value: 20
});
```

3. Do vrednosti drsnika lahko nato pridemo na naslednji način

```
$('#drsnik).slider("value");
```

ali pa jo programsko priredimo

```
$('#drsnik).slider("value", 15);
```

6.1 Kreiranje glavne zahteve in prikaz odgovora

Kot smo v poglavju 5.3 opisali obdelavo glavne zahteve bomo v tem poglavju opisali kako se ta zahteva na odjemalcu kreira in kako odjemalec obdela odgovor za prikaz.

1. Ob kliku na gumb za pridobitev rezultatov se sproži klik dogodek, ki kliče funkcijo, ki najprej preveri ali so vsi obvezni parametri izpolnjeni in če so nadaljuje s kreiranjem glavne zahteve. V nasprotnem primeru uporabnika obvesti z ustreznim obvestilom.

2. Pred kreiranjem zahteve uporabniku prikažemo vrstico napredka in skrijemo gumb, da uporabnik ne more poslati še ene zahteve, medtem ko je ena v obdelavi.

3. Pri kreiranju zahteve preberemo vse vrednosti iz ustreznih elementov input, checkbox ter preberemo uteži in shranjene parametre z drsnikov in ustreznih vrstic napredka. Vse skupaj zapakiramo v tako imenovani *master* objekt, ki ga nato pošljemo preko zahteve POST na strežnik. Pri tem bi radi poudarili, da je treba *master* objekt pretvoriti v ustrezen format JSON s pomočjo funkcije

   ```
   JSON.stringify(master)
   ```

4. Ob uspešnem odgovoru na zahtevo skrijemo vrstico napredka, prikažemo nazaj gumb za pridobitev rezultatov in poženemo funkcijo za obdelavo odgovora.

5. Pri obdelavi odgovora najprej preberemo tip in iskane interesne točke ter sestavimo ustrezno strukturo HTML, na katero bomo pripeli vrstice napredka za interesne točke vsake nepremičnine posebej.

6. V tem koraku se podamo v zanko, v kateri se sprehodimo preko vseh deset dobljenih nepremičnin. Tu sestavimo strukturo html za vsako nepremičnino posebej in jo združimo s strukturo sestavljeno iz pete točke, ter jo pripnemo na ustrezno mesto v DOM naše strani.

7. Po tem ko smo pripeli strukturo v DOM, je treba še ustrezno opremiti vse vrstice napredka in vrstice napredka interesnih točk opremiti z namigi, v katerih je celoten opis interesne točke. Prav tako skrijemo elemente, ki jih ne želimo imeti prikazanih, v začetnem prikazu, ter pripnemo poslušalce klika na ustrezne elemente. Te so največkrat vezani na funkcijo prikaži oziroma skrij.

8. Določene elemente opremimo tudi s podatki s funkcijo

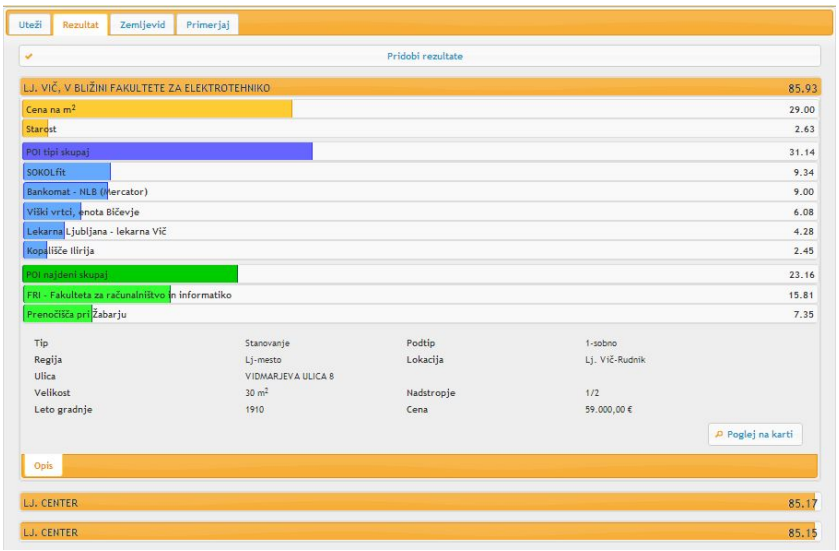
   ```
   $(element).data('kljuc', podatki)
   ```

   Te podatke lahko nato uporabimo v drugih funkcionalnostih. Na primer gumb za prikaz nepremičnine in interesnih točk na zemljevidu smo

opremili s podatkom o objektu, ki hrani nepremičnino, skupaj s pripetimi interesnimi točkami. Podatek o nepremičnini nato posredujemo ustrezni funkciji za prikaz zemljevida, tako da ga preberemo z gumba s funkcijo

```
$(element).data('kljuc')
```

Primer prikaza odgovora si lahko ogledamo na sliki 6.1.

