



Sveučilište u Zagrebu – Geodetski fakultet
University of Zagreb – Faculty of Geodesy

Katedra za geoinformatiku
Chair of Geoinformation Science

Kačićeva 26, 10000 Zagreb, Croatia
Web: <http://geoinfo.geof.hr>; Tel.: +385 (1) 46 39 227; Fax: +385 (1) 48 26 953

Diplomski rad

Upravljanje prostornim i atributnim podacima pomoću GeoDjango tehnologije

Izradio: Tomislav Pavlović

Mentor: prof. dr. sc. Damir Medak

Zagreb, rujan 2010.

Upravljanje prostornim i atributnim podacima pomoću GeoDjango tehnologije

Sažetak: U ovom radu se opisuje upravljanje prostornim i atributnim podacima te izrada praktičnog projekta korištenjem GeoDjango dodatka za Django. Praktični projekt opisuje izradu web aplikacije za aktivno sudjelovanje građana neke gradske četvrti u prijavljivanju kvarova i problema na javnim površinama. Kao prostorna baza podataka je korišten PostgreSQL s PostGIS dodatkom, a aplikacija je pisana u programskom jeziku Python.

Ključne riječi: Django, GeoDjango, PostgreSQL, PosGIS, otvoreni kod, Python

Managing spatial and non-spatial data with GeoDjango

Abstract: This paper describes managing spatial and non-spatial data with GeoDjango and creating web application using Django framework and GeoDjango extension. Application is used for active citizen participation in city district where citizens reports problems in public spaces. Spatial database used for this project is PostgreSQL with PostGIS extension and application is written in Python programming language.

Keywords: Django, GeoDjango, PostgreSQL, PostGIS, open-source, Python

Sadržaj

1. Uvod	1
2. Korištene tehnologije	2
2.1. Python	2
2.2. HTML	3
2.3. CSS	3
2.4. JavaScript	4
2.5. OGC standardi	4
2.5.1. OpenGIS Simple Features	5
2.5.2. OpenGIS Simple Features Implementation Specification for SQL	7
2.6. GeoJSON	7
2.7. Google Maps i Google Maps API	8
3. Korišteni alati i programi	9
3.1. Poslužiteljski softver	9
3.1.1. Apache	9
3.1.2. PostgreSQL i PostGIS	10
3.2. Softverska okruženja	11
3.2.1. Django	11
3.2.2. GeoDjango	15
3.2.3. Prostorne biblioteke	17
4. Korištenje mogućnosti GeoDjanga	18
4.1. Unos podataka	18
4.1.1. Ispitivanje datoteke za unos	18
4.1.2. Stvaranje modela	21
4.1.3. Uvoz podataka	23
4.2. Prostorni upiti	25
4.3. GeoAdmin i OSMGeoAdmin	28
5. Praktični rad	30
5.1. Instalacija softvera i podešavanje okruženja	31
5.1.1. Instalacija poslužiteljskog softvera i biblioteka	31
5.1.2. Podešavanje PostgreSQL-a i stvaranje PostGIS baze podataka	32
5.1.3. Instalacija Django okruženja	33

5.1.4.	Stvaranje Django projekta.....	34
5.1.5.	Podešavanje Django projekta	35
5.1.6.	Stvaranje Django aplikacije.....	38
5.2.	Stvaranje modela podataka i podešavanje administracijskog sučelja	38
5.3.	Unos inicijalnih podataka	42
5.4.	Stvaranje korisničkog sučelja.....	43
5.4.1.	Predlošci	43
5.4.2.	Jednostavan pogled.....	45
5.4.3.	URL uzorci	45
5.4.4.	Stvaranje pogleda u aplikaciji.....	46
5.5.	Daljnji razvoj.....	55
6.	Zaključak.....	56
7.	Literatura.....	57
8.	Popis slika	59
9.	Prilozi	60
9.1.	Sadržaj priloženog medija.....	60
10.	Životopis	61

1. Uvod

Klasični geoinformacijski sustavi koji omogućuju pregledavanje i izmjenu prostornih podataka često su skupi te ih koriste rijetke tvrtke i ustanove koje si to mogu priuštiti. Ponekad su nedovoljno fleksibilni i nisu primjereni malim i srednjim tvrtkama.

Kako bi se GIS približili malim tvrtkama, oni moraju biti jednostavni i jeftini. Danas je to moguće putem *open-source* alata (alata otvorenog koda) i besplatnih servisa za prikaz digitalnih karata (npr. Google Maps). Budući da takvi alati i servisi već postoje, nisu potrebna dodatna financijska i vremenska ulaganja u njihov razvoj te je konačni proizvod (GIS) jeftiniji. Takvim se sustavima pristupa preko Interneta, što je dodatna prednost jer je za korištenje potreban jedino Internet preglednik i pristup Internetu.

Veliku ulogu u takvim sustavima imaju standardi i specifikacije o načinu pohrane, uređivanju i razmjeni prostornih podataka koji olakšavaju nadogradnje sustava i povezivanje s postojećim. U ovom se radu opisuje razvoj jednog takvog sustava koji se temelji na *open-source* tehnologijama. Sustav je zamišljen kao besplatan servis putem kojeg građani gradske četvrti Donji Grad u Zagrebu mogu aktivno sudjelovati u prijavljivanju problema ili kvarova koji se mogu pojaviti na javnim površinama. Osnova za pohranu prostornih podataka je PostgreSQL baza podataka s PostGIS dodatkom, a aplikacijsko je okruženje Django s GeoDjango dodatkom.

2. Korištene tehnologije

Za GeoDjango web aplikaciju korištene su različite tehnologije, programski jezici i standardi. Osnova se aplikacije temelji na Django okruženju (*frameworku*) koje je pisano u programskom jeziku Python. Django omogućuje upravljanje prostornim podacima pomoću dodatka GeoDjango koji implementira OGC *OpenGIS Simple Features Implementation Specification for SQL*. Prikazni dio aplikacije, koji korisnik vidi u Internet pregledniku, izveden je pomoću HTML-a, CSS-a i JavaScripta, a kao podloga su korištene karte dobivene Google Maps programskim sučeljem (Google Maps API).

2.1. Python

Python je dinamički i objektno orijentirani programski jezik koji može biti primijenjen u različite svrhe (URL 1). Može se usporediti s programskim jezicima Tcl, Perl, Ruby, Scheme ili Javom. Odlikuje se vrlo čistom, čitljivom sintaksom i bogatom standardnom bibliotekom koja pokriva područja kao što su:

- upravljanje tekstom (regularni izrazi, *Unicode*, datoteke),
- Internet protokoli (HTTP, FTP, SMTP, XML-RPC, POP, IMAP) i
- operacijski sustavi (sistemske pozivi, datotečni sustavi, TCP/IP).

Python kod se može izvoditi na svim važnijim operacijskim sustavima (Windows, Linux/Unix, OS/2, OS X), a uključen je većinu GNU/Linux distribucija i OS X gdje nije potrebna posebna instalacija.

Za pokretanje Python programa potreban je Python interpreter. Referentni i najčešće korišteni interpreter je CPython pisan u programskom jeziku C. Uključena standardna biblioteka pisana je djelomično u C-u, djelomično u Pythonu. Napisani Python program interpreter prevodi u međukod (*bytecode*) koji se izvršava u virtualnom okruženju.

Python standardna biblioteka dobro je prilagođena pisanju web aplikacija, ali ne uključuje module za direktnu komunikaciju s bazama podataka već pruža API (*Application programming interface*) (URL 2) pomoću kojeg se mogu napraviti takvi moduli. Web aplikacije najčešće koriste baze podataka pa su takvi moduli

izvan standardne biblioteke već dostupni za sve bitnije baze podataka (Oracle, MSSQL, PostgreSQL, MySQL).

Jedan je od takvih modula i `psycopg2` (URL 3) za komunikaciju s PostgreSQL bazom podataka. Koristi sve mogućnosti Python DB API-ja i može se sigurno koristiti u višedretvenom okruženju (dretve mogu zajednički koristiti istu konekciju prema bazi podataka).

2.2. HTML

HTML (*HyperText Markup Language*) je jezik za stvaranje web stranica. Koristi se za strukturiranje sadržaja iako je prvotno bio zamišljen kao jezik kojim se može definirati i struktura i izgled samih web stranica. Stranica napisana HTML-om može sadržavati tekstualne i multimedijalne sadržaje, kao i skripte (JavaScript) kojima se postiže povećana interaktivnost samih stranica. Za pregled se koriste različiti Internet preglednici (Mozilla Firefox, Internet Explorer, Chrome, Safari, Opera...) zbog čega je bitno da se svi pridržavaju standarda. Standarde razvija W3C (*World Wide Web Consortium*), a zadnji dovršeni HTML standard je XHTML 1.1 (URL 4) koji je nastao kako bi HTML bio kompatibilan s XML sintaksom.

Novi HTML5 standard (URL 5), koji je još u razvoju, donosi nove mogućnosti kao što su izvođenje videozapisa bez korištenja dodatka za Internet preglednik i elemente za direktno dvodimenzionalno prikazivanje vektorskih crteža (*canvas*). Uvedeni su i novi elementi koji su zaduženi za bolje semantičko označavanje tipova podataka koje sadrži.

2.3. CSS

CSS (*Cascading Style Sheets*) je jezik koji se koristi za opisivanje izgleda dokumenata pisanih u HTML-u i sličnim jezicima. CSS je primarno stvoren kako bi odvojio sadržajni od prezentacijskog dijela web stranice. CSS-om je moguće referencirati elemente HTML-a te im dodavati stilska svojstva kao što su boja teksta, boja pozadine i veličina fonta. Kao i HTML standarde, tako i CSS specifikacije razvija W3C. Zadnja je dovršena verzija CSS specifikacije CSS2 (URL 6), a CSS3 je još u razvoju. Najnovije inačice Internet preglednika

podržavaju većinu mogućnosti iz CSS3 specifikacije te se očekuje da će puna podrška biti dostupna u svim preglednicima najkasnije 2012. godine (URL 7).

2.4. JavaScript

JavaScript je skriptni programski jezik za dinamičko programiranje web stranica kako bi one bile što interaktivnije. Izvršava se u Internet pregledniku korisnika koji pregledava stranice. Sintaksom podsjeća na programski jezik C i Javu, ali se smatra funkcionalnim programskim jezikom zbog čega je sličniji jezicima Scheme i Ocaml.

JavaScriptom je moguća direktna manipulacija elementima web stranica pa time one dobivaju na interaktivnosti. U današnje je vrijeme JavaScript sve bitniji jer postoji veliki broj postojećih biblioteka i programskih sučelja kojima se pristupa tim jezikom.

JavaScript biblioteke olakšavaju sam rad i ubrzavaju programiranje zbog mnoštva gotovih funkcija koje se često koriste kod razvoja web stranica. U ovom je projektu korištena Query biblioteka (URL 8) za jednostavniji pristup elementima stranica i asinkroni dohvat podataka (AJAX) (URL 9).

2.5. OGC standardi

Open Geospatial Consortium (Udruženje za otvorenost prostornih podataka) međunarodno je udruženje koje okuplja 401 tvrtku, vladine organizacije i sveučilišta radi stvaranja javno dostupnih standarda vezanih uz prostorne podatke (URL 10). Ti su standardi zamišljeni kao rješenje za razmjenu prostornih podataka na Internetu, bežičnim i položajno orijentiranim servisima i općenito u računalnim informacijskim sustavima.

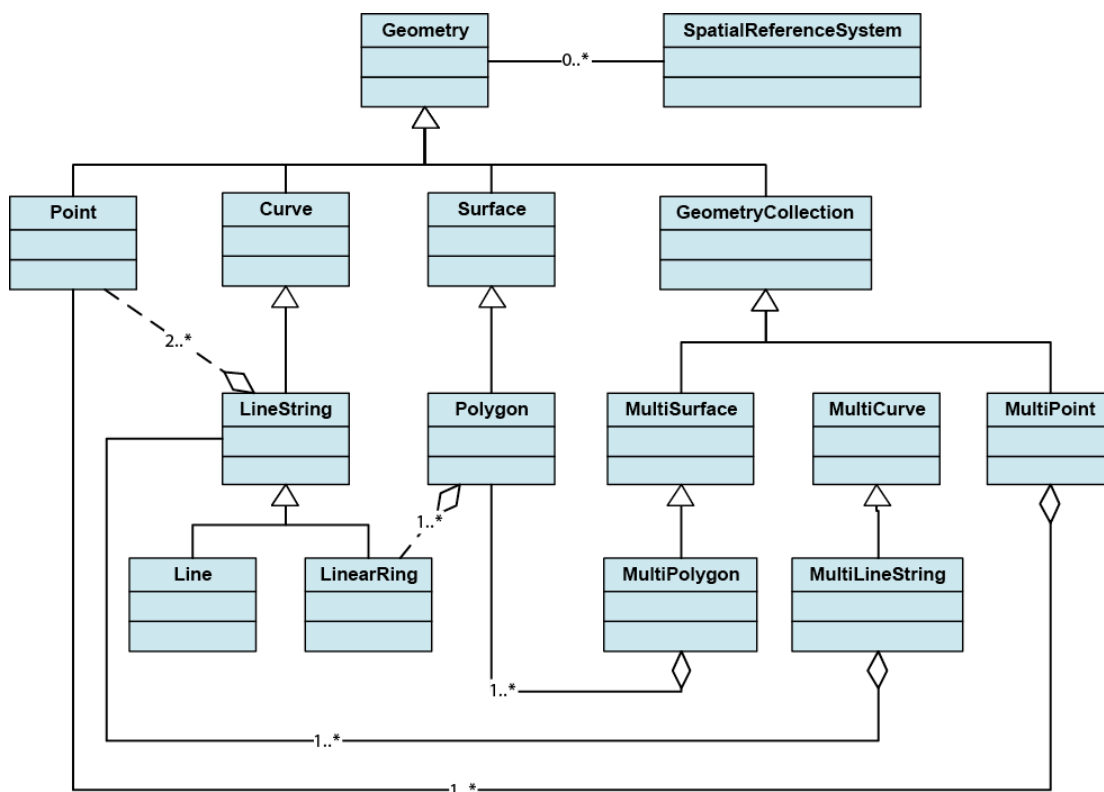
OGC standardi i specifikacije su tehnički dokumenti koji opisuju sučelja prema kojima razvojni programeri stvaraju podršku u svojim programskim paketima ili servisima. U idealnom slučaju, kada dva programera u svom softveru nezavisno implementiraju mogućnosti prema specifikacijama, rezultat bi trebala biti ispravna (bez grešaka) razmjena podataka između ta dva nezavisno razvijana softverska paketa ili servisa.

OGC standardi i specifikacije nose naziv *OpenGIS Standards*, a u nastavku su detaljnije opisani *Simple Features* (jednostavna obilježja) i *Simple Features Implementation Specification for SQL* (specifikacija implementacije jednostavnih obilježja za SQL). U ovom radu te specifikacije implementiraju PostGIS dodatak za PostgreSQL bazu podataka i GeoDjango dodatak za Django.

2.5.1. OpenGIS Simple Features

Simple Features (jednostavna obilježja) je *OpenGIS* standard koji određuje način na koji se atributni i prostorni podaci (npr. točke, linije, poligoni i sl.) pohranjuju u digitalnom obliku. Jednostavna se obilježja temelje na dvodimenzionalnoj geometriji s linearnom interpolacijom među vršnim točkama. Navedeni standard također definira i operatore i metode za stvaranje i manipulaciju prostornim podacima.

U tom su standardu prostorni podaci prikazani objektnim modelom za što se koristi UML dijagram (Slika 1).



Slika 1. Hijerarhija tipova geometrije

Osnovna je klasa prostornog podatka geometrija (*Geometry*), a ostali oblici nasljeđuju tu klasu. Klase koje izravno nasljeđuju osnovnu klasu su točka (*Point*),

krivulja (*Curve*), ploha (*Polygon*) i kolekcija geometrija (*GeometryCollection*). Svaki je geometrijski objekt povezan s referentnim koordinatnim sustavom (*SpatialReferenceSystem*) koji opisuje u kojem je koordinatnom sustavu taj objekt definiran. Osnovna klasa sadrži metode koje su zajedničke svim ostalim geometrijama, a podijeljene su na tri tipa: metode na geometrijskim objektima, metode za ispitivanje prostornih odnosa između geometrijskih objekata i metode za prostornu analizu (Tablica 1). Geometrije koje nasljeđuju tu klasu mogu imati svoje metode koje su usko vezane uz njihov tip. Na primjer, poligon ima metodu za dohvaćanje površine (*Area*) koja se ne može primijeniti na neke druge tipove geometrije kao što je točka.

Osnovne metode na geometrijskim objektima	Metode za ispitivanje prostornih odnosa između geometrijskih objekata	Metode za prostornu analizu
Dimension GeometryType SRID Envelope AsText AsBinary IsEmpty IsSimple Boundary	Equals Disjoint Intersects Touches Crosses Within Contains Overlaps Relate	Distance Buffer ConvexHull Intersection Union Difference SymDifference

Tablica 1. Metode osnovne geometrijske klase

Svaka geometrija ima WKT (*Well-known text*) prikaz koji koristi alfanumeričke znakove. Takav se prikaz koristi za stvaranje novih geometrijskih objekata i pretvaranje već postojećih geometrijskih objekata u tekstualni prikaz. Prema tome se jasno vidi o kakvom se geometrijskom objektu radi i koje su koordinate točaka (Tablica 2).

Tip objekta	Primjer tekstualni prikaza (WKT)
Točka	POINT(1 3)
Linija	LINestring(1 3, 2 2)
Poligon	POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))
Kolekcija geometrija	GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINestring (15 15, 20 20))

Tablica 2. Primjer tekstualnog prikaza geometrijskih objekata

2.5.2. OpenGIS Simple Features Implementation Specification for SQL

Svrha je ove specifikacije definirati standardnu SQL shemu pomoću koje je kolekciju prostornih obilježja moguće pohranjivati, dohvaćati, postavljati upite i ažurirati. Jednostavna su obilježja opisana *OpenGIS Simple Features* specifikacijom, koja se navodi u prethodnom poglavlju, prema kojem ta obilježja mogu imati prostorne i atributne podatke. Prostorni su podaci temeljeni na dvodimenzionalnoj geometriji s linearnom interpolacijom među točkama.

U relacijskoj je bazi podataka svako obilježje spremljeno kao redak u tablici (relaciji). Atributni podaci obilježja spremaju se u polja određena SQL92 standardom (npr. numerički tip, tekstualni tip) dok se prostorni podaci spremaju u polja geometrijskog tipa. Takvo obilježje može sadržavati jedno ili više polja geometrijskog tipa, a tablica koja sadrži taj tip naziva se tablicom obilježja. Kako bi baza podataka bila usuglašena s ovom specifikacijom, ona mora zadovoljavati jedan od uvjeta:

- korištenje numeričkog SQL tipa podatka za pohranjivanje prostornih podataka,
- korištenje binarnog SQL tipa podatka za pohranjivanje prostornih podataka ili
- implementacija tablice obilježja koja podržava i tekstualni i binarni pristup prostornom tipu podatka.

PostGIS implementacija koristi zadnje navedeni način podrške za geometrijski tip podatka.

2.6. GeoJSON

GeoJSON je otvoreni format za kodiranje različitih prostornih podataka, a nazvan je prema JSON-u (*JavaScript Object Notation* – Zapis objekata u JavaScriptu). JSON je podatkovna struktura koju koristi JavaScript za svoje objekte, tako da JavaScript “razumije” takav prikaz. Svaki je GeoJSON objekt ujedno i JSON objekt pri čemu alati koji imaju mogućnost prepoznavanja JSON sintakse, prepoznaju i GeoJSON. Nastao je kao alternativa različitim XML prikazima (GML, KML) zbog lakše čitljivosti i kompaktnosti.

GeoJSON objektima mogu se označavati različiti tipovi geometrije kao što su: točke, linije, poligoni i kombinacije istih, a može sadržavati i atributne podatke. Također, moguće je definirati i referentni koordinatni sustav. Ako referentni koordinatni sustav nije naveden, GeoJSON podrazumijeva WGS84. Jednostavni primjer GeoJSON objekta koji definira poligon:

```
{
  "type": "Polygon",
  "coordinates":
  [[
    [15.957426899291978, 45.81330743159681],
    [15.957341068603501, 45.80888028944348],
    [15.96583830676268, 45.805110958662695],
    [15.968670719482407, 45.80888028944348],
    [15.969185703613267, 45.81306813561233],
    [15.957426899291978, 45.81330743159681]
  ]]
}
```

Iz primjera je vidljivo da atribut "type" određuje tip geometrije, dok atribut "coordinates" određuje točke tog poligona.

2.7. Google Maps i Google Maps API

Google Maps je Internet servis tvrtke Google kojim je moguće pregledavati digitalne karte u Internet pregledniku. Može prikazivati satelitske snimke i prometnice, a pokriveno područje obuhvaća cijeli svijet, s različitim razinama detaljnosti. Koristi varijantu Mercatorove projekcije, a geografske koordinate su u WGS84 datumu.

Google Maps JavaScript API je programsko sučelje kojim je moguće ugrađivati Google Maps karte u web stranice. Usluga je besplatna za korištenje sve dok je web stranica koja koristi te karte javno dostupna i besplatna. Usluzi se pristupa putem JavaScripta te je moguće dodavati vlastite rasterske ili vektorske slojeve. U ovom su radu Google Maps karte poslužile kao podloga za iscrtavanje poligona i točaka.

3. Korišteni alati i programi

Alati i programi koji su korišteni prilikom stvaranja ovog rada se mogu podijeliti na tri skupine:

- poslužiteljski softver,
- softverska okruženja,
- obavezne biblioteke,
- pomoćni softver.

3.1. Poslužiteljski softver

Za stvaranje GeoDjango aplikacije potrebno je izabrati operacijski sustav na kojem će se izvršavati poslužiteljski softver. U ovom se radu koristi Ubuntu distribucija GNU/Linux operacijskog sustava verzije 10.04 (poslužiteljsko izdanje). Prva inačica Ubuntu distribucije stvorena je 2004. godine, temeljena na Debian distribuciji. Uz RedHat i Debian, Ubuntu distribucija se često koristi za potrebe posluživanja web aplikacija zbog jednostavnog paketnog sustava i lake instalacije i konfiguracije poslužiteljskog softvera.

U testnom okruženju Django aplikacija može pokrenuti svoj testni web poslužitelj, a kao bazu podataka može koristiti SQLite bazu koja se sastoji od datoteke kojoj Django aplikacija pristupa pomoću ugrađene biblioteke funkcija. Takva kombinacija nije primjerena za produkcijsko okruženje zbog slabih performansi ugrađenog web poslužitelja i nedostatak funkcija za upravljanje prostornim podacima u SQLite bazi podataka. Za produkcijsko je okruženje potrebno odabrati samostalni web poslužitelj i bazu prostornih podataka. Web poslužitelj korišten u ovom radu je Apache s `mod_wsgi` dodatkom, a baza prostornih je podataka PostgreSQL s `PostGIS` dodatkom.

3.1.1. Apache

Apache je robusni HTTP poslužitelj čija je namjena posluživanje HTML-a i ostalih tipova datoteka preko Interneta. Otvorenog je koda (*open-source*) i najpopularniji je web poslužitelj na Internetu. Moduli mu omogućuju integraciju različitih programskih jezika kao što su PHP, Python, Perl i Ruby.

Apache web poslužitelj bitna je karika preko koje korisnik (klijent) zahtjeva određene resurse (npr. web stranicu ili sliku) od udaljenog računala (poslužitelja). Apache tada prema zahtjevu klijenta isporučuje zahtijevani resurs ili vraća grešku ukoliko se ona dogodila.

Povezivanje Django aplikacije i Apache web poslužitelja može se napraviti na više načina. Najpopularniji su načini putem dodatnih modula za Apache: `mod_wsgi` i `mod_python`.

`Mod_python` je najjednostavniji način implementacije Django aplikacije s Apache web poslužiteljem. To je dodatak pomoću kojeg Apache može pokrenuti Django aplikaciju te s njom komunicirati. Nedostaci su mu veliki memorijski zahtjevi i uska povezanost s Apache procesom (ili više njih). Prilikom bilo kakve greške koja se dogodi u Django aplikaciji, Apache se “sruši” što rezultira prekidom njegova rada.

U takvim se slučajevima mogu koristiti dva ili više Apache poslužitelja gdje je jedan zadužen samo za aplikacije koje koriste `mod_wsgi`, dok drugi poslužuje ostale sadržaje. Konfiguracija je tada kompleksnija te se ta kombinacija najčešće ne koristi.

`Mod_wsgi` je dodatak sličan `mod_python`-u, ali koristi manje radne memorije i procesorskog vremena. Uz to je sigurniji i rijetko utječe na “rušenje” Apache web poslužitelja. Django aplikacija može komunicirati putem WSGI sučelja te se zbog toga u ovom radu koristi dodatak `mod_wsgi`. U praktičnom smislu, kada korisnik uputi zahtjev web poslužitelju on komunicira s Django aplikacijom pomoću WSGI sučelja. Web poslužitelj prosljeđuje zahtjev Django aplikaciji, a Django aplikacija vraća rezultat koji se isporučuje krajnjem korisniku.

Ostale datoteke (kao što su datoteke CSS-a ili JavaScripta) poslužuje direktno Apache.

3.1.2. PostgreSQL i PostGIS

PostgreSQL je *open-source* objektno-relacijski sustav baza podataka. Kao jezik za dohvat podataka koristi SQL, a može koristiti i većinu skriptnih programskih jezika kao što su Perl, Python i Ruby. Već ugrađeni jezik je

PL/pgSQL koji je sličan proceduralnom jeziku PL/SQL koji se koristi u Oracle bazama podataka.

Sadrži gotovo sve mogućnosti iz SQL:2008 standarda te je po tome najpotpunija *open-source* relacijska baza podataka. PostgreSQL ne sadrži podršku za upravljanje prostornim podacima, ali PostGIS dodatkom dobiva i tu mogućnost.

PostGIS je dodatak koji PostgreSQL bazi podataka dodaje mogućnost za upravljanje prostornim podacima te se tada može smatrati bazom prostornih podataka. PostGIS funkcije su u skladu OpenGIS specifikacijom "*Simple Features Specification for SQL*".

Kada je PostGIS instaliran u tablice PostgreSQL baze podataka, može se dodati polje tipa "geometrija" (*geometry*) koje može sadržavati točke, linije, poligone ili kolekciju navedenih objekata. Kako bi sve to imalo smisla, potrebno je koristiti PostGIS funkcije za upravljanje tim podacima. To se postiže SQL upitima koji sadrže funkcije za manipulaciju geometrijskim poljima. Postoji nekoliko tipova takvih funkcija kao što su:

- konstruktori geometrije (za stvaranje geometrijskih polja),
- funkcije za dohvaćanje svojstava ili pojedinih elemenata geometrijskih polja,
- funkcije za uređivanje geometrijskih polja,
- funkcije za konverziju geometrijskih polja u neki od izlaznih formata (WKB, WKT, GML, GeoJSON, KML, SVG, GeoHash) i
- funkcije za mjerenje i odnose između geometrijskih polja.

3.2. Softverska okruženja

Prilikom izrade ovog rada korišteno je Django okruženje koje sadrži dodatak GeoDjango kako bi se moglo manipulirati prostornim podacima.

3.2.1. Django

Django je skup alata i temelj za izradu web aplikacija u programskom jeziku Python. Sam je Django pisan u Python-u te je stvoren kako bi se što više ubrzalo i

pojednostavilo programiranje web aplikacija. Sadrži već gotove alate za standardne postupke kod razvoja kao što su:

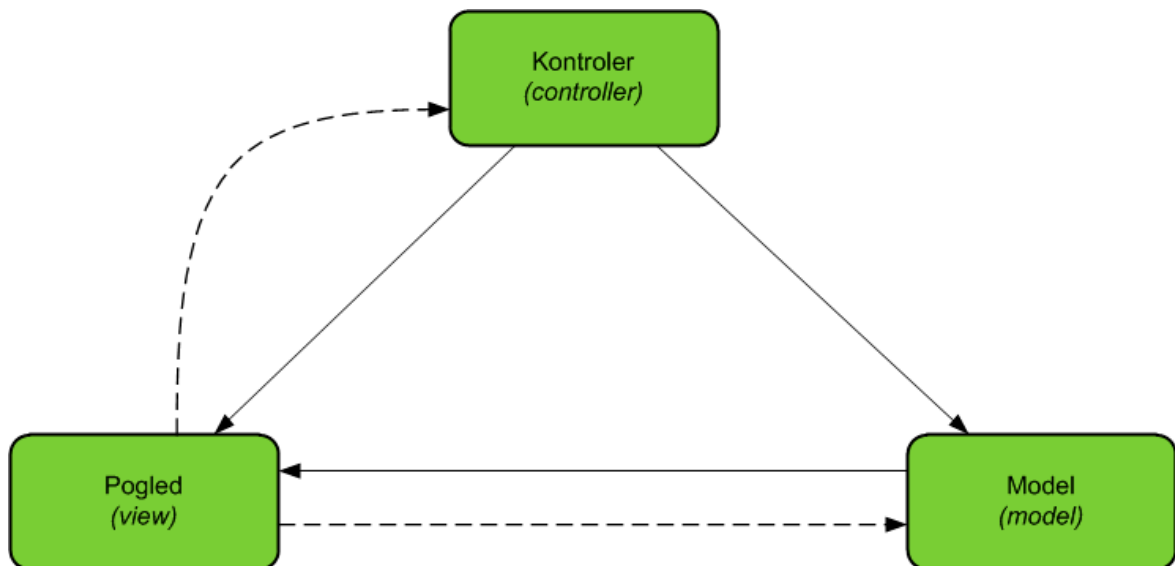
- objektno relacijski modeli podataka,
- automatsko sučelje za administraciju,
- elegantni prikaz URL-a i
- sustav predložaka.

Programiranje web aplikacija često uključuje ponavljanje istih zadataka. Takvi su zadaci

- izrada administracijskog sučelja,
- autentifikacija korisnika,
- korisnička prava,
- izrada formulara za unos podataka,
- validacija unesenih podataka i
- internacionalizacija web aplikacija.

U drugim programskim jezicima i okruženjima takvi zadaci predstavljaju mukotrpan posao. U Django okruženju navedeni su postupci pojednostavljeni i svode se na pisanje nekoliko linija Python koda. Objektno-relacijskim modelom podataka izbjegava se pisanje SQL upita koje Django automatski generira.

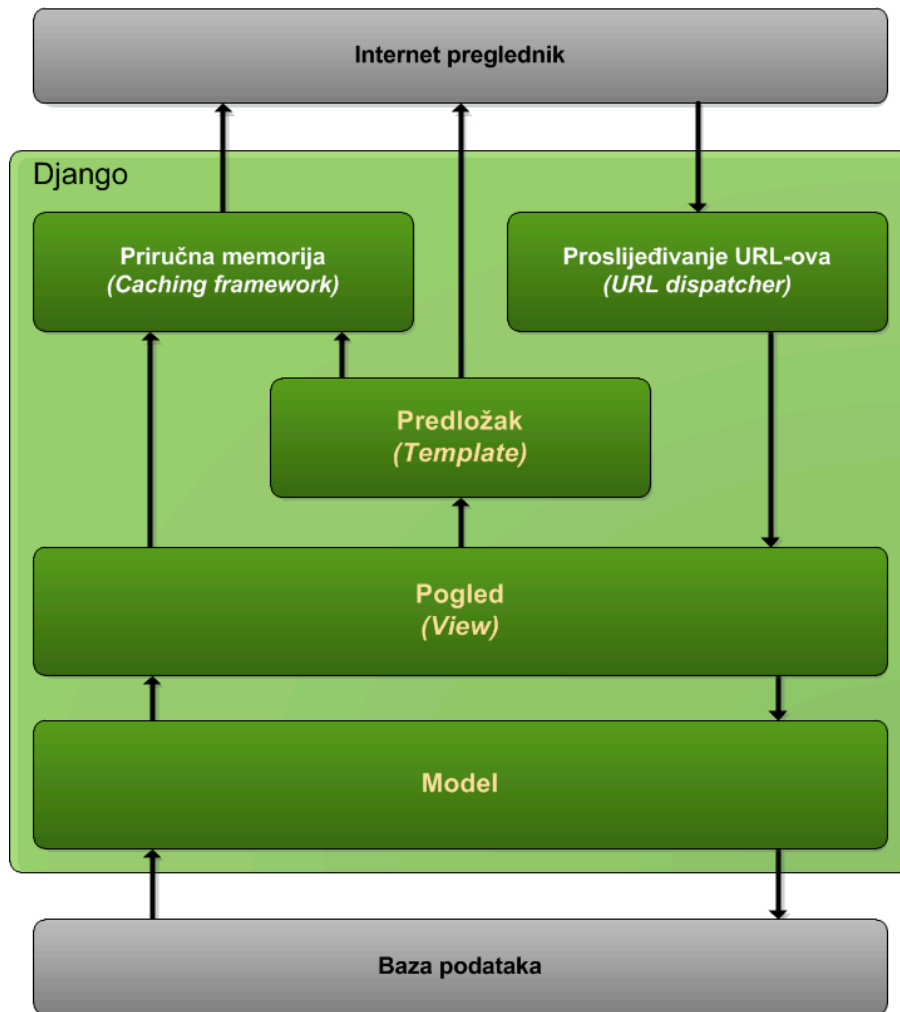
Kod kompleksnijih web aplikacija potrebno je izolirati logiku aplikacije od prikaznog dijela aplikacije. Django koristi MVC (*model-view-controller*) arhitekturu kod koje model predstavlja model podataka u bazi podataka, pregled (*view*) predstavlja prikazni dio aplikacije (ono što vidi krajnji korisnik) dok je kontroler (*controller*) zadužen za samu logiku aplikacije i povezivanje modela i pregleda (Slika 2).



Slika 2. Koncept modela, pogleda i kontrolera u MVC-u

Pune linije označavaju izravnu vezu između navedenih elemenata za razmjenu podataka, dok iscrtkana linija označava neizravnu vezu kod koje neki drugi objekt vrši tu funkciju.

Django koristi drugačije nazive elemenata MVC-a, što je pomalo i zbunjujuće. Django *view* zapravo predstavlja kontroler, a *template* (predložak) predstavlja *view* iz MVC-a, tako da se implementacija MVC-a u Django može nazvati MTV (*model-template-view*) (Slika 3). Kada korisnik u svom Internet pregledniku upiše adresu Django aplikacije, tu adresu prvo prihvaća modul za prosljeđivanje URL-ova. Taj se modul sastoji od regularnih izraza prema kojima se odabire pogled koji će biti aktiviran. Pogled komunicira s modelima koji su zaduženi za komunikaciju s bazom podataka. Model vraća pogledu podatke iz baze podataka koji se zatim šalju predlošku (najčešće HTML). Predložak generira HTML kod prema dobivenim podacima te se oni šalju natrag korisniku. Ako se koristi priručna memorija, ili ako nisu potrebni podaci iz baze podataka, neki od ovih koraka mogu se i zaobići.



Slika 3. Dijagram toka u Django aplikaciji

U Django aplikaciji model predstavlja tablicu u bazi podataka. Jednostavan model može izgledati ovako:

```

from django.db import models

class Osoba(models.Model):
    ime = models.CharField(max_length=50)
    prezime = models.CharField(max_length=50)
    starost = models.IntegerField()

```

Načini i mjesta gdje se zapisuju ti modeli bit će objašnjeni u kasnijim poglavljima. Klasa “Osoba” sadrži attribute “ime”, “prezime” i “starost”. Polja imena i prezimena tekstualnog su tipa i maksimalne duljine do 50 znakova, dok je starost numeričkog tipa. Django sa svojim alatom za automatsku sinkronizaciju s bazom podataka stvara tablicu u bazi s navedenim atributima i tu strukturu nije potrebno dodatno podešavati.