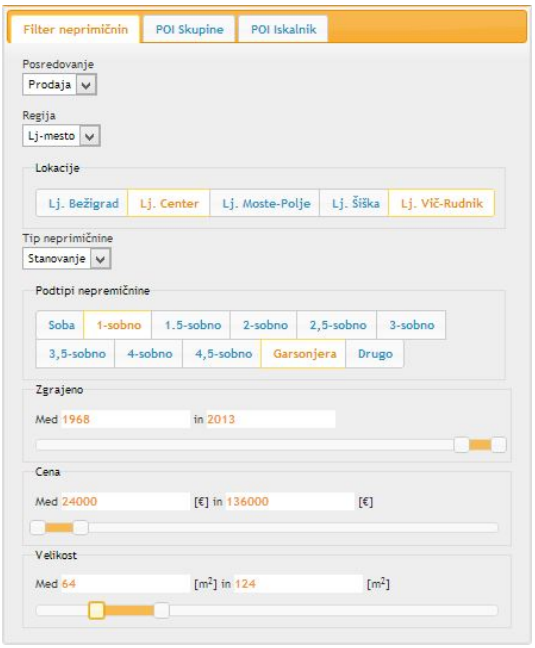


Slika 6.1: Primer prikaza rezultata

6.2 Opis odjemalca

V tem poglavju bomo opisali običajno uporabo odjemalca iz vidika uporabnika in celoten postopek opremili s slikami.

1. Uporabnik najprej izpolni obrazec filter nepremičnin, kjer določi, kakšno nepremičnino išče (primer na sliki 6.2). Vsa polja v filtru se napolnijo dinamično iz podatkovne baze s pomočjo ajax klicev na naše spletne servise.



Slika 6.2: Filter

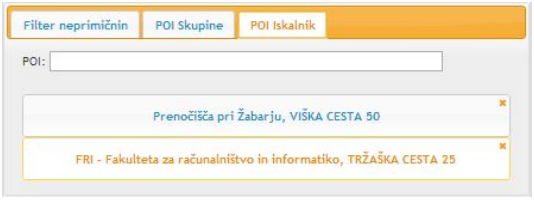
2. Nato v zavihku POI skupine izbere vse morebitne tipe interesnih točk, ki jih želi imeti v bližini svoje nepremičnine (prikazano na sliki 6.3). V primeru da se zmoti, lahko izbiro odstrani iz izbora, tako da klikne na gumb odstrani za ustrezen tip interesne točke v zavihku uteži (prikazano na sliki 6.4).

3. V tretjem koraku uporabnik v zavihku POI iskalnik (prikazan na sliki 6.5) najde točno določene interesne točke, ki jih želi imeti v bližini nepremičnine. S klikom na najdene interesne točke jih doda med izbiro v zavihek uteži. Če jih želi odstraniti iz izbire, jih enostavno ponovno klikne ali pa klikne na gumb odstrani v zavihku uteži. Najdeno interesno točko lahko odstranimo s seznama najdenih z enostavnim klikom na križec.