Novi se redak u takvoj tablici dodaje stvaranjem objekta klase "Osoba", pridruživanjem vrijednosti pripadajućim atributima i pozivanjem funkcije objekta "save()":

```
>> nova_osoba = Osoba()
>> nova_osoba.ime = "Tomislav"
>> nova_osoba.prezime = "Horvat"
>> nova_osoba.starost = 33
>> nova_osoba.save()
```

Za postavljanje upita i traženja određenih redaka koristi se upravitelj (*Manager*) modelima, a pristupa se pomoću objekta "*objects*" zadanog modela. Upravitelj izravno komunicira s bazom podataka šaljući joj SQL upite koje on generira prema zadanim uvjetima.

```
>> trazena_osoba = Osoba.objects.get(ime__exact="Tomislav")
>> print trazena_osoba.ime
Tomislav
>> print trazena_osoba.prezime
Horvat
>> print trazena_osoba.starost
33
```

U gornjem je primjeru prikazan upit kojim se traže osobe s imenom "Tomislav". Takav bi upit u SQL-u izgledao ovako:

```
SELECT * FROM osoba WHERE osoba.ime = 'Tomislav'
```

3.2.2. GeoDjango

GeoDjango je dodatak za Django okruženje kojim se dodaje mogućnost upravljanja prostornim podacima. Razvoj GeoDjanga počeo je u jesen 2006. godine neovisno o Djangu, a 2008. godine postao je dio Django okruženja (URL 11). GeoDjango dodatak omogućuje brzo i jednostavno stvaranje web aplikacija koje koriste prostorne podatke. U tom se slučaju mora koristiti prostorna baza podataka, a GeoDjango podržava sljedeće:

- PostgreSQL s PostGIS dodatkom
- MySQL
- Oracle
- SQLite sa SpatialLite dodatkom

U ovom se radu koristi PostgreSQL baza s PostGIS dodatkom jer ima najpotpuniju podršku za prostorne podatke i besplatna je.

Kako bi Django aplikacija mogla koristiti i manipulirati prostornim podacima, GeoDjango dodaje geometrijske tipove podataka koji se podudaraju s *OpenGIS Simple Features* specifikacijom:

- *GeometryField* – polje koje sadrži proizvoljnu geometriju
- *PointField* – polje koje sadrži točku
- *LineStringField* – polje koje sadrži liniju
- *MultiPointField* – polje koje sadrži više točaka
- *MultiLineStringField* – polje koje sadrži više linija
- *MultiPolygonField* – polje koje sadrži više poligona
- *GeometryCollectionField* – polje koje sadrži različite tipove geometrija

Django model koji sadrži geometrije izgleda ovako:

```
from django.contrib.gis.db import models
class Drzava(models.Model):
    naziv = models.CharField(max_length=100)
    geom = models.MultiPolygonField()
    objects = models.GeoManager()
```

Geometrijsko se polje definira tipom geometrije koji u ovom slučaju može sadržavati više poligona (*MultiPolygonField*). Za razliku od modela koji ne sadrži prostorne podatke, u navedenom se modelu mora stvoriti poseban upravitelj “GeoManager” koji omogućava prostorne upite. Postavljanje upita opisano je u kasnijim poglavljima.

GeoDjango uključuje module za jednostavni rad s udaljenostima i površinama te konverziju između mjernih jedinica. To omogućavaju klase *Distance* i *Area*:

```
>>> from django.contrib.gis.measure import Distance
>>> udaljenost1 = Distance(km=7)
>>> udaljenost1.mi
4.3495983456613381
>>> udaljenost2 = Distance(km=5)
>>> udaljenost1 + udaljenost2
Distance(km=12.0)
>>> udaljenost1 * udaljenost2
Area(sq_km=35.0)
```

U prvoj se liniji koda dohvaća klasa “Distance” pomoću koje se mogu stvoriti objekti tog tipa. Argumenti su kod stvaranja mjerna jedinica i vrijednost. Stvoreni su objekti “udaljenost1” s vrijednošću od 7 km i “udaljenost2” s vrijednošću od 5

km. Prva je udaljenost pretvorena u milje, a množenjem dviju udaljenosti dobiva se površina s jasno označenom mjernom jedinicom (kvadratni kilometar).

3.2.3. Prostorne biblioteke

GeoDjango upravlja prostornim podacima pomoću biblioteka za upravljanje prostornim podacima. Kada se spaja s PostgreSQL bazom podataka i PostGIS dodatkom, obavezne su GEOS i PROJ.4.

GEOS (*Geometry Engine – Open Source*) je biblioteka pisana u C++ programskom jeziku. Sadrži funkcije i prostorne operatore temeljene na OpenGIS specifikaciji “Simple Features for SQL”. Ta je biblioteka u GeoDjangu osnova za stvaranje geometrija i konverziju između različitih tipova unosa.

PROJ.4 je biblioteka za konverziju prostornih podataka između različitih referentnih koordinatnih sustava.

4. Korištenje mogućnosti GeoDjanga

Jednom kad je prostorna baza podataka podešena i stvoren Django projekt (što je detaljno opisano u sljedećem poglavlju), mogu se koristiti sve mogućnosti GeoDjanga:

- unos prostornih podataka,
- manipulacija prostornim podacima,
- prostorni upiti.

Django posjeduje posebnu školjku (*shell*) za interaktivno programiranje u Pythonu koje će se koristiti u ovom poglavlju. Koristi se za brze promjene i analizu Django aplikacije, a u ovom slučaju za demonstraciju mogućnosti GeoDjango dodatka.

Prostorni podaci mogu biti u bilo kojem koordinatnom sustavu koji postoji u bazi PROJ.4 biblioteke. Ako ne postoji, moguće ga je definirati parametrima.

4.1. Unos podataka

Za unos podataka potrebno je imati izvor prostornih podataka. Za njih će kasnije biti potrebno stvoriti model podatka s atributnim i geometrijskim tipovima podatka. U ovom je primjeru preuzet ESRI Shapefile koji sadrži granice država svijeta (URL 12) naziva “TM_WORLD_BORDERS-0.3.shp”.

4.1.1. Ispitivanje datoteke za unos

Pokretanjem interaktivne školjke mogu se ispitati svojstva te datoteke:

```
$ python manage.py shell
...
>>> from django.contrib.gis.gdal import DataSource
>>> izvor = DataSource('shp/TM_WORLD_BORDERS-0.3.shp')
>>> print izvor
shp/TM_WORLD_BORDERS-0.3.shp (ESRI Shapefile)
>>> print izvor.layer_count
1
```

ESRI Shapefile datoteka se učitava pomoću “DataSource” klase iz GDAL biblioteke. Potkomponenta GDAL biblioteke koja je zadužena za čitanje različitih vektorskih formata (npr. već navedeni ESRI Shapefile, ESRI ArcSDE, GML, KML, GeoJSON...) zapravo je OGR (URL 13). Nakon učitavanja datoteke mogu se

ispitati njezina osnovna svojstva. Samim ispisom novostvorenog objekta prikazuje se njezin položaj u datotečnom sustavu (nalazi se u direktoriju “shp”), njezin naziv i tip datoteke (što je u ovom slučaju ESRI Shapefile). Atribut “layer_count” daje do znanja da taj Shapefile ima samo jedan sloj. Objekt “izvor” ponaša se kao Python lista te se na taj način može, s indeksom 0, dohvatiti navedeni sloj:

```
>>> sloj = izvor[0]
>>> print sloj.geom_type # tip geometrije
Polygon

>>> print len(sloj) # broj obilježja
246

>>> print sloj.srs # referentni koordinatni sustav
GEOGCS["GCS_WGS_1984",
  DATUM["WGS_1984",
    SPHEROID["WGS_1984",6378137.0,298.257223563]],
  PRIMEM["Greenwich",0.0],
  UNIT["Degree",0.0174532925199433]]
>>> sloj.srs.proj4
'+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs '

>>> print sloj.fields # atributna polja
['FIPS', 'ISO2', 'ISO3', 'UN', 'NAME', 'AREA', 'POP2005',
 'REGION', 'SUBREGION', 'LON', 'LAT']

>>> [polje.__name__ for polje in sloj.field_types] # tipovi
['OFTString', 'OFTString', 'OFTString', 'OFTInteger',
 'OFTString', 'OFTInteger', 'OFTInteger', 'OFTInteger',
 'OFTInteger', 'OFTReal', 'OFTReal']
```

Kada je sloj dohvaćen, može se ispitati koji tip geometrije sadrži. U ovom je primjeru vidljivo da je to poligon. Funkcija “len” u Pythonu dohvaća broj elemenata liste, a pošto je sloj sastavljen od liste obilježja, ona vraća broj obilježja, u ovom slučaju 246. Vrlo je bitno provjeriti i referenti koordinatni sustav koji se nalazi u “srs” atributu sloja kako bi se taj isti mogao primijeniti u modelu podatka. Taj atribut može dohvatiti i PROJ.4 prikaz.

Osim geometrije, sloj sadrži i atributna polja kao što su oznake i šifre država po raznim standardima, ime države, površina, broj stanovnika i slično. Također, može se provjeriti i tip atributnih podataka. U ovom primjeru “FIPS”, “ISO2”, “ISO3” i “NAME” sadrže znakovne nizove, “UN”, “AREA”, “POP2005”, “REGION” i “SUBREGION” cijele brojeve, a “LON” i “LAT” realne brojeve. Nastavlja se s dodatnom provjerom obilježja i podacima koje sadrži:

```

>>> for obiljezje in sloj:
...     if obiljezje.get('NAME') == 'Croatia':
...         hrvatska = obiljezje
...
>>> hrvatska.get('NAME')
'Croatia'

>>> hrvatska.get('POP2005')
455149

>>> hrvatska.get('ISO2')
'HR'

>>> hrvatska.get('ISO3')
'HRV'

>>> for obiljezje in sloj:
...     if obiljezje.get('NAME') == 'Germany':
...         njemacka = obiljezje
...
>>> njemacka.get('POP2005')
82652369

```

U gornjem je primjeru dohvaćeno jedno obilježje kojem je atribut "NAME" jednak "Croatia", odnosno, dohvaćeno je obilježje Hrvatske. Provjerom nekih atributnih podataka vidi se da je broj stanovnika netočan te je vjerojatno u procesu izrade te Shapefile datoteke izostavljena zadnja znamenka broja stanovnika. Dodatnom provjerom neke druge države (Njemačka) vidljivo je da su podaci o broju stanovnika ostalih država točni.

Obilježja sadrže i geometrijsko polje kojem se pristupa preko atributa "geom":

```

>>> hrvatska.geom
<django.contrib.gis.gdal.geometries.MultiPolygon object at
0x2546a50>

>>> hrvatska.geom.json[0:100]
'{"type": "MultiPolygon", "coordinates": [ [ [ [ 16.903889,
42.726105 ], [ 16.894444, 42.724716 ], [ '

>>> hrvatska.geom.kml[0:100]
'<MultiGeometry><Polygon><outerBoundaryIs><LinearRing><coordin
ates>16.903889000000049,42.726105000000'

>>> hrvatska.geom.wkt[0:100]
'MULTIPOLYGON (((16.903889000000049
42.726105000000132,16.894443999999964
42.724716000000114,16.83583'

```

U ovom se primjeru vidi kako je tip geometrije zapravo više poligona (*MultiPolygon*), što nije bilo vidljivo iz tipa geometrije koje je prikazivao sam sloj. GEOS biblioteka automatski je prepoznala više poligona te ga je tako i interpretirala. Geometrija objekta može se prikazati u GeoJSON, KML i WKT tekstualnim formatima.

4.1.2. Stvaranje modela

Nakon što je datoteka s granicama država proučena, može se stvoriti model podatka kako bi se podaci iz ESRI Shapefile datoteke mogli ispravno unijeti. Za stvaranje modela potrebno je napraviti Django aplikaciju. Django aplikacija je poseban modul koji je namijenjen određenim zadacima unutar kojeg se definiraju modeli podataka. Django projekt može imati više aplikacija koje bi trebale biti neovisne jedna o drugoj (iako međusobno mogu komunicirati). Za potrebe ovog primjera stvorena je aplikacija “granice”:

```
$ python manage.py startapp granice
$ cd granice
$ ls
__init__.py  models.py  tests.py  views.py
```

Naredba za stvaranje aplikacije pokreće se u direktoriju u kojem je stvoren Django projekt. Nastao je direktorij s nazivom aplikacije (“granice”) te unutar njega pripadajuće datoteke. One su zapravo prazne i služe samo kao predložak. U ovom je slučaju potrebno urediti datoteku “models.py”:

```
from django.contrib.gis.db import models

class Granica(models.Model):
    naziv = models.CharField(max_length=50)
    površina = models.IntegerField()
    broj_stanovnika = models.IntegerField()

    geom = models.MultiPolygonField(srid=4326)

    objects = models.GeoManager()

    def __unicode__(self):
        return self.naziv
```

Za primjer su postavljena tri neprostorna atributa (“naziv”, “površina” i “broj_stanovnika”) i jedan prostorni “geom” koji će predstavljati poligone granica država dohvaćenih iz ESRI Shapefile datoteke. Geometrijskom polju “geom” dodana je SRID vrijednost 4326 (WGS84) koja se podudara s podacima iz ESRI Shapefile datoteke. Metoda “__unicode__” služi kao pomoć kod testiranja u interaktivnoj školjci, a kasnije i u administracijskom sučelju – određuje vrijednost kojom će objekt biti prezentiran kako bi se lako mogao identificirati. U ovom se slučaju ispisuje “naziv”. Za stvaranje iste tablice u bazi podataka potrebno je dodati aplikaciju u “settings.py” (glavnu datoteku Django projekta za postavke) i pokrenuti sinkronizaciju baze podataka:

```
$ python manage.py syncdb
Creating table granice_granica
Installing index for granice.Granica model
No fixtures found.
```

Django je prema modelu podatka “Granica” stvorio tablicu u bazi “granice_granica” (prvi je dio naziv aplikacije, a drugi naziv modela), a SQL kod koji se izvršio može se vidjeti preko alata “manage.py”:

```
$ python manage.py sqlall granice
BEGIN;
CREATE TABLE "granice_granica" (
  "id" serial NOT NULL PRIMARY KEY,
  "naziv" varchar(50) NOT NULL,
  "povrsina" integer NOT NULL,
  "broj_stanovnika" integer NOT NULL
)
;
SELECT AddGeometryColumn('granice_granica', 'geom', 4326,
'MULTIPOLYGON', 2);
ALTER TABLE "granice_granica" ALTER "geom" SET NOT NULL;
CREATE INDEX "granice_granica_geom_id" ON "granice_granica" USING GIST
( "geom" GIST_GEOMETRY_OPS );
COMMIT;
```

Iako u modelu nije naveden, atribut “id” automatski se stvara kao primarni ključ koji je brojanog tipa i povećava se za 1 svakim novim dodavanjem retka. U podebljanoj je liniji način na koji se stvara geometrijsko polje u PostgreSQL bazi podataka s PostGIS dodatkom. Pisanje SQL koda u Django nije obavezno, a ovdje se koristi samo kao demonstracija pozadinskog rada koji Django samostalno obavlja.

Nakon što su stvoreni model podatka i tablica u bazi, model se može testirati u interaktivnom *shellu*:

```
$ python manage.py shell
(InteractiveConsole)
>>> from granice.models import Granica
>>> nova_granica = Granica()
>>> nova_granica.naziv = 'Nepostojeca drzava'
>>> nova_granica.povrsina = 1000
>>> nova_granica.broj_stanovnika = 10000
>>> from django.contrib.gis.geos import GEOSGeometry
>>> multi_poly = GEOSGeometry('MULTIPOLYGON(((15.957770222045884
45.80050365244182, 16.00351797900389 45.80044381478984,
16.00257384143065 45.815461049188364, 15.959401005126939
45.81504229673487, 15.957770222045884 45.80050365244182))))')
>>> nova_granica.save()
>>> print nova_granica.id
1
>>> nova_granica.delete()
```

Iz aplikacije “granice” pozvan je model “Granica” prema kojem se stvara novi objekt “nova_granica”. Pridružene su mu probne vrijednosti za potrebe testiranja

tog modela. Postupci za punjenje podataka iz ESRI Shapefile datoteke još nisu dovršeni pa je geometrija od više poligona stvorena “ručno” pomoću “GEOSGeometry” klase koja prepoznaje više formata, a u ovom je primjeru unesen poligon u WKT formatu. Objekt je spremljen metodom “save()” i vidljiv je u bazi podataka (Slika 4). Geometrijsko polje je tamo prikazano u WKB formatu, odnosno binarnom WKB formatu. Naposljetku je taj objekt obrisao metodom “delete()” jer nije više potreban, a ujedno je obrisao i redak u bazi podataka.

The screenshot shows the phpPgAdmin interface. At the top, there are navigation links for phpPgAdmin, PostgreSQL, geodb, public, and granice_granica. Below that is a 'Browse' button. The main area displays a table with the following columns: Actions, id, naziv, površina, broj_stanovnika, and geom. The first row of data contains the following values: Edit, Delete, 1, Nepostojeća država, 1000, and a long hexadecimal string representing the geometry.

Actions	id	naziv	površina	broj_stanovnika	geom
Edit Delete	1	Nepostojeća država	1000	100000106000020E61000000100000001030000000100000005000...	

Slika 4. Prikaz retka u tablici baze podataka pomoću phpPgAdmin sučelja

4.1.3. Uvoz podataka

Model je ispravno definiran i sada je moguće unositi podatke granica država iz ESRI Shapefile datoteke. Za to je potrebna klasa “LayerMapping” iz GeoDjango dodatka kojoj se daje ime ESRI Shapefile datoteke, model podatka koji se puni i veza između atributa iz Shapefile datoteke i modela. Stvara se modul imena “unos.py” koji se sprema u direktorij aplikacije “granice”:

```
import os
from django.contrib.gis.utils import LayerMapping
from models import Granica

atributi = {
    'naziv' : 'NAME',
    'površina' : 'AREA',
    'broj_stanovnika' : 'POP2005',
    'geom' : 'MULTIPOLYGON',
}

granice_shp = 'granice/shp/TM_WORLD_BORDERS-0.3.shp'

def pokreni(verbose=True):
    lm = LayerMapping(Granica, granice_shp, atributi,
                     transform=False, encoding='iso-8859-1')
    lm.save(strict=True, verbose=verbose)
```

Iako ESRI Shapefile ima više atributnih podataka, u ovom se slučaju u obzir uzimaju samo “NAME”, “AREA”, “POP2005” i geometrija. Jasno je naznačeno da će atribut “NAME” biti spremljen u atribut “naziv” modela “Granica”, a tako i ostali atributi. Navedena je i relativna putanja Shapefile datoteke u odnosu na direktorij u kojem se nalazi Django projekt. Funkcija “pokreni” zapravo pokreće unos

podataka. Stvara se objekt "Im" prema klasi "LayerMapping" kojoj je zadano popunjavanje modela "Granica" iz datoteke "granice_shp" gdje u obzir uzima attribute "atributi". Transformacija koordinata nije potrebna jer je Shapefile zadan kao WGS84, isto kao model "Granica". Metodom "save()" pokreće se sam unos podataka u model, a naposljetku i u bazu podataka. Pokretanje se izvršava u interaktivnoj školjci:

```
$ python manage.py shell
(InteractiveConsole)
>>> from granice import unos
>>> unos.pokreni()
Saved: Antigua and Barbuda
Saved: Algeria
Saved: Azerbaijan
Saved: Albania
Saved: Armenia
.
.
.
Saved: Jersey
Saved: South Georgia South Sandwich Islands
Saved: Taiwan
>>>
```

Nakon pokretanja funkcije "pokreni()" ispisuje se naziv svake države koja je spremljena te je time, ako nije bilo greške, unos u bazu podataka završen. To se može provjeriti u phpPgAdmin sučelju (Slika 5).

phpPgAdmin: PostgreSQL? geodb? public? granice_granica?

Browse

1 2 3 4 5 6 7 8 9 Next > Last >>

Actions	id	naziv	povrsina	broj_stanovnika	geom
Edit Delete	2	Antigua and Barbuda	44	83039	0106000020E6100000200000001030000000100000017000...
Edit Delete	3	Algeria	238174	32854159	0106000020E6100000010000000103000000001000000D9040...
Edit Delete	4	Azerbaijan	8260	8352021	0106000020E6100000040000000103000000010000006C000...
Edit Delete	5	Albania	2740	3153731	0106000020E61000000100000001030000000100000051010...
Edit Delete	6	Armenia	2820	3017661	0106000020E6100000020000000103000000010000000C000...
Edit Delete	7	Angola	124670	16095214	0106000020E61000000300000001030000000100000020000...
Edit Delete	8	American Samoa	20	64051	0106000020E61000000500000001030000000100000009000...
Edit Delete	9	Argentina	273669	38747148	0106000020E6100000060000000103000000010000000C000...
Edit Delete	10	Australia	768230	20310208	0106000020E610000006100000001030000000100000019000...
Edit Delete	11	Bahrain	71	724788	0106000020E6100000060000000103000000010000000A000...
Edit Delete	12	Barbados	43	291933	0106000020E610000001000000010300000001000000031000...
Edit Delete	13	Bermuda	5	64174	0106000020E6100000030000000103000000010000000F000...
Edit Delete	14	Bahamas	1001	323295	0106000020E610000001C000000010300000001000000034000...
Edit Delete	15	Bangladesh	13017	15328112	0106000020E61000000230000000103000000010000001B000...
Edit Delete	16	Belize	2281	275546	0106000020E61000000600000001030000000100000011000...
Edit Delete	17	Bosnia and Herzegovina	5120	3915238	0106000020E61000000100000001030000000100000008F010...
Edit Delete	18	Bolivia	108438	9182015	0106000020E61000000100000001030000000100000002040...
Edit Delete	19	Burma	65755	47967266	0106000020E610000002C00000001030000000100000014000...
Edit Delete	20	Benin	11062	8490301	0106000020E61000000100000001030000000100000007F010...
Edit Delete	21	Solomon Islands	2799	472419	0106000020E61000000330000000103000000010000001B000...
Edit Delete	22	Brazil	845942	186830759	0106000020E610000003F00000001030000000100000002A000...
Edit Delete	23	Bulgaria	11063	7744591	0106000020E610000001000000010300000001000000034020...
Edit Delete	24	Brunei Darussalam	527	373831	0106000020E610000002000000010300000001000000031000...
Edit Delete	25	Canada	909351	32270507	0106000020E61000000DE1000000103000000010000000E000...
Edit Delete	26	Cambodia	17652	13955507	0106000020E610000005000000010300000001000000012000...
Edit Delete	27	Sri Lanka	6463	19120763	0106000020E61000000600000001030000000100000000C000...
Edit Delete	28	Congo	34150	3609851	0106000020E610000001000000010300000001000000020040...
Edit Delete	29	Democratic Republic of the Congo	226705	58740547	0106000020E610000002000000010300000001000000001C000...
Edit Delete	30	Burundi	2568	7858791	0106000020E61000000100000001030000000100000001010...
Edit Delete	31	China	932743	1312978855	0106000020E610000002200000001030000000100000000B000...

30 row(s)

1 2 3 4 5 6 7 8 9 Next > Last >>

Slika 5. Prikaz redaka granica država u phpPgAdmin sučelju

4.2. Prostorni upiti

Nakon što su podaci o granicama država upisani u bazu, nad njima se mogu provoditi prostorni upiti. Testiranje se može izvoditi u interaktivnoj školjci, a takvi se upiti mogu provoditi u bilo kojem dijelu Django aplikacije.

Upiti se provode pomoću “objects” objekta koji se nalaze u klasi modela. Ima nekoliko metoda koje vraćaju objekt klase “QuerySet”.

```
>>> from granice.models import Granica
>>> Granica.objects.all()
[<Granica: Antigua and Barbuda>, <Granica: Algeria>, <Granica:
Azerbaijan>, <Granica: Albania>, <Granica: Armenia>, <Granica:
Angola>, <Granica: American Samoa>, <Granica: Argentina>, <Granica:
Australia>, <Granica: Bahrain>, <Granica: Barbados>, <Granica:
Bermuda>, <Granica: Bahamas>, <Granica: Bangladesh>, <Granica:
Belize>, <Granica: Bosnia and Herzegovina>, <Granica: Bolivia>,
<Granica: Burma>, <Granica: Benin>, <Granica: Solomon Islands>,
'...(remaining elements truncated)...']

>>> type(Granica.objects.all())
<class 'django.contrib.gis.db.models.query.GeoQuerySet'>

>>> Granica.objects.all()[3]
<Granica: Albania>

>>> Granica.objects.filter(naziv__exact='Germany')
[<Granica: Germany>]

>>> Granica.objects.filter(naziv__startswith='Ge')
[<Granica: Georgia>, <Granica: Germany>]

>>> Granica.objects.filter(naziv__startswith='Ge')
.exclude(naziv__exact='Georgia')
[<Granica: Germany>]

>>> Granica.objects.all().count()
246

>>> Granica.objects.filter(broj_stanovnika__gt = 1000000000)
[<Granica: China>, <Granica: India>]

>>> Granica.objects.get(naziv__exact='Slovenia')
<Granica: Slovenia>
```

U navedenom se primjeru provode upiti u modelu “Granica” koji nisu prostorne prirode, ali pokazuju način na koji rade. Koriste se metode “all”, “filter”, “exclude”, “count” i “get”. Metoda “all” vraća sve objekte modela “Granica”, odnosno sve redove iz baze podataka. Oni se ponašaju kao lista i može im se pristupiti putem indeksa. Metodom “filter” filtriraju se rezultati prema zadanim parametrima, a može ih biti i više. U ovom se primjeru dohvaćaju sve granice gdje je naziv jednak “Germany”. Metode se mogu i ulančavati kao što je pokazano u gornjem primjeru gdje se filtriraju sve granice kojima naziv počinje s “Ge”, ali isključuju (metoda “exclude”) one granice kojima je naziv jednak “Georgia”. Metoda “count” može se

iskoristiti nakon svakog filtriranja te pokazuje koliko je na kraju filtriranih rezultata. Metoda “get” koristi se u slučajevima kada se dohvaća samo jedan objekt s zadanim parametrima, ali ako prema zadanim parametrima postoji više rezultata, metoda prijavljuje grešku. Sve metode i načini na koje se provode upiti mogu se vidjeti u dokumentaciji Djanga (URL 14).

Metoda “filter” prepoznaje uvjete upita prema nazivu parametra koji je s dvije donje crte (__) podijeljen na naziv atributa i naziv operatora koji se koristi (npr. “naziv__exact” znači da se traži naziv jednak vrijednosti koja je dodijeljena tom parametru, a “broj_stanovnika__gt” – *greater than* – znači da se traže elementi koji imaju veći broj stanovnika od vrijednosti tog parametra). Na isti se način filter koristi u prostornim upitima. GeoDjango može koristiti sljedeće:

bbcontains	distance_gte	touches
bboverlaps	distance_lt	within
contained	distance_lte	left
contains	dwithin	right
contains_properly	equals	overlaps_left
coveredby	exact	overlaps_right
covers	intersects	overlaps_above
crosses	overlaps	overlaps_below
disjoint	relate	strictly_above
distance_gt	same_as	strictly_below

Tablica 3. Prostorni operatori koje koristi GeoDjango

Navedeni operatori opisani su u GeoDjango dokumentaciji (URL 15). Kako bi se što zornije pokazali različiti primjeri prostornih upita, potrebno je napraviti još jedan model u aplikaciji “granice”:

```
class Grad(models.Model):
    naziv = models.CharField(max_length=50)
    geom = models.PointField(srid=4326)

    objects = models.GeoManager()

    def __unicode__(self):
        return self.naziv
```

Model "Grad" sadrži dva atributa od kojeg je jedan prostorni ("geom") i predstavlja točku. Nakon sinkronizacije s bazom podataka, nekoliko je gradova moguće upisati putem interaktivnog *shell*:

```
>>> from granice.models import Grad
>>> from django.contrib.gis.geos import Point
>>> Grad(naziv="Zagreb", geom=Point(15.978514, 45.814912)).save()
>>> Grad(naziv="Ljubljana", geom=Point(14.505965, 46.051426)).save()
>>> Grad(naziv="Budimpešta", geom=Point(19.040758, 47.498406)).save()
>>> Grad(naziv="Maribor", geom=Point(15.645982, 46.557399)).save()
>>> Grad(naziv="Samobor", geom=Point(15.709175, 45.801854)).save()
>>> Grad(naziv="Varaždin", geom=Point(16.333921, 46.309906)).save()
```

Koordinate gradova dobivene su jednostavnim pretraživanjem u aplikaciji Google Maps. Za razliku od prethodnih primjera stvaranja novih objekata, u ovom se koristi skraćeni način stvaranja gdje se vrijednosti odmah prosljeđuju klasi i metodom "save" spremaju u bazu podataka. Klasa "Point" iz biblioteke GEOS olakšava stvaranje točaka.

Slijede primjeri prostornih upita u interaktivnom *shell*:

```
>>> from granice.models import Grad, Granica
>>> zg = Grad.objects.get(naziv__exact='Zagreb')
>>> zg
<Grad: Zagreb>
>>> Granica.objects.filter(geom__contains=zg.geom)
[<Granica: Croatia>]
>>> hr = Granica.objects.get(naziv__exact='Croatia')
>>> hr
<Granica: Croatia>
>>> lj = Grad.objects.get(naziv__exact='Ljubljana')
>>> Granica.objects.filter(geom__contains=lj.geom)
[<Granica: Slovenia>]

>>> Grad.objects.filter(geom__within=hr.geom)
[<Grad: Zagreb>, <Grad: Samobor>, <Grad: Varaždin>]

>>> lj = Grad.objects.get(naziv__exact='Ljubljana')
>>> Grad.objects.filter(geom__left=zg.geom)
[<Grad: Ljubljana>, <Grad: Maribor>, <Grad: Samobor>]

>>> Grad.objects.filter(geom__right=zg.geom)
[<Grad: Budimpešta>, <Grad: Varaždin>]
>>> Grad.objects.filter(geom__strictly_above=zg.geom)
[<Grad: Ljubljana>, <Grad: Budimpešta>, <Grad: Maribor>, <Grad:
Varaždin>]

>>> Grad.objects.filter(geom__strictly_below=zg.geom)
[<Grad: Samobor>]
```

Grad "Zagreb" dohvaća se u objekt "zg" te se prema uvjetu "contains" filtriraju sve granice, odnosno države, koje sadrže geometriju tog objekta, odnosno točku. Kod tog je upita očito da će jedini rezultat biti granica Hrvatske. Na isti se način

pomoću grada Ljubljane može dobiti i granica Slovenije. Objekt "hr" sadrži geometriju hrvatske granice i pomoću upita "within" nad objektima gradova dohvaćaju se samo oni koji su unutar tih granica (definiranih poliginom ili više njih). Uvjetima "left", "right", "strictly_above" i "strictly_below" mogu se dobiti gradovi koji su lijevo, desno, iznad ili ispod grada Zagreba.

Uvjeti mogu sadržavati i udaljenosti pomoću klase "Distance":

```
>>> Grad.objects.filter(geom__distance_lt=(zg.geom, Distance(km=100)))
[<Grad: Zagreb>, <Grad: Maribor>, <Grad: Samobor>, <Grad: Varaždin>]

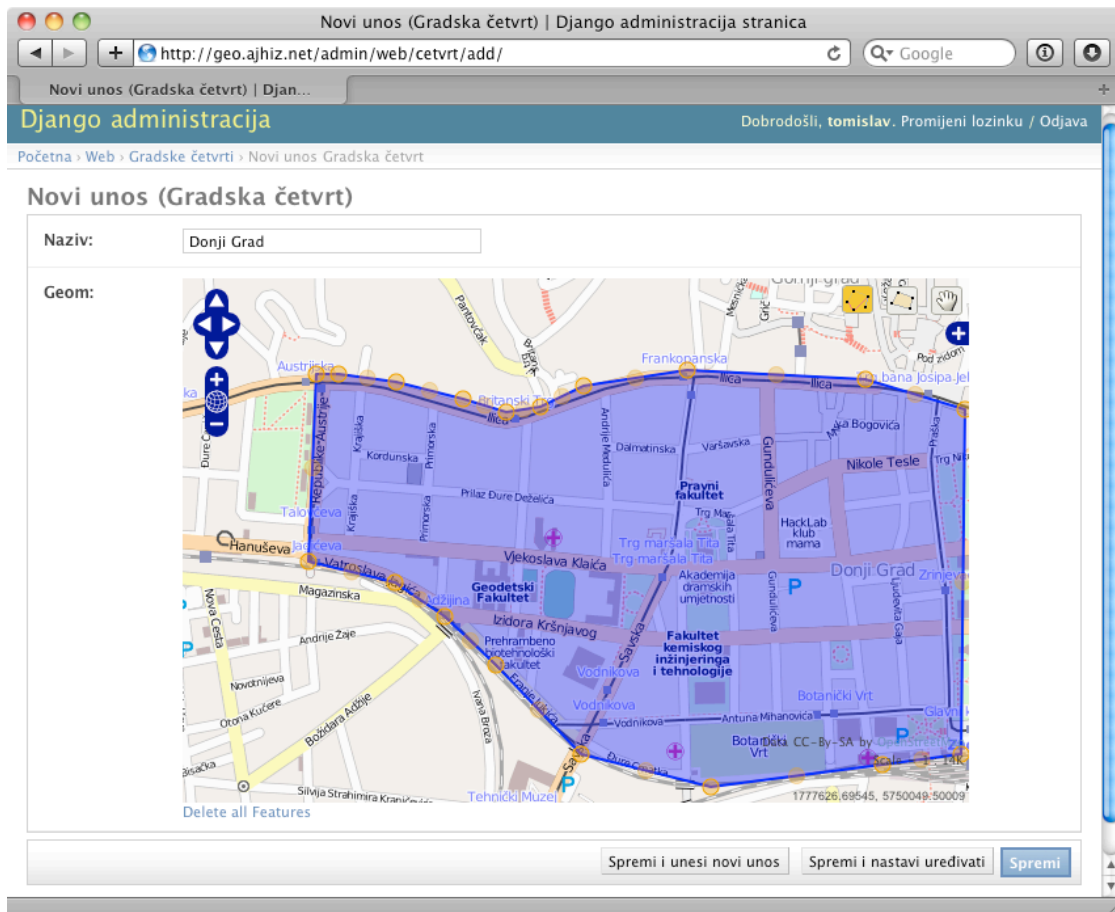
>>> Grad.objects.filter(geom__distance_lt=(zg.geom,
Distance(km=100))).exclude(id=zg.id)
[<Grad: Maribor>, <Grad: Samobor>, <Grad: Varaždin>]

>>> Grad.objects.filter(geom__distance_gt=(zg.geom, Distance(km=100)))
[<Grad: Ljubljana>, <Grad: Budimpešta>]
```

Prvi primjer dohvaća sve gradove kojima je zračna udaljenost u odnosu na Zagreb manja od 100 kilometara. Uvjet je za to "distance_lt" kojem se dodjeljuju dvije vrijednosti – geometrija (točka) Zagreba i željena udaljenost u kilometrima. Pošto u taj uvjet ulazi i sam Zagreb, u sljedećem se primjeru isključuje pomoću metode "exclude". Na isti način se mogu dohvatiti gradovi kojima je udaljenost veća od 100 kilometara.