4. V zavihku uteži imamo tri vrstice napredka, s katerimi določamo kolikšen je celoten delež posameznih skupin uteži. Tako imamo skupine

Slika 6.3: Izbira tipov interesnih točk

Slika 6.4: Zavihek uteži

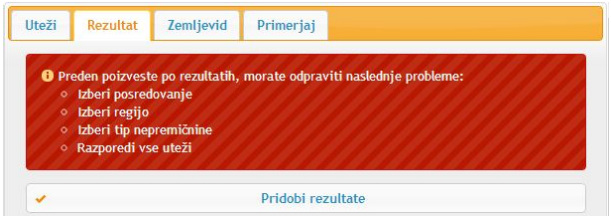


Slika 6.5: Iskanje interesnih točk

uteži za nepremičnine, tipe interesnih točk in najdene interesne točke. Z gumboma minus in plus uporabnik odvzema oziroma dodaja vrednost deleža posamezne skupine. Delež vseh skupin ne more biti večji kot 100, kar nam prikazuje prva vrstica napredka. Primer zavihka uteži imamo na sliki 6.4.

5. Z drsniki v zavihku uteži uporabnik določi pomembnost posameznega atributa. Vrednost uteži lahko uporabnik vpiše tudi v vnosno polje.

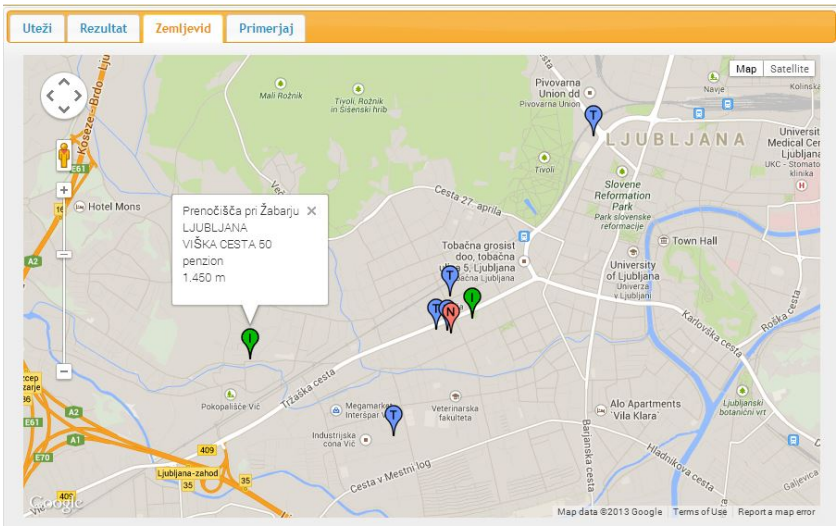
6. Ko je uporabnik dokončno nastavil uteži, lahko v zavihku rezultat poizve o ustreznih nepremičninah, s klikom na gumb pridobi rezultate. V primeru, da uporabnik ni izpolnil vseh obveznih vnosov, nas o tem obvesti ustrezno obvestilo (prikazano na sliki 6.6).



Slika 6.6: Obvestilo o napaki

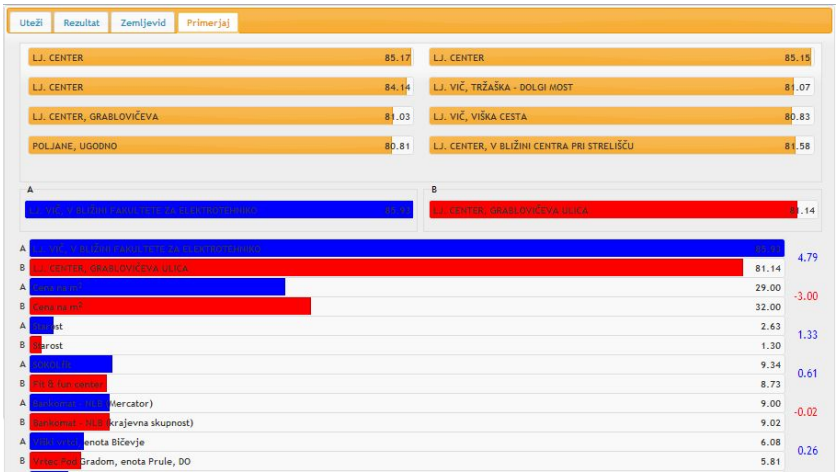
7. Rezultati se prikažejo in uporabnik lahko s klikom na posamezen rezultat pride do podrobnejšega opisa rezultata in nepremičnine (primer na sliki 6.1). Podobno lahko do podrobnejšega opisa pride tudi, če klikne na posamezno skupino, torej POI tipi skupaj ali POI najdeni skupaj.