4.3. GeoAdmin i OSMGeoAdmin

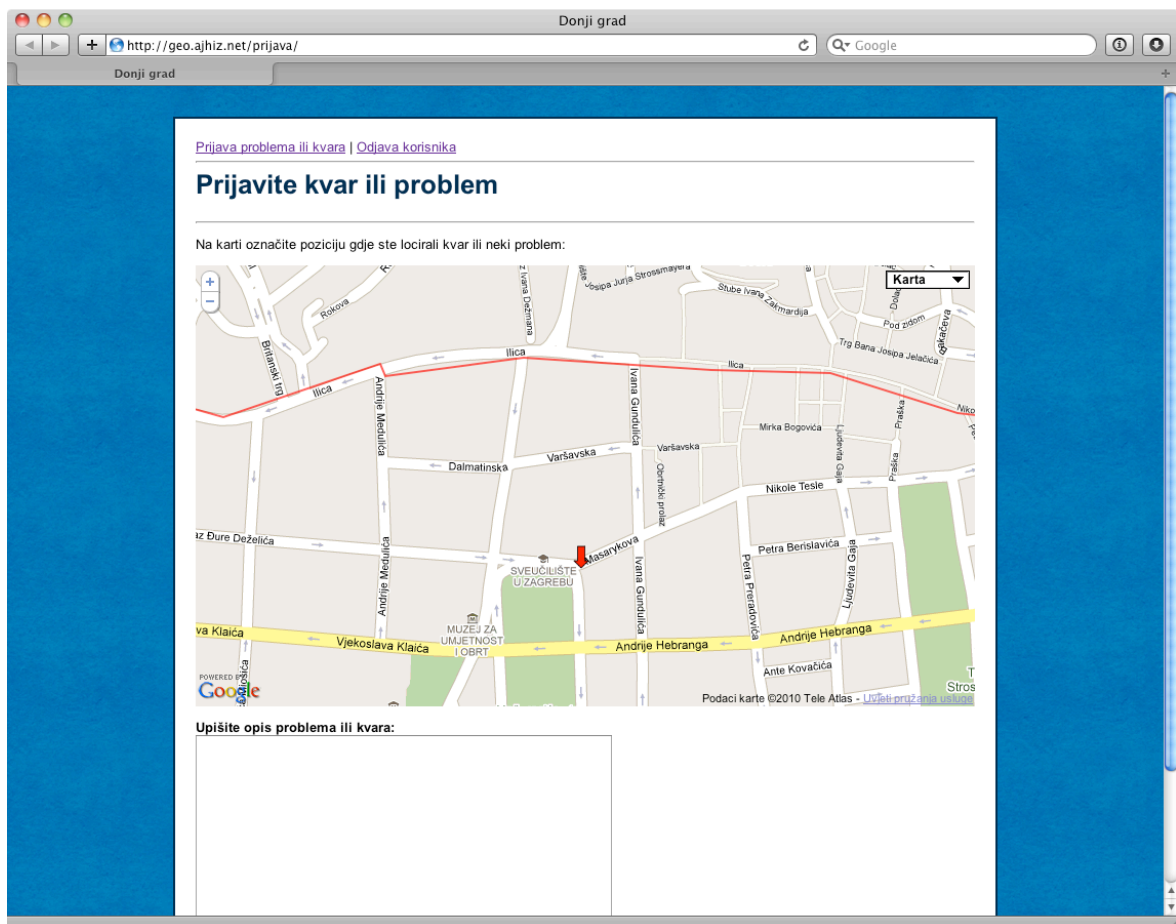
GeoDjango sadrži dodatke za administracijsko sučelje pomoću kojih je moguće stvaranje i modifikacija geometrijskih oblika. Osnovni je dodatak "GeoAdmin" koji koristi kao podlogu OpenLayers (URL 16) karte. Ovisno o tipu prostornog podatka (točka, poligon...) "GeoAdmin" pruža jednostavne alate za crtanje geometrijskih oblika (Slika 6). Za detaljnije podloge se može koristiti "OSMGeoAdmin" koji koristi Open Street Map slojeve. Sama aktivacija i podešavanje može se vidjeti u sljedećem poglavlju.



Slika 6. Korištenje mogućnosti za stvaranje i uređivanje prostornih podataka pomoću OSMGeoAdmin dodatka

5. Praktični rad

Kao primjer korištenja Django okruženja s GeoDjango dodatkom napravljena je web aplikacija kojom građani neke gradske četvrti mogu prijavljivati kvarove (npr. krvar na uličnoj rasvjeti) ili probleme (smeće na cesti). Prijavljivanje je jednostavno jer se označava na karti (Slika 7), a osobe koje su zadužene za rješavanje takvih problema (administratori) mogu na karti lako vidjeti gdje se problem nalazi.



Slika 7. Izgled gotove aplikacije

Web aplikacija se sastoji od nekoliko stranica koje služe za registraciju korisnika, prijavljivanje korisnika i samu prijavu problema ili kvarova. Administratori sustava mogu u administracijskom sučelju pregledavati i kontrolirati prijave i njihove prostorne podatke.

5.1. Instalacija softvera i podešavanje okruženja

Operacijski sustav na kojem se izvodi web aplikacija je GNU/Linux, točnije, Ubuntu distribucija GNU/Linux (verzija 10.04). Instalacija samog sustava opisana je u dokumentaciji distribucije (URL 17). U ovom je radu Ubuntu instaliran na udaljenom poslužiteljskom računalu te mu se pristupa korištenjem Interneta i SSH protokola. Poslužiteljem se upravlja putem tekstualnog sučelja, a datoteke se mogu prenositi korištenjem SCP ili SFTP protokola. Za pristup se koristi administratorsko korisničko ime i lozinka koji su stvoreni prilikom instalacije operacijskog sustava.

5.1.1. Instalacija poslužiteljskog softvera i biblioteka

Instalacija softvera na Ubuntu distribuciji vrlo je jednostavna. Koristi se alat “apt-get” koji služi za instalaciju softverskih paketa i nadograđivanje postojećih paketa. Nazive paketa potrebno je znati unaprijed, ali mogu se i pretraživati po nazivu:

```
$ apt-cache --names-only search postgis
libpostgis-java - geographic objects support for PostgreSQL -- JDBC
support
postgis - geographic objects support for PostgreSQL -- common files
postgis-8.4-postgis - geographic objects support for PostgreSQL 8.4
```

Alat “apt-cache” vraća sve pakete koji sadrže riječ “postgis”, a paket koji se traži je “postgis-8.4-postgis”. Naziv daje naslutiti kako je to dodatak PostgreSQL baze podataka verzije 8.4. Ako se taj paket pokuša instalirati prije PostgreSQL paketa, “apt-get” alat prepoznaje zavisnost (*dependency*) te instalira i sam PostgreSQL. Isti je slučaj s ostalim programima i bibliotekama te nije potrebno razmišljati o ostalim međuzavisnostima.

Za Django projekt potrebni su sljedeći paketi:

- Python
- Psycopg2 modul za Python
- Apache web poslužitelj s modulom mod_wsgi
- PostgreSQL s PostGIS dodatkom
- GEOS biblioteka
- PROJ.4 biblioteka
- GDAL biblioteka

Nakon što su pronađeni svi nazivi paketa, on se mogu najednom instalirati:

```
$ sudo apt-get install python python-psycopg2 apache2-mpm-prefork
libapache2-mod-wsgi postgresql-8.4 postgresql-8.4-postgis libgeos-
3.1.0 libproj0 libgdal1-1.6.0 gdal-bin libgdal-dev
```

Ako su neki paketi već instalirani, kao što je to slučaj s Pythonom koji je u sklopu osnovne Ubuntu instalacije, “apt-get” ih ignorira i nastavlja instalaciju ostalih paketa. Nakon pokretanja gornje naredbe ispisuju se svi paketi koji će biti instalirani (i paketi o kojima ovisе) te je potrebno potvrditi instalaciju istih.

5.1.2. Podešavanje PostgreSQL-a i stvaranje PostGIS baze podataka

PostgreSQL baza podataka inicijalno je dostupna samo Linux korisniku “postgres”. Potrebno je stvoriti novog PostgreSQL korisnika koji se podudara s administratorskim korisničkim imenom kojim se upravlja operacijskim sustavom:

```
$ sudo -u postgres createuser -superuser tomlav
$ sudo -u postgres psql
$ postgres=# \password tomlav
```

U ovom je slučaju korisničko ime “tomlav”, a u zadnjoj se liniji postavlja lozinka za tog korisnika. Time se olakšava daljnja administracija baze s klijentskim alatom “psql”.

Sljedeći je korak stvaranje PostGIS predloška za bazu podataka. Predložak će sadržavati prostorne funkcije i referentne koordinatne sustave koji su potrebni za same prostorne upite. PostGIS paket sadrži gotove SQL datoteke kojima se jednostavno stvara takav predložak:

```

$ # stvaranje prazne baze podataka koja će poslužiti kao predložak
$ createdb -E UTF8 template_postgis

$ # dodavanje podrške za plpgsql jezik
$ createlang -d template_postgis plpgsql # dodavanje podrške za

$ # označavanje baze "template_postgis" kao predložak
$ psql -d postgres -C "UPDATE pg_database SET datistemplate='true'
WHERE datname='template_postgis';"

$ # unošenje prostornih funkcija i geometrijskih polja u predložak
$ psql -d template_postgis -f
/usr/share/postgresql/8.4/contrib/postgis.sql

$ #Unošenje podataka o referentnim koordinatnim sustavima
$ psql -d template_postgis -f
/usr/share/postgresql/8.4/contrib/spatial_ref_sys.sql

$ # postavljanje prava kako bi svi korisnici imali pristup tim
podacima
$ psql -d template_postgis -c "GRANT ALL ON geometry_columns TO
PUBLIC;"
$ psql -d template_postgis -c "GRANT ALL ON spatial_ref_sys TO
PUBLIC;"

```

Stvoren je predložak naziva "template_postgis" prema kojem se može stvoriti nova baza podataka za projekt:

```
$ createdb -T template_postgis geodb
```

Baza je stvorena s imenom "geodb", a potreban je još korisnički račun kojim će se pristupati toj bazi:

```

$ createuser -l -P -S -R -D geouser
$ psql -d geodb -c "GRANT ALL PRIVILEGES ON DATABASE geodb TO
geouser;"

```

Prilikom stvaranja korisničkog računa "geouser" pojavljuje se polje za upis lozinke koja će se koristiti prilikom pristupa bazi.

5.1.3. Instalacija Django okruženja

Django je moguće instalirati putem paketnog sustava, ali je paket starije verzije, pa je potrebna ručna instalacija:

```

$ wget http://www.djangoproject.com/download/1.2.1/tarball/
$ tar xzf Django-1.2.1.tar.gz
$ cd Django-1.2.1/
$ sudo python setup.py install

```

Datoteka se prenosi pomoću alata "wget", otpakira pomoću "tar", a sama se instalacija izvršava putem "setup.py". Za provjeru uspješnosti instalacije potrebno je dohvatiti "django" modul putem interaktivnog *shella* Pythona:

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(1, 2, 1, 'final', 0)
```

Dohvaćena verzija Django modula podudara se s instaliranom i time se može zaključiti da je ispravno instaliran.

5.1.4. Stvaranje Django projekta

Django projekt je direktorij koji obuhvaća sve datoteke koje se koriste u web aplikaciji. Stvara se pomoću alata “django-admin.py” koji dolazi s Django okruženjem. Potrebno je odabrati naziv projekta:

```
$ django-admin.py startproject geo
```

U ovom je slučaju stvoren projekt “geo” gdje je alat “django-admin.py” napravio kostur projekta:

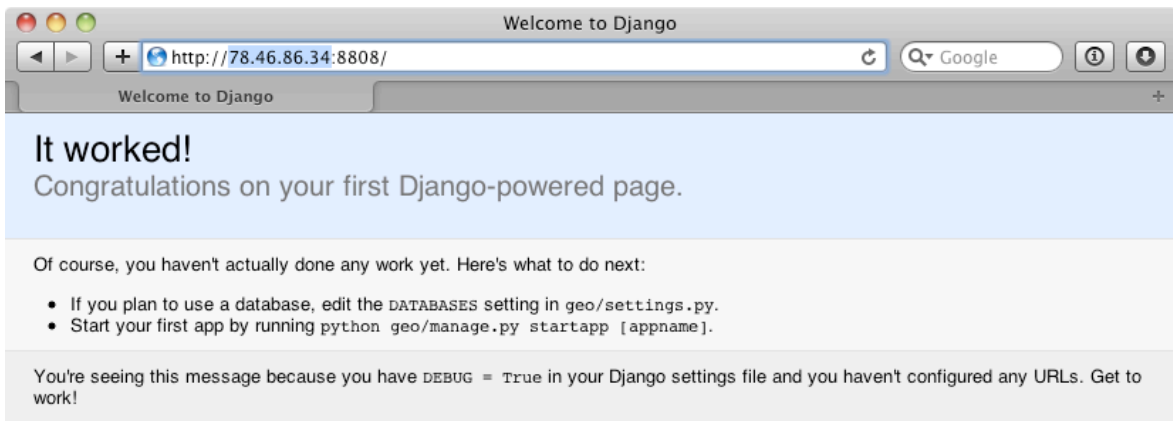
```
~/geo $ ls
__init__.py  manage.py  settings.py  urls.py
```

Datoteka “__init__.py” je prazna i postoji samo zato da se direktorij “geo” može ponašati kao Python modul, “manage.py” je skripta kojom će se dalje upravljati Django projektom, a “settings.py” sadrži osnovne postavke cijelog projekta kao što su podaci za spajanje na bazu podataka i slično. Datoteka “urls.py” kasnije se koristi za pridruživanje URL-ova funkcijama Django pogleda (*views*).

Rad se projekta može isprobati putem “manage.py” skripte i naredbe “runserver” koja pokreće testni web poslužitelj:

```
~/geo $ python manage.py runserver 78.46.86.34:8808
Validating models...
0 errors found
```

Kako bi sadržaj bio vidljiv preko Interneta, navodi se IP adresa poslužitelja i port koji se upisuju u Internet preglednik. U ovom je slučaju IP poslužitelja 78.46.86.34, a port je 8808. U Internet preglednik upisuje se adresa `http://78.46.86.34:8808/`, a rezultat se može vidjeti na slici (Slika 8).



Slika 8. Uspješno postavljen Django projekt

Kao što i sama slika prikazuje, testni web poslužitelj uspješno je pokrenut, a za daljnji je rad potrebno podesiti postavke u "settings.py" i stvoriti aplikaciju.

5.1.5. Podešavanje Django projekta

Datoteka "settings.py" već sadrži neke postavke, ali za početak je potrebno promijeniti samo podatke za pristup bazi podataka:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'geodb',
        'USER': 'geouser',
        'PASSWORD': 'geouser1234',
        'HOST': '127.0.0.1',
        'PORT': ''
    }
}
```

Navodi se tip baze podataka (u ovom slučaju "postgis"), ime baze (već prije stvorene "geodb") i korisnički račun koji može pristupiti toj bazi ("geouser" s lozinkom "geouser1234"). Nakon toga se izvodi sinkronizacija s bazom podataka:

```
~/geo $ python manage.py syncdb
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
. . .
You just installed Django's auth system, which means you don't have
any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'tomislav'):
E-mail address: tpavlovic@geof.hr
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Permission model
. . .
```

Pri prvom pokretanju sinkronizacije Django stvara tablice koje se koriste za interne potrebe te se stvara administratorski račun koji je potreban za administracijsko sučelje.

Administracijsko se sučelje aktivira dodavanjem “admin” aplikacije u “settings.py” i praćenjem uputa iz “urls.py” datoteke:

```
Datoteka “settings.py”:
# . . .
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
)
# . . .

Datoteka “urls.py”:
from django.conf.urls.defaults import *

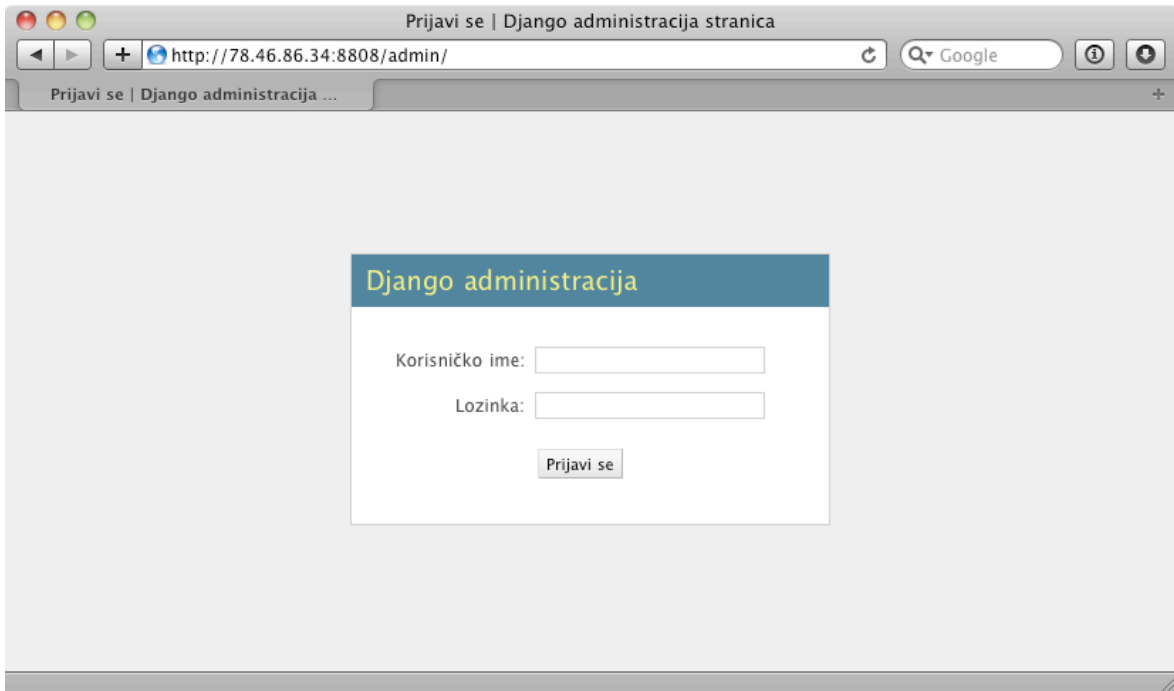
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^admin/', include(admin.site.urls)),
)
```

Potrebno je pokrenuti još jednom sinkronizaciju s bazom podataka te se pokreće testni web poslužitelj:

```
~/geo $ python manage.py syncdb
~/geo $ python manage.py runserver 0.0.0.0:8808
```

Kao što je i navedeno u “urls.py”, administraciji se pristupa preko adrese <http://78.46.86.34:8808/admin/> (Slika 9), a korisnički podaci za pristup su oni koji su stvoreni pri prvoj sinkronizaciji s bazom podataka.



Slika 9. Početni ekran Django administracije

Nakon uspješnog prijavljivanja pojavljuje se ekran s popisom ugrađenih aplikacija (“Auth” i “Sites”) (Slika 10). U odjeljku “Auth” mogu se stvarati novi korisnici i grupe korisnika, a “Sites” se u ovom slučaju neće koristiti. U administracijskom se sučelju ne može još ništa korisnoga raditi sve dok se ne napravi Django aplikacija.



Slika 10. Django administracija nakon prijave

5.1.6. Stvaranje Django aplikacije

Django projekt sam po sebi ne može raditi ništa korisno sve dok se unutar njega ne napravi aplikacija. Aplikacija će sadržavati modele podataka i poglede. Slično kao i kod stvaranja projekta, aplikacija mora imati svoj naziv (ovdje će to biti “web”), a pokreće se pomoću skripte “manage.py”:

```
~/geo $ python manage.py startapp web
```

Naredba stvara direktorij “web” i u njemu nekoliko datoteka:

```
~/test/geo/web $ ls  
__init__.py  models.py  tests.py  views.py
```

Kao i kod projekta, prazna datoteka “__init__.py” omogućava direktoriju da se ponaša kao Python modul. U datoteci “models.py” se stvaraju modeli podataka, a u “views.py” se unose funkcije pogleda (kontroleri iz MVC-a). Datoteka “tests.py” služi za pisanje funkcija koje će testirati cijelu aplikaciju, ali se u ovom projektu neće koristiti.

Novostvorenu aplikaciju potrebno je dodati u “settings.py” kako bi u Django projektu bila prepoznata:

```
Datoteka “settings.py”:  
# . . .  
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    'django.contrib.messages',  
    'django.contrib.admin',  
    'geo.web',  
)  
# . . .
```

5.2. Stvaranje modela podataka i podešavanje administracijskog sučelja

Za potrebe aplikacije stvaraju se tri osnovna modela podatka “Cetvrt”, “Zona” i “Prijava”. Model “Cetvrt” predstavlja gradsku četvrt s poligonom kao geometrijskim podatkom, model “Zona” sadrži poligon unutar gradske četvrti, a “Prijava” predstavlja prijavu kvara ili problema koji će geometrijski biti predstavljen točkom.

Uz modele koji sadrže geometrijske podatke, potreban je još jedan dodatni, a to je “KorisnickiProfil” koji sadrži dodatne podatke o korisniku.

Modeli se upisuju u datoteku “models.py”:

```
from django.contrib.gis.db import models
from django.contrib.gis import admin as geoadmin
from django.contrib.auth.models import User
from django.db.models.signals import post_save

class Cetvrt(models.Model):
    naziv = models.CharField(max_length=50)

    geom = models.PolygonField(srid=4326)

    objects = models.GeoManager()
    def __unicode__(self):
        return self.naziv

class CetvrtAdmin(geoadmin.OSMGeoAdmin):
    list_display = ('naziv',)

class Zona(models.Model):
    naziv = models.CharField(max_length=50)
    cetvrt = models.ForeignKey(Cetvrt)
    boja = models.CharField(max_length=50)

    geom = models.PolygonField(srid=4326)

    objects = models.GeoManager()

class ZonaAdmin(geoadmin.OSMGeoAdmin):
    list_display = ('naziv',)

class Prijava(models.Model):
    korisnik = models.ForeignKey(User)
    opis = models.TextField(max_length=1000, blank=True)
    vrijeme = models.DateTimeField(auto_now_add=True)

    geom = models.PointField(srid=4326)

    objects = models.GeoManager()

    def __unicode__(self):
        return self.korisnik.username

class KorisnickiProfil(models.Model):
    user = models.OneToOneField(User, primary_key=True)
    adresa = models.CharField(max_length=50, blank=True)
    telefon = models.CharField(max_length=50, blank=True)
    mobitel = models.CharField(max_length=50, blank=True)

def create_user_profile(sender, instance, created, **kwargs):
    if created:
        profile, created =
        korisnickiProfil.objects.get_or_create(user=instance)

post_save.connect(create_user_profile, sender=User)
```

Model “Cetvrt” predstavlja gradsku četvrt s nazivom i geometrijom poligona. Model “Zona” sadrži naziv i boju te je povezan stranim ključem na model “Cetvrt”. To znači da je svaka zona povezana s jednom gradskom četvrti, odnosno da

gradska četvrt sadrži više zona. Svaki model koji sadrži geometrije mora imati poseban atribut “objects” koji nastaje iz “GeoManager()” klase. Prostorni upiti bez njega ne bi bili mogući.

U modelu “Prijava” koristi se strani ključ prema modelu “User” (model koji nije potrebno definirati jer je u Django okruženju već definiran i može se koristiti) i opis koji je tekstualnog tipa. Django model “User” (korisnik) ne sadrži sva potrebna polja, pa je dodan model “KorisnickiProfil” koji sadrži adresu, telefon i mobitel te jedan-na-jedan vezu prema modelu “User”. Funkcija “create_user_profile” postoji kako bi se kod stvaranja korisnika stvorio automatski i korisnički profil. To se postiže registriranjem signala (“post_save”) koji pokreće funkciju “create_user_profile” svaki puta kad se objekt nastao pomoću modela “User” spremi u bazu podataka.

Nakon što su modeli napisani, potrebno je izvršiti sinkronizaciju s bazom podataka kako bi se stvorile tablice:

```
~/geo $ python manage.py syncdb
```

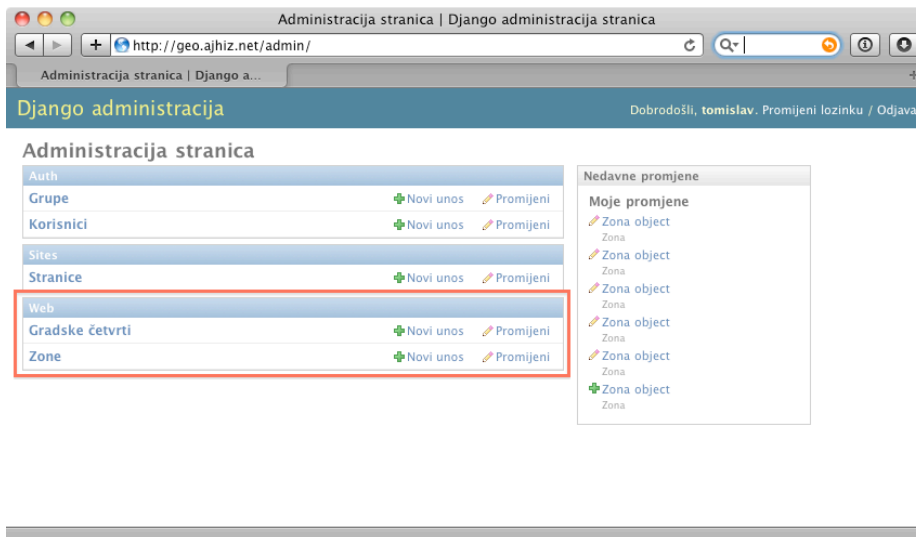
U datoteci “models.py” postoje klase “CetvrtAdmin” i “ZonaAdmin”. One moraju postojati kako bi se ti modeli pojavili u administracijskom sučelju te mogu sadržavati dodatne podatke kako će se prikazivati. U ovom je slučaju definirano da će na popisu objekata biti prikazan atribut “naziv”. Za razliku od “običnih” modela, ovi administracijski modeli nasljeđuju posebnu “OSMGeoAdmin” klasu koja omogućava prikaz geometrijskih podataka na OpenStreetMaps karti.

Administracijski modeli trebaju biti registrirani, a za to je potrebno stvoriti “admin.py” datoteku u direktoriju aplikacije i u nju staviti sljedeći sadržaj:

```
from django.contrib.gis import admin as geoadmin
from models import Cetvrt, CetvrtAdmin, Zona, ZonaAdmin

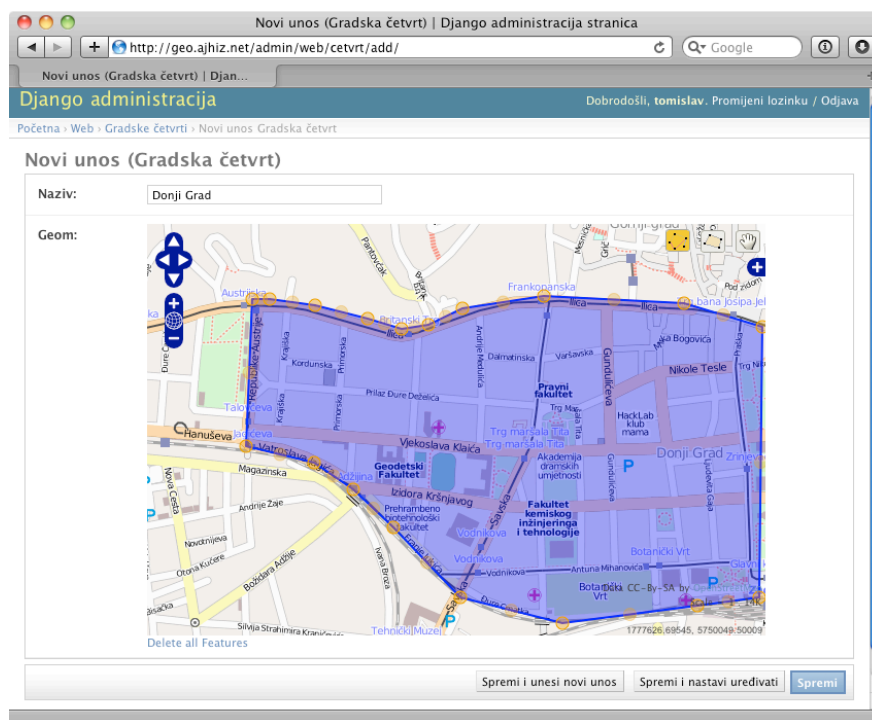
geoadmin.site.register(Cetvrt, CetvrtAdmin)
geoadmin.site.register(Zona, ZonaAdmin)
```

Pokretanjem testnog web poslužitelja i pristupanjem adresi administracije mogu se vidjeti novostvoreni modeli (Slika 11).



Slika 11. Prikaz stvorenih modela u Django administraciji

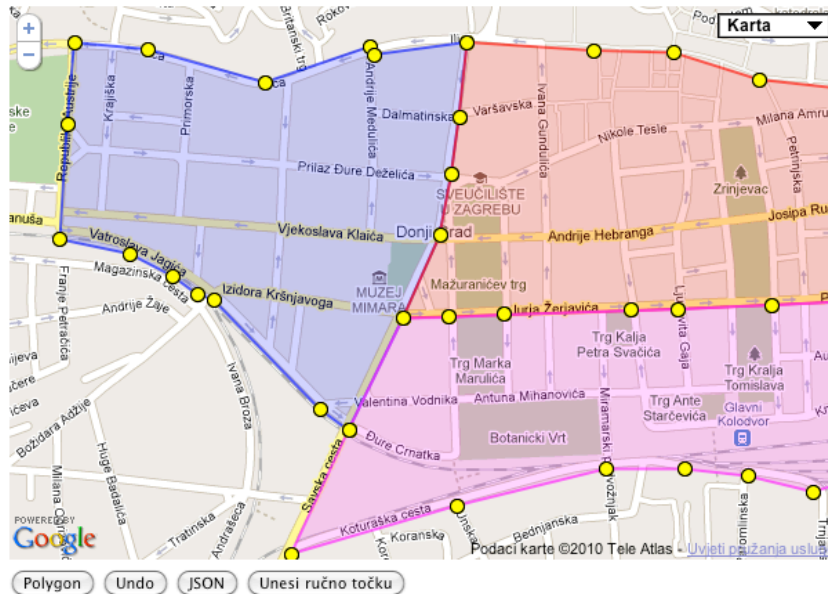
Pritiskom na “Novi unos” gradske četvrti ili zone može se stvoriti novi objekt. Geometrijsko je polje prikazano na OpenStreetMaps karti. Na istoj je karti moguće crtati geometrijske oblike, ovisno o tipu geometrijskog polja koje je navedeno u modelu. Geometrijski je tip gradske četvrti poligon pa su omogućeni jednostavni alati za crtanje poligona. Dostupni alati nisu namijenjeni za vektorizaciju, ali mogu poslužiti za brze i jednostavne promjene geometrijskih oblika.



Slika 12. Vektoriziranje u Django administraciji

5.3. Unos inicijalnih podataka

Za potrebe ovog praktičnog rada odabrana je zagrebačka četvrt Donji Grad koja će biti proizvoljno podijeljena na četiri zone. Za potrebe vektoriziranja napravljen je jednostavan program koji koristi Google Maps API (Slika 13). Vektoriziranje je obavljeno prema granicama navedenim na službenim stranicama Grada Zagreba (URL 18).



Slika 13. Vektoriziranje gradske četvrti i zona

Nacrtni se poligoni mogu dobiti u GeoJSON formatu koji je pogodan za unošenje u definirane modele. To se može napraviti putem interaktivnog *shella*:

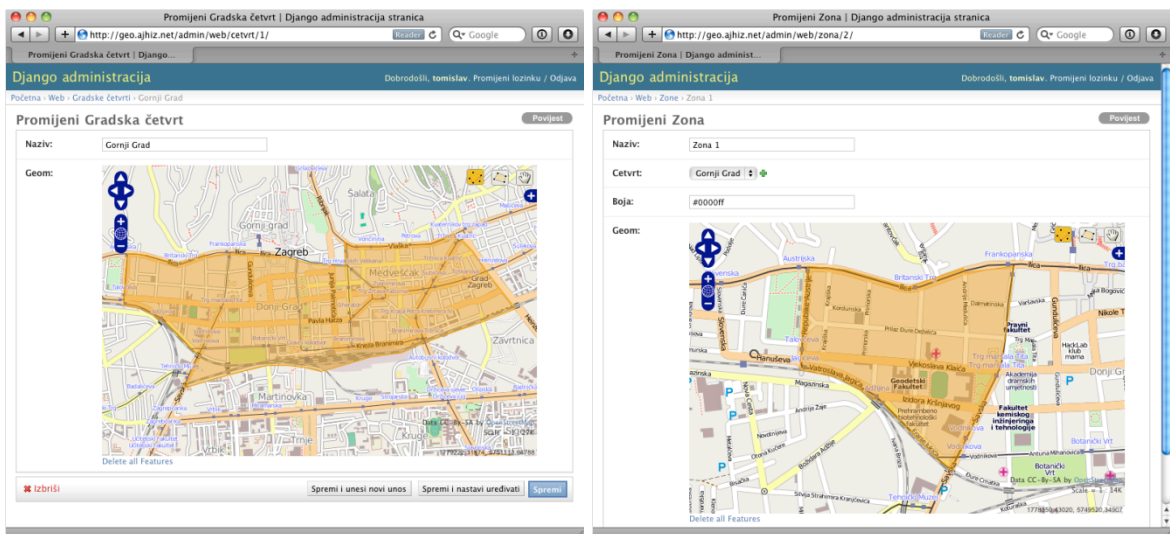
```
~/geo $ python manage.py shell
>>> from web.models import Cetvrt, Zona
>>> from django.contrib.gis.geos import GEOSGeometry
>>> donji_grad_objekt = Cetvrt()
>>> donji_grad_objekt.naziv = "Donji Grad"
>>> donji_grad_objekt.geom = GEOSGeometry('{ "type": "Polygon",
"coordinates": [ [ [15.957513, 45.813180], [15.959787, 45.813038],
[15.963435, 45.812320], [15.966718, 45.813098], <rez...> , [15.957277,
45.811408], [15.957513, 45.813180] ] ] }')
>>> donji_grad_objekt.save()

>>> zona1 = Zona()
>>> zona1.naziv = "Zona 1"
>>> zona1.geom = GEOSGeometry('{ "type": "Polygon", "coordinates": [ [
[ 15.966032, 45.804796 ], [ 15.967727, 45.807220 ], <rez ...> , [
15.966032, 45.804796 ] ] ] }')
>>> zona1.save()

>>> zona2 = Zona()
>>> zona2.naziv = "Zona 2"
>>> zona2.geom = GEOSGeometry(. . .)

>>> # ostale zone . . .
```

Unešena četvrt i zone mogu se vidjeti u administraciji (Slika 14).



Slika 14. Prikaz gradske četvrti Donji Grad i zone

5.4. Stvaranje korisničkog sučelja

Korisničko se sučelje stvara pomoću Django predložaka i pogleda. Pogledi sadrže programski kod koji dohvaća podatke potrebne za prikaz u Internet pregledniku. Django pogled te podatke šalje predlošku te se iz njega generira HTML kod. Sam se predložak sastoji od HTML-a i posebnih Django *tagova* koji služe za prikazivanje i manipulaciju podacima dobivenim iz pogleda.

Prednost je predložaka je što se ne mora za svaku stranicu pisati cijeli predložak, već se stvori jedan osnovni koji drugi predlošci “nasljeđuju” i mijenjaju samo dijelove koji su definirani kao blok.