Če se uporabnik z miško postavi na katero koli vrstico napredka, ki opisuje interesno točko, se mu pokaže namig, s podrobnim opisom interesne točke. Spodaj imamo opisane še atribute stanovanja ter gumb, ki nam prikaže nepremičnino in interesne točke v zavihku zemljevid (prikazano na sliki 6.7).



Slika 6.7: Prikaz nepremičnine in interesnih točk na karti

8. Zavihek primerjaj služi za direktno primerjavo rezultatov (primer na sliki 6.8). Uporabnik mora le odvleči rezultata, ki ju želi primerjati, v okno A in okno B.



Slika 6.8: Primerjava rezultatov

Poglavje 7

Sklep

V diplomski nalogi je bil razvit preprost prototipni sistem za priporočanje nepremičnin glede na bližino interesnih točk. Postavljeni so bili temelji, ki jih potrebujemo za razvoj takega sistema in zbrane ustrezne tehnologije. Tako smo razvili ustrezne spletne servise, postavili podatkovno bazo z vsemi potrebnimi razširitvami in razvili spletni odjemalec.

Sistem lahko uporabniku bistveno pomaga pri odločitvi nakupa nepremičnine, prav tako je sam prototip lahko dobro izhodišče za razvoj kateregakoli podobnega priporočilnega sistema, ki temelji na podatkih iz sistema GIS.

Sistem je tudi zelo razširljiv in ponuja veliko možnosti za razširitev ter izboljšave. Naj naštejem le nekaj možnih funkcionalnosti, ki bi sistemu dodale dodatne vrednosti: lahko bi na primer dodali integracijo socialnih omrežji kot so Facebook ali FourSquare za nastavitev privzetih izbir za tipe interesnih točk. Lahko bi tudi dodali možnosti negativne ocene, torej za tipe interesnih točk, ki jih ne želimo, lahko bi za razdaljo določili, da nas zanima namesto zračne razdalje razdalja po cestni mreži, lahko bi za ocene bližine in ostalih atributov ponudili več možnosti glede načina izračuna ocene poleg linearne. Skratka možnosti je veliko in znamo si predstavljati produkcijsko uspešen sistem. Potrebno bi bilo le prepričati ponudnike nepremičnin, na katere bi se sistem povezoval, da svoje nepremičnine lokacijsko določijo s hišnim naslovom ali koordinato.

Literatura

[1] K. Kvamme, K. Oštir-Sedej, Z. Stanić, R. Šumrada, *Geografski informacijski sistemi*, Ljubljana, 1997.

[2] E. Jendrock, R. Cervera-Navarro, I. Evans, D. Gollapudi, K. Haase W. Markito, C. Srivathsa, *The Java EE 6 Tutorial*, Oracle, jan. 2013.

[3] M. Škrinjar, G. Bizjak, ustna komunikacija, 2013.

[4] (2013) Spletno mesto PostGISa. Dostopno na: http://www.postgis.net.

[5] (2013) Spletna dokumentacija podatkovne baze PostgreSQL in nekaterih njenih razširitev. Dostopno na: http://www.postgresql.org/docs/.

[6] (2013) Spletno mesto knjižnice jsoup. Dostopno na: http://jsoup.org/.

[7] (2013) Spletno mesto w3school. Dostopno na: http://www.w3schools.com/.

[8] (2013) Wikipedijin opis tehnologije REST. Dostopno na: http://en.wikipedia.org/wiki/Representational_state_transfer.

[9] (2013) Wikipedijin opis standarda WGS84. Dostopno na: http://en.wikipedia.org/wiki/World_Geodetic_System.

[10] (2013) Spletno mesto CORS filtra. Dostopno na:
http://software.dzhuvinov.com/cors-filter.html.

[11] (2013) Spletno mesto knjižnice jQuery UI. Dostopno na:
http://jqueryui.com/.

[12] (2013) Spletno mesto knjižnice jQuery. Dostopno na:
http://jquery.com/.