5.4.1. Predlošci

Primjer takvog predloška može se vidjeti ovdje:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Donji grad</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <link rel="stylesheet" type="text/css" media="screen"
href="{{ MEDIA_URL }}css/screen.css"/>
    {% block head %}

    {% endblock %}
  </head>
  <body>
    <div class="container">
```

```

        <div class="inner-container">
            <div class="head">
                <a href="/">Prijava problema ili kvara</a> |
                <a href="/odjava/">Odjava korisnika</a>
            </div>
            <hr/>
            {% block sadrzaj %}
            {% endblock %}
        </div>
    </div>
</body>
</html>

```

Ovakav osnovni predložak sadrži HTML kod i Django *tagove*. *Tag* koji počinje s “{{” i završava s “}}” ispisuje navedenu varijablu (koja je poslana iz pogleda). Varijabla “MEDIA_URL” specijalna je varijabla koja je uvijek dostupna. *Tag* koji počinje s “{%” i završava s “%}” obično ima svoj početak i kraj. U ovom se slučaju definira prazan blok koji počinje s “{% block NAZIV_BLOKA %}” i završava s “{% endblock %}”. Blokovi su bitni zbog toga što ih je moguće popunjavati u drugim predlošcima koji nasljeđuju osnovni predložak. Predložak za početnu stranicu tada izgleda ovako:

```

{% extends 'base.html' %}

{% block sadrzaj %}
<h1>Dobrodošli</h1>
<hr/>
<p>Za korištenje sustava potrebno se prijaviti. Ako nemate korisnički račun kliknite na gumb "Registracija".</p>
<hr/>
<form method="post" action="/">{% csrf_token %}
    {% if poruka %}<ul class="errorlist"><li>{{ poruka }}</li></ul>{%
endif %}
    <p>
        <label for="korisnicko_ime">Korisničko ime:</label>
        <input class="text" type="text" name="korisnicko_ime"
value=""/>
    </p>
    <p>
        <label for="lozinka">Lozinka:</label>
        <input class="text" type="password" name="lozinka" value=""/>
    </p>
    <p>
        <input type="submit" value="Prijava"/>
    </p>
</form>
<hr/>
<p>
<a class="gumb" href="/registracija/">Registracija</a>
</p>
{% endblock %}

```

Predložak s *tagom* “extends” nasljeđuje osnovni predložak te popunjava blok “sadrzaj”. Kada se iz takvog predloška generira HTML, on sadrži sve svoje elemente i elemente iz osnovnog predloška. Time se olakšavaju globalne promjene na predlošcima gdje je određenu stvar potrebno promijeniti samo na

osnovnom predlošku. Django predložak može sadržavati standardne programske konstrukcije kao što su “if” i “for” te filtere koji se mogu primijeniti nad varijablama.

Predlošci se spremaju u direktorij naveden u “settings.py”:

```
TEMPLATE_DIRS = (  
    "/home/tomislav/geo/templates",  
)
```

Za potrebe ovog projekta stvoreno je 5 predložaka:

- osnovni predložak,
- predložak za početnu stranicu,
- predložak za registraciju korisnika,
- predložak za prijavu problema i
- predložak za poruku o uspješnoj prijavi problema.

5.4.2. Jednostavan pogled

Pogledi su funkcije koje služe za dohvat podatka, obradu i generiranje HTML-a iz predložaka. Pogledi se pišu u datoteci “views.py”, a najjednostavniji se pogled s predloškom može napisati ovako:

```
from django.shortcuts import render_to_response  
  
def index(request):  
    data = {}  
    return render_to_response('index.html', data, request)
```

Naziv je pogleda proizvoljan, a u ovom je slučaju “index”. Funkcija prima parametar “request” koji sadrži informacije o korisniku i samom zahtjevu koji je korisnik uputio s Internet preglednikom. Funkcija “render_to_response” skraćeni je oblik stvaranja objekta “HttpResponse” koji svaki pogled mora vratiti. Parametri su joj naziv datoteke predloška, podaci spremljeni u asocijativnom polju i sam “request” objekt.

5.4.3. URL uzorci

Kako bi se pogled “index” mogao izvršiti, potrebno ga je povezati s URL-om. Najprije je potrebno u datoteci “urls.py” dodati jedan redak koda:

```
from django.conf.urls.defaults import *  
  
from django.contrib import admin  
admin.autodiscover()
```

```
urlpatterns = patterns('',
    (r'^admin/', include(admin.site.urls)),
    (r'', include('geo.web.urls')),
)
```

Podobljeni dio koda javlja Django okruženju da će svaki zahtijevani URL obrađivati modul “urls” iz aplikacije “web”. Nakon toga potrebno je napraviti “urls” modul u toj aplikaciji. Taj se modul stvara s datotekom “urls.py” koja se treba nalaziti u direktoriju aplikacije:

```
from django.conf.urls.defaults import *
urlpatterns = patterns('geo.web.views',
    (r'^$', 'index'),
    # . . .
)
```

Modul “urls” sastoji se od liste uzoraka (*patterna*) gdje je prvi parametar u jednom elementu liste regularni izraz (*regular expression*), a drugi je funkcija, odnosno pogled koji će se izvršiti kada taj regularni izraz odgovara ulaznom tekstu, odnosno dijelu URL-a (dio koji se nalazi iza `http://www.primjer.com/`). Izraz “^\$” će se pogoditi s URL-om koji nema ništa nakon kose crte, odnosno, označava početnu stranicu. Popis svih URL uzoraka ovog projekta može se vidjeti ovdje:

```
from django.conf.urls.defaults import *
urlpatterns = patterns('geo.web.views',
    (r'^$', 'index'),
    (r'^registracija/$', 'registracija'),
    (r'^prijava/$', 'prijava_kvara'),
    (r'^prijava-uspjesna/$', 'prijava_uspjesna'),
    (r'^cetvrt-json/(?P<cetvrt_id>[0-9]+)/$', 'cetvrt_json'),
    (r'^odjava/$', 'odjava'),
)
```

Popis ukazuje na to da postoji šest uzoraka preko kojih se dolazi do pogleda. Npr. `http://geo.ajhiz.net/registracija/` će izvršiti pogled “registracija”, a `/cetvrt-json/1/` će pokrenuti pogled “cetvrt_json” i poslati parametar s nazivom “cetvrt_id” i vrijednošću 1.

5.4.4. Stvaranje pogleda u aplikaciji

Nakon što su URL-ovi povezani s pogledima, mogu se stvoriti funkcije koje vraćaju podatke. Pogled početne stranice izgleda ovako:

```
from django.http import HttpResponse, HttpResponseRedirect
```

```

from django.template import RequestContext
from django.shortcuts import render_to_response, get_object_or_404
from django.views.decorators.csrf import csrf_protect
from django.contrib.auth.models import User
from django.core.urlresolvers import reverse
from models import Cetvrt, Zona, Prijava

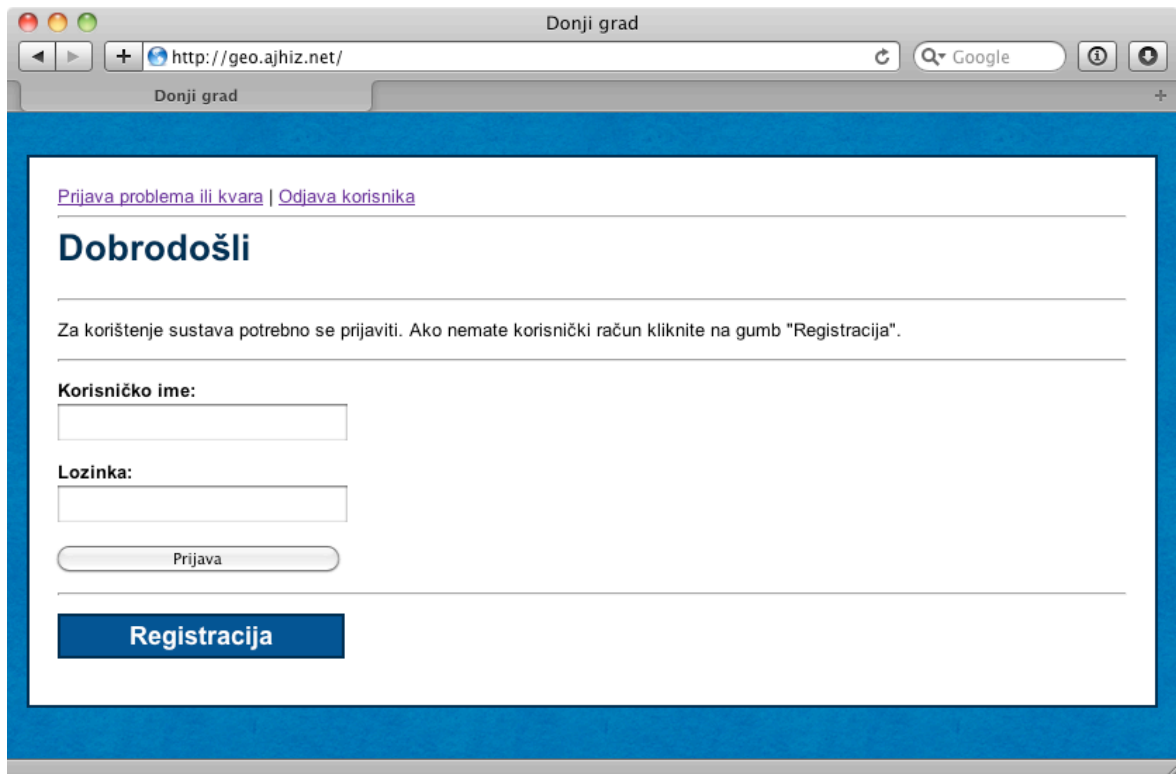
@csrf_protect
def index(request):
    data = {}
    if request.user.is_authenticated():
        return
    HttpResponseRedirect(reverse('geo.web.views.prijava_kvara'))
    if request.method == 'POST':
        username = request.POST['korisnicko_ime']
        password = request.POST['lozinka']
        user = authenticate(username=username, password=password)
        if user is not None:
            login(request, user)
            return
    HttpResponseRedirect(reverse('geo.web.views.prijava_kvara'))
    else:
        data['poruka'] = "Pogrešno korisničko ime ili lozinka"

    return render_to_response('index.html', data,
    RequestContext(request))

```

Prvo se provjerava je li korisnik autentificiran, odnosno, je li prijavljen u sustav. Korisnički se računi mogu stvoriti u administracijskom sučelju, ali sami korisnici neće imati pristup, već samo administratori sustava. Zbog toga je napravljena posebna stranica putem koje se korisnici mogu sami registrirati, a bit će opisana kasnije.

Ako je korisnik autentificiran, prenosi ga se automatski na stranicu gdje se prijavljuju problemi (pogled 'prijava_kvara'). Ako korisnik nije autentificiran, ostaje na početnoj stranici koja mu nudi mogućnost prijave u sustav (ako već posjeduje korisnički račun) i mogućnost registracije (ako nema korisnički račun) (Slika 15).



Slika 15. Početna stranica web aplikacije

Ako korisnik nema korisnički račun, pritiskom na gumb “Registracija” on se preusmjerava na pogled “registracija”:

```
@csrf_protect
def registracija(request):
    if request.method == 'POST':
        form = RegistracijaForm(request.POST)
        if form.is_valid():
            user = User.objects.create_user(
                username=form.cleaned_data['korisnicko_ime'],
                email=form.cleaned_data['email'],
                password=form.cleaned_data['lozinka1'])
            user.is_active = True
            user.save()

            profil = user.get_profile()

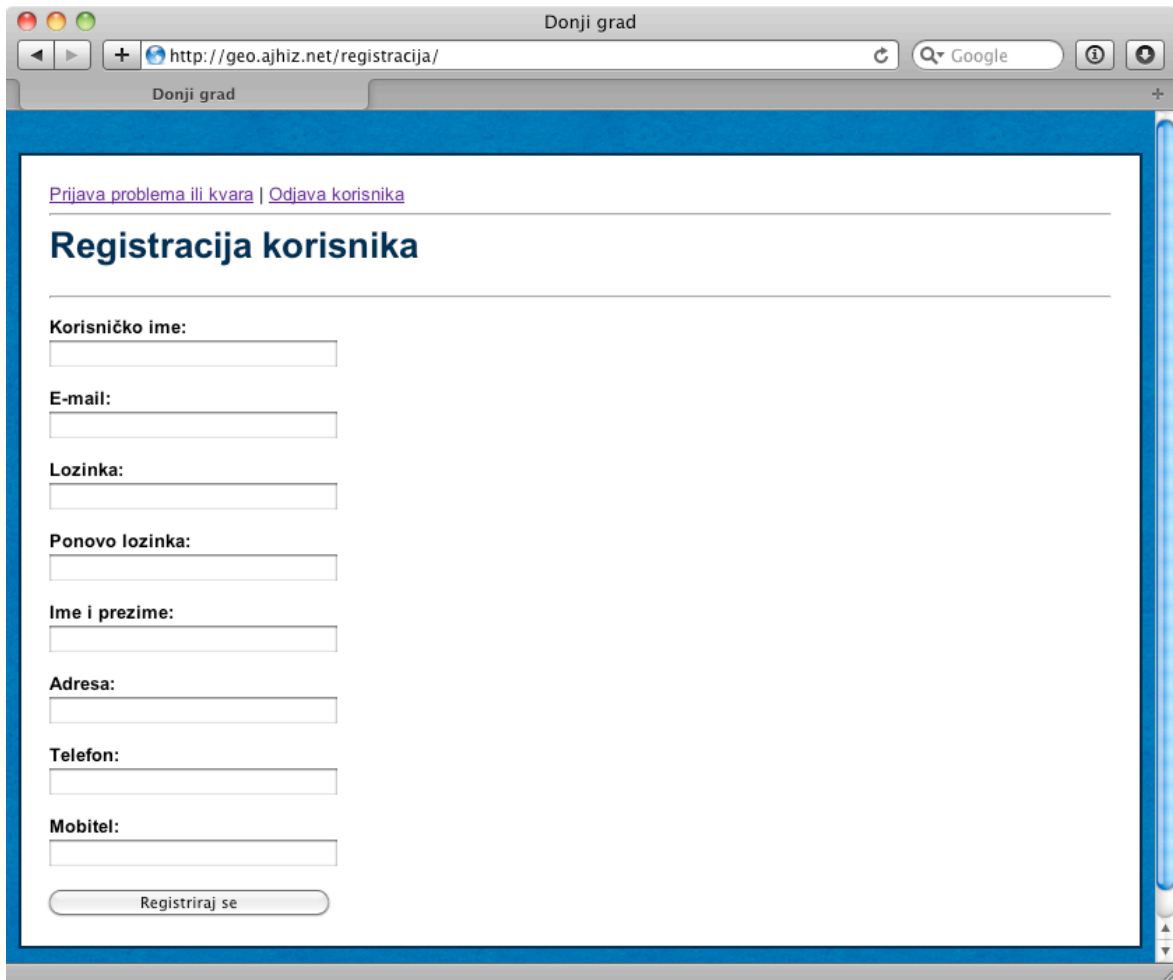
            profil.ime_prezime = form.cleaned_data['ime_prezime']
            profil.adresa = form.cleaned_data['adresa']
            profil.telefon = form.cleaned_data['telefon']
            profil.mobitel = form.cleaned_data['telefon']

            profil.save()

            return
    HttpResponseRedirect(reverse('geo.web.views.index'))
    else:
        form = RegistracijaForm()

    data = {
        'registracija_form': form
    }
    return render_to_response("korisnik_registracija.html", data,
        context_instance = RequestContext(request))
```

Pogled “registracija” koristi formu “RegistracijaForm” koja je definirana u datoteci “forms.py”. Ona služi za jednostavno stvaranje obrazaca, validaciju unosa podataka i jednostavan prikaz u HTML-u. Kada je korisniku ova stranica prvi puta predstavljena, prikazuje se prazan obrazac s potrebnim poljima (Slika 16).



The screenshot shows a web browser window with the address bar containing "http://geo.ajhiz.net/registracija/". The page title is "Donji grad". The main content area has a blue header with the text "Registracija korisnika". Below the header, there are several input fields: "Korisničko ime:", "E-mail:", "Lozinka:", "Ponovo lozinka:", "Ime i prezime:", "Adresa:", "Telefon:", and "Mobitel:". At the bottom of the form is a button labeled "Registriraj se".

Slika 16. Prikaz stranice za registraciju

Predložak je kratak i koristi funkciju “as_p” koji formu prenesenu iz pogleda generira u HTML-u:

```
{% extends 'base.html' %}
{% block sadrzaj %}
<h1>Registracija korisnika</h1>
<hr/>
<form action="/registracija/" method="post">{% csrf_token %}
  {{ registracija_form.as_p }}
  <input type="submit" value="Registriraj se" />
</form>
{% endblock %}
```

Prilikom pritiska gumba “Registriraj se” podaci se šalju pogledu “registracija” koji preko “request” objekta dohvaća vrijednosti, šalje ih u formu

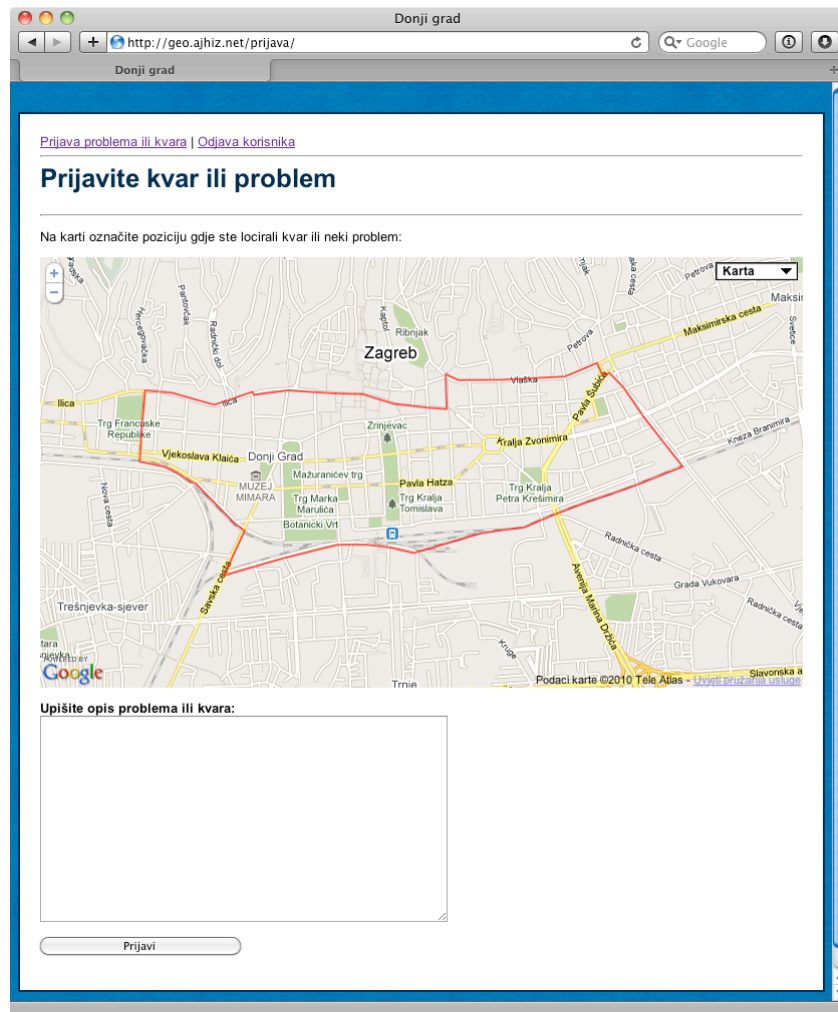
“RegistracijaForma” koja validira podatke te javlja je li obrazac ispravno ispunjen. Ako je ispravno ispunjen, stvara se korisnik s navedenim podacima i preusmjerava ga se na početni pogled. Ako je obrazac neispravno popunjen (npr. ako nedostaje korisničko ime ili e-mail adresa), on se ponovo prikazuje, ali ovaj puta s opisima grešaka (npr. “Niste popunili polje s korisničkim imenom”).

Ako je registracija uspješno obavljena, prijava se obavlja na početnoj stranici s upisom korisničkog imena i lozinke (istim koji su upisani u procesu registracije).

Nakon uspješne prijave, korisnika se automatski preusmjeruje na pogled “prijava_kvar” koji sadrži sljedeće:

```
def prijava_kvara(request):
    if request.method == 'POST':
        form = PrijavaKvaraForm(request.POST)
        if form.is_valid():
            nova_prijava = Prijava()
            cd = form.cleaned_data
            koordinate = cd['koordinate'].split(',')
            nova_prijava.geom = Point(float(koordinate[1]),
float(koordinate[0]))
            nova_prijava.opis = cd['opis']
            nova_prijava.korisnik = request.user
            nova_prijava.save()
            return
    HttpResponseRedirect(reverse('geo.web.views.prijava_uspjesna'))
    else:
        form = PrijavaKvaraForm()
        data = {
            'prijava_form': form
        }
        return render_to_response('prijava.html', data,
RequestContext(request))
```

Taj pogled koristi formu naziva “PrijavaKvaraForm”. Pri prvom pregledavanju te stranice, otvara se obrazac za prijavu kvarova i problema (Slika 17).



Slika 17. Stranica za prijavu kvarova ili problema

Predložak za ovaj pogled malo je složeniji zbog korištenja JavaScripta i Google Maps API-ja:

```
{% extends 'base.html' %}
{% block head %}
    <script src="{{ MEDIA_URL }}js/jquery.min.js"></script>
    <script
src="http://maps.google.com/maps/api/js?sensor=false&language=hr&region=hr" type="text/javascript" ></script>
    <script src="{{ MEDIA_URL }}js/json2.js"></script>
    <script src="{{ MEDIA_URL }}js/ggeojson.js"></script>
    <script src="{{ MEDIA_URL }}js/prijava.js"></script>
{% endblock %}

{% block sadrzaj %}
<h1>Prijavite kvar ili problem</h1>
<hr/>
<p class="greske">
    {{ prijava_form.koordinate.errors }}
    {{ prijava_form.opis.errors }}
</p>
<form method="post" action="/prijava/">{% csrf_token %}
    <p>
        Na karti označite poziciju gdje ste locirali kvar ili neki
        problem:
    </p>
</form>
```

```

        <div id="map-canvas"></div>
        <input type="hidden" id="koordinata" name="koordinata"
value="" />
    </p>
    <p>
        <label for="opis">Upišite opis problema ili kvara:</label>
        <textarea name="opis"></textarea>
    </p>
    <p>
        <input type="submit" value="Prijavi" />
    </p>
</form>
{% endblock %}

```

U zaglavlje HTML-a dodaju se skripte potrebne za iscrtavanje karte na ekranu. U datoteci "prijava.js" funkcija "map_init" pokreće iscrtavanje Google Maps karte:

```

var MAP = null;
var MARKER = null;
function map_init() {
    var latlng = new google.maps.LatLng(45.808282, 15.981202);
    var parametri = {
        zoom: 14,
        center: latlng,
        draggableCursor: 'crosshair',
        draggingCursor: 'pointer',
        mapTypeControl: true,
        mapTypeControlOptions: {
            style: google.maps.MapTypeControlStyle.DROPDOWN_MENU
        },
        navigationControl: true,
        navigationControlOptions: {
            style: google.maps.NavigationControlStyle.SMALL
        },
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    MAP = new google.maps.Map(document.getElementById("map-canvas"),
parametri);
}

```

Zadana je početna pozicija prikaza ("latlng"), razina uvećanja i parametri vezani uz sam izgled te karte. Takva karta sama po sebi ništa ne radi, već se može samo pregledavati. Dodatni kod koji služi za iscrtavanje poligona gradske četvrti slijedi nakon toga:

```

$.get('/cetvrt-json/1/', function(data) {
    var cetvrt_poly = GGeoJSON.parse(data, {
        strokeColor: 'red',
        strokeOpacity: 0.7,
        clickable: false,
        strokeWeight: 2,
        fillOpacity: 0,
        draggableCursor: 'crosshair',
        draggingCursor: 'pointer'
    });
    cetvrt_poly.setMap(MAP);
});

```


Pomoću jQuery biblioteke dohvaća se URL “/cetvrt-json/1/” koja vraća GeoJSON formatiran poligon gradske četvrti. Biblioteka GGeoJSON pretvara dohvaćene podatke u poligon prikladan prikazivanju na Google Maps karti. Kako bi poligon bio ispravno prikazan, potrebno je odrediti parametre izgleda kao što su boja linije, boja popunjenja i slično.

Prikazani poligon pomaže korisnicima kako bi lakše ustanovili koje su granice četvrti i gdje mogu prijavljivati probleme. Prijava se izvodi pomoću klika na kartu gdje se stvara oznaka (crvena strelica) (Slika 18).



Slika 18. Prikaz oznake na karti

Označavanje se izvodi pomoću JavaScripta sljedećim kodom:

```
map_click = google.maps.event.addListener(MAP, 'click', function
(event) {
  if(MARKER == null) {
    var strelica = new google.maps.MarkerImage('/media/strelica.png',
      new google.maps.Size(16, 23),
      new google.maps.Point(0,0),
      new google.maps.Point(8, 23));

    MARKER = new google.maps.Marker({
      position: event.latLng,
      map: MAP,
      icon: strelica
    });
  } else {
    MARKER.setPosition(event.latLng);
  }
  $('#koordinat').val(event.latLng.lat() + ',' + event.latLng.lng());
})
```

Metodom “addListener” registrira se funkcija koja će se izvršiti klikom na kartu. Funkcija stvara novi objekt “Marker” (oznaku) s pozicijom gdje je kliknuto mišem. Ako oznaka postoji na karti, tada se samo mijenja njezina pozicija. Pri svakoj se

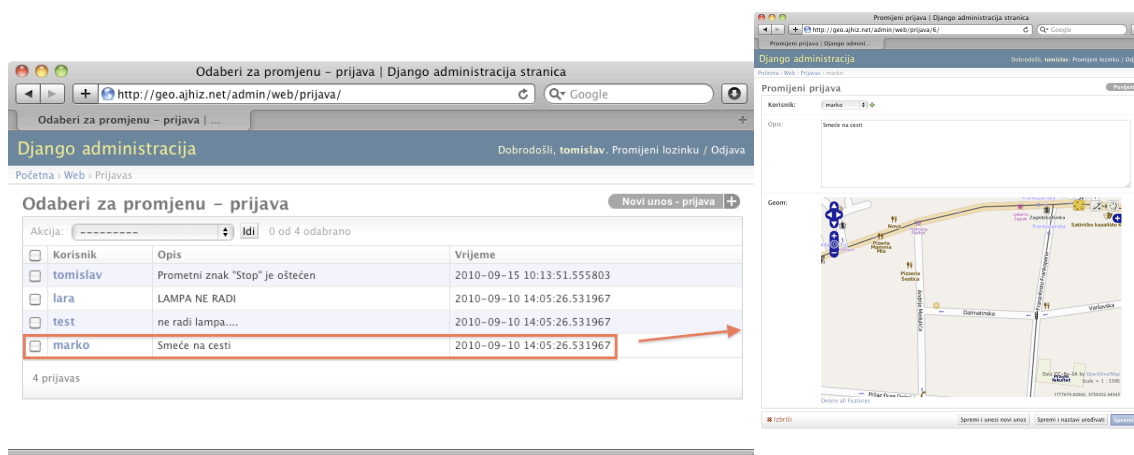
promjeni mijenja vrijednost skrivenog polja u HTML-u koji sadrži koordinate te oznake.

Kada korisnik označi mjesto kvara ili problema na karti i napiše opis, može kliknuti na gumb "Prijavi". Prilikom prijavljivanja provjerava se je li označeno mjesto na karti i je li popunjen opis. Također, provjerava se nalazi li se označena pozicija u toj četvrti. Kod za validaciju nalazi se u formi "PrijavaKvaraForm":

```
class PrijavaKvaraForm(forms.Form):
    koordinate = forms.CharField(max_length=200)
    opis = forms.CharField(max_length=500)
    def clean(self):
        cleaned_data = self.cleaned_data
        koordinate = cleaned_data.get("koordinate")
        opis = cleaned_data.get("opis")
        if koordinate == '' or koordinate == None:
            msg = u"Morate označiti mjesto na karti!"
            self._errors["koordinate"] = ErrorList([msg])
        else:
            from django.contrib.gis.geos import Point
            k = koordinate.split(',')
            tocka = Point(float(k[1]), float(k[0]))
            cetvrt = Cetvrt.objects.get(id=1)
            if not tocka.within(cetvrt.geom):
                msg = u"Mjesto koje ste označili mora biti u
granicama četvrti!"
                self._errors["koordinate"] = ErrorList([msg])
        if opis == '' or opis == None:
            msg = u"Morate unijeti opis problema ili kvara!"
            self._errors["opis"] = ErrorList([msg])
        return cleaned_data
```

Dohvaćena vrijednost koordinata pretvara se u geometriju pomoću klase "Point" (iz GEOS biblioteke), a geometrija gradske četvrti (poligon) dohvaća se iz modela podatka "Cetvrt". Metodom "within" provjerava se pozicija oznake na način da se ispituje nalazi li se ona u poligonu gradske četvrti. Ako se ne nalazi, korisnika se obavještava o tome.

Popis prijava se može vidjeti u administrativnom sučelju, a klikom na jednu prijavu moguće je vidjeti detalje prijave i poziciju na karti (Slika 19).



Slika 19. Popis prijava i detalji jedne prijave

5.5. Daljnji razvoj

U ovom je poglavlju opisan način izrade web sustava pomoću Django okruženja (s GeoDjango dodatkom). Pokazano je da je sustav vrlo fleksibilan i lako je dodati nove mogućnosti od kojih su neke u ovom radu izostavljene zbog usredotočenosti na sam GeoDjango.

Mogućnost koja se logično nameće je proširivanje modela "Prijava" koji bi trebao sadržavati tip prijave (npr. oštećenje prometnice, kvar na rasvjeti, smeće...) pomoću kojeg je moguće filtrirati prijave po njihovom tipu. Također, potreban je i atribut koji sadrži status prijave – je li prijava pregledana, ispravna, u postupku rješavanja ili riješena. Korisnik bi se obavještavao e-mail porukom pri svakoj promjeni statusa te bi svojim odgovorima i primjedbama na nju mogao olakšati rad osobama zaduženim za rješavanje problema.

Administratori bi trebali imati jednostavan pregled svih prijava na karti (filtrirani po statusu) koji je sličan prikazu kod postupka prijavljivanja. Na tom bi prikazu mogli odmah vidjeti koje se prijave odnose na isti problem.

6. Zaključak

Ovim je radom prikazan način izrade web aplikacije koja koristi prostorne podatke. Aplikacija je veoma fleksibilna i može se manjim programerskim zahvatima prilagoditi za različite namjene. Prednost joj je i to što se može izvršavati na različitim operacijskim sustavima, a za korištenje je potreban samo Internet preglednik i veza prema Internetu.

Problemi koji mogu nastati prilikom izrade takve aplikacije su najčešće tehničke prirode zbog velikog broja različitih komponenti (programi, alati, biblioteke) koje je potrebno uklopiti u jedan sustav i pobrinuti se da on ispravno radi. U ovom praktičnom radu nije bilo takvih problema zbog dobrog paketnog sustava Ubuntu distribucije koji je riješio sve ovisnosti između programa, alata i biblioteka.

Aplikacija stvorene pomoću GeoDjango okruženja najčešće egzistiraju na Internetu, a za to je potrebno zakupiti samostalno poslužiteljsko računalo kojem je danas cijena vrlo niska. Za izradu same aplikacije potrebno je poznavati programski jezik Python, Django okruženje i osnovno znanje o prostornim bazama podataka.

Potencijalni korisnici (kupci) aplikacije mogu biti ministarstva, vladine organizacije i razna komunalna poduzeća koja žele imati izravniji kontakt s građanima nekog područja.

7. Literatura

Bellussi A., Catania B., Clementini E., Ferrari E. (2007): Spatial Data on the Web, Springer, Berlin (Germany)

Bennett J. (2009): Practical Django Projects (Second Edition), Apress, Berkley (USA)

Douglas K. (2005): PostgreSQL (Second Edition), Sams, USA

Flanagan D. (2006): Javascript The Definitive Guide (Fifth Edition), O'Reilly Media, Sebastopol (USA)

Lutz M. (2009): Learning Python (Fourth Edition), O'Reilly Media, Sebastopol (USA)

Popis URL-ova

URL 1. General Python FAQ - Python 2.7 documentation, <http://docs.python.org/faq/general.html>, 26.08.2010.

URL 2. PEP 249 -- Python Database API Specification v2.0, <http://www.python.org/dev/peps/pep-0249/>, 26.08.2010.

URL 3. Psycopg - PostgreSQL database adapter for Python, <http://initd.org/psycopg/>, 26.08.2010.

URL 4. XHTML 1.1 - Module-based XHTML, <http://www.w3.org/TR/2001/REC-xhtml11-20010531/>, 26.08.2010.

URL 5. HTML5, <http://www.w3.org/TR/2010/WD-html5-20100624/>, 27.08.2010.

URL 6. Cascading Style Sheets, Level 2, <http://www.w3.org/TR/2008/REC-CSS2-20080411/>, 27.08.2010.

URL 7. When can I use, <http://caniuse.com/>, 27.08.2010.

URL 8. jQuery: The Write Less, Do more, JavaScript Library, <http://jquery.com/>, 01.09.2010.

- URL 9. Ajax (programming) - Wikipedia, the free encyclopedia,
[http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)), 01.09.2010.
- URL 10. About OGC, <http://www.opengeospatial.org/ogc>, 01.09.2010.
- URL 11. GeoDjango - A world-class Geographic Web Framework,
<http://geodjango.org/presentations/>, 29.08.2010.
- URL 12. thematicmapping.org,
http://thematicmapping.org/downloads/world_borders.php, 25.08.2010.
- URL 13. OGR: OGR Simple Feature Library, <http://www.gdal.org/ogr/>, 01.09.2010.
- URL 14. Django | QuerySet API reference | Django documentation,
<http://docs.djangoproject.com/en/1.2/ref/models/querysets/>, 01.09.2010.
- URL 15. Django | GeoQuerySet API Reference | Django documentation,
<http://docs.djangoproject.com/en/dev/ref/contrib/gis/geoquerysets/>,
01.09.2010.
- URL 16. OpenLayers: Home, <http://openlayers.org/>, 01.09.2010.
- URL 17. Ubuntu Installation Guide, <https://help.ubuntu.com/10.04/installation-guide/i386/index.html>, 01.09.2010.
- URL 18. Zagreb.hr - Službene stranice Grada Zagreba - Osnovni podaci,
<http://www.zagreb.hr/default.aspx?id=13067>, 01.09.2010.
- URL 19. Django | Quick install guide | Django documentation,
<http://docs.djangoproject.com/en/dev/intro/install/>, 27.08.2010.

8. Popis slika

Slika 1. Hijerarhija tipova geometrije	5
Slika 2. Koncept modela, pogleda i kontrolera u MVC-u	13
Slika 3. Dijagram toka u Django aplikaciji.....	14
Slika 4. Prikaz retka u tablici baze podataka pomoću phpPgAdmin sučelja.....	23
Slika 5. Prikaz redaka granica država u phpPgAdmin sučelju.....	24
Slika 6. Korištenje mogućnosti za stvaranje i uređivanje prostornih podataka pomoću OSMGeoAdmin dodatka.....	29
Slika 7. Izgled gotove aplikacije.....	30
Slika 8. Uspješno postavljen Django projekt.....	35
Slika 9. Početni ekran Django administracije.....	37
Slika 10. Django administracija nakon prijave	37
Slika 11. Prikaz stvorenih modela u Django administraciji	41
Slika 12. Vektoriziranje u Django administraciji	41
Slika 13. Vektoriziranje gradske četvrti i zona	42
Slika 14. Prikaz gradske četvrti Donji Grad i zone.....	43
Slika 15. Početna stranica web aplikacije.....	48
Slika 16. Prikaz stranice za registraciju	49
Slika 17. Stranica za prijavu kvarova ili problema	51
Slika 18. Prikaz oznake na karti.....	53
Slika 19. Popis prijava i detalji jedne prijave.....	55

9. Prilozi

9.1. Sadržaj priloženog medija

RB	Datoteka	Sadržaj
1.	diplomski.doc	Tekst diplomskog rada
2.	geo.tar.gz	Komprimirana datoteka sa sadržajem praktičnog projekta
3.	geodb.sql	Struktura baze podataka
4.	konfiguracija.tar.gz	Konfiguracijske datoteke

10. Životopis

European
curriculum vitae
format



Osobne informacije

Ime **Pavlović, Tomislav**
Adresa **A. Stepinca 25, 42250 Lepoglava, Hrvatska**
Telefon **098 745 655**
E-pošta **tpavlovic@gmail.com**

Državljanstvo Hrvatsko

Datum rođenja 27. 12. 1983.

Radno iskustvo

- Datum (od – do) siječanj 2009. -
- Naziv i sjedište tvrtke zaposlenja Životopis d.o.o., Ilica 510, 10090 Zagreb
- Vrsta posla ili područje PHP i Python programer, administrator baza podataka i web poslužitelja

- Datum (od – do) veljača 2006. – siječanj 2009.
- Naziv i sjedište tvrtke zaposlenja FS d.o.o., Hrvatske bratske zajednice 4, 10000 Zagreb
- Vrsta posla ili područje C# i JavaScript programer, MSSQL baza podataka

Obrazovanje

- Datum (od – do) listopad 2005. – rujan 2008.
- Naziv i vrsta obrazovne ustanove Tehničko veleučilište u Zagrebu
- Osnovni predmet /zanimanje Stručni studij informatike
 - Naslov postignut obrazovanjem spec. ing. bacc. inf.
- Datum (od – do) rujan 1998. – lipanj 2002.
- Naziv i vrsta obrazovne ustanove Srednja škola Ivanec, opća gimnazija

Ostale vještine

*Stečene radom/životom,
karijerom, a koje nisu
potkrijepljene potvrdama i
diplomama.*

Strani jezik

- sposobnost čitanja
- sposobnost pisanja
- sposobnost usmenog
izražavanja

Tehničke vještine i sposobnosti

*S računalima, posebnim vrstama
opreme, strojeva, itd.*

Engleski

Da.

Da.

Da.

Programiranje u jezicima:

Python

C#

JavaScript

PHP

Rad s bazama podataka:

MSSQL

MySQL

PostgreSQL

Pisanje opisnog koda u (X)HTML, XML i CSS jezicima.

Održavanje poslužiteljskih računala (web, mail, dns).

Rad s grafičkim alatima Adobe Illustrator, Adobe Fireworks,
Adobe Flash, Adobe Photoshop.

Vozačka dozvola

Da, B kategorije.