Matematički fakultet
Univerzitet u Beogradu

Tema master rada:

Razvoj aplikacije “Spotty”, za internet pretraživače i mobilne uređaje sa android operativnim sistemom korišćenjem programskih jezika Ruby i Java

Mentor:
doc. dr Miroslav Marić

Student:
Ivan Bajalović

U Beogradu, Novembar 2013.

SAŽETAK

"Spotty" je aplikacija koja pomaže korisniku da pronađe ugostiteljske objekte u svojoj blizini, i da se upozna sa ponudama tih objekata.

Aplikacija je razvijena za internet pretraživače i za mobilne uređaje sa Android operativnim sistemom. Aplikacija za internet pretraživače je razvijena korišćenjem programskog jezika Ruby i njegovog najpopularnijeg alata za razvoj internet aplikacija Ruby on Rails. U radu će biti opisana arhitektura alata Ruby on Rails, kao i primena svojstava ovog alata u komunikaciji aplikacije za mobilne uređaje sa internet aplikacijom.

Za korisnike mobilnih uređaja izabran je trenutno najpopularniji operativni sistem za mobilne uređaje, Android. Do septembra 2012. godine je aktivirano preko 500 miliona uređaja sa ovim operativnim sistemom, što čini oko 75% ukupnog svetskog tržišta. U oktobru 2012. godine bilo je oko 700,000 aplikacija za ovaj operativni sistem, koje su skinute oko 25 milijardi puta sa Androidove internet prodavnice. Detaljnije informacije i aktuelne podatke možete pronaći na [1].

SADRŽAJ

1. PREDGOVOR

Od početka korišćenja personalnih računara i interneta pa do danas prošlo je malo više od 40 godina. Kompjuteri su postali deo svakodnevnice i veoma nam je teško da poverujemo da je nekada bilo drugačije.

Pre samo 15 godina, u Srbiji su mobilni telefoni bili retkost i luksuz, a danas skoro svaki čovek poseduje bar jedan mobilni telefon.

Zahvaljujući ekspanziji tehnologije, savremen čovek je navikao da su mu informacije blizu. Ako ne zna kako se rešava kvadratna jednačina, pretražiće Google i dobiće odgovor. Ako ne zna ko je glumio u prvoj verziji Bond filma, pretražiće Google i dobiće odgovor.

Popularizacijom društvenih mreža kao što su Facebook i Twitter, korisnici interneta su naučili da "slušaju" jedni druge. Korisnici su počeli da pridaju visok značaj utiscima i komentarima svojih (virtuelnih) prijatelja i da se vode njihovim utiscima.

Aplikacija "Spotty" koristi intenzivan kontakt i razmenu informacija između korisnika u cilju nalaženja relevantnih podataka o ugostiteljskim objektima i njihovim ponudama.

2. UVOD

2.1. O aplikaciji

Razmotrimo sledeći scenario.

Turista na proputovanju stiže u Beograd i želeo bi da večera. Ali, on ne poznaje Beograd, ne poznaje restorane, i nije upoznat sa ponudama tih restorana. Uz pomoć “Spotty” aplikacije, koju može pokrenuti na svom mobilnom uređaju, on može da pretraži restorane u krugu od 30 minuta pešaka od mesta gde se trenutno nalazi, koji služe špagete.

Aplikacija, na osnovu njegove lokacije i informacija koje je uneo, pretraži bazu podataka i prikaže rezultate. Za svaki restoran može da pogleda ponudu tog restorana, može da vidi lokaciju restorana na mapi kao i upute kako da do tamo stigne peške.

Kada uspešno pronađe mesto koje mu odgovara, ima mogućnost da podeli svoje utiske o tom mestu, da postavi slike, cene jela i pića, znajući da će na taj način pomoći sledećem korisniku koji se nađe u sličnoj situaciji.

Na ovaj način korisnici jedni drugima preporučuju restorane, njihove lokacije i ponude tih restorana.

2.2. Internet pretraživači

Iako mobilni uređaji preuzimaju primat, internet pretraživači su i dalje veoma bitan deo interneta. Veliki broj vlasnika internet sajtova ne želi da razvija aplikacije za različite platforme (Android, iOS, Bada,..) već koriste sve popularniji i jednostavniji pristup, a to je pravljenje internet sajtova koji imaju fleksibilan dizajn i koji se prilagođava različitim dimenzijama ekrana.

Na taj način, internet sajt će predstaviti iste informacije bilo da se otvori sa kompjutera ili sa internet pretraživača mobilnog uređaja. Ovaj pristup nije nova tehnologija, nije novi programski jezik već jednostavna kombinacija postojećih tehnologija HTML, Javascript i CSS-a.

Osnova svakog savremenog internet sajta jeste HTML (u kombinaciji sa JavaScript-om i CSS-om). Međutim, za dinamičko popunjavanje sadržaja, za komunikaciju sa drugim servisima, programeri biraju razne skript ili programske jezike. Zajedničko za sve njih jeste to

da na kraju svaki proizvodi isti izlaz – a to je HTML. Najpopularniji skript jezik za izradu internet sajtova jeste PHP (pokreće oko 80% sajtova), a prate ga ASP, Python, Ruby...

U poslednjih par godina, sve popularniji postaju jezici poput Python, Ruby (Ruby on Rails), Scala...

Za razvoj aplikacije "Spotty" korišćen je programski jezik Ruby (odnosno njegov alat za razvoj Ruby on Rails), koji će biti detaljnije opisan u sekciji 3. Ruby i Ruby on Rails.

2.3. Mobilni uređaji

Mobilni uređaji su mali lagani uređaji koji najčešće imaju ekran osetljiv na dodir, i/ili minijaturnu tastaturu. Najpoznatija podvrsta su mobilni telefoni, čiji najnoviji modeli se često nazivaju pametnim telefonima (smartphones). Neki od proizvođača pametnih telefona su Samsung, Apple, LG, HTC, Motorola. U mobilne uređaje takođe spadaju i tablet računari. Oni se najčešće sastoje samo od ekrana osetljivog na dodir.

Pametni telefoni i tablet računari, pored ekrana osetljivog na dodir, najčešće poseduju i USB priključke, Wi-Fi prijemnike, Bluetooth, kameru, GPS, ... Ovi uređaji poseduju i operativni sistem (Android, iOS, Symbian, Bada, ...) što im omogućava instaliranje različitih aplikacija.

Veliku popularnost su stekli svojim modernim dizajnom, dobrim karakteristikama (brzi procesori, ekrani visoke rezolucije, kamere sa preko 5 megapiksela rezolucije,..), izdržljivim baterijama (koje mogu da traju i do 20 sati bez dopunjavanja) i malom težinom (tablet računari su obično lakši od jednog kilograma, dok su telefoni lakši od 200 grama). Detaljniju podelu mobilnih uređaja možete pronaći na [2].

Sve veći broj korisnika mobilnih uređaja i njihove navike da koriste aplikacije radije nego internet pretraživače, dovele su do ekspanzije u svetu programiranja. Svaki operativni sistem ima svoje alate za razvoj (za Android operativni sistem se najčešće koristi Java programski jezik, za iOS objektno orijentisani C, ..). Ovi alati za razvoj se nazivaju prirodni (engl. native). Pored njih, postoje i takozvani hibridni alati kao što su jQuery Mobile, PhoneGap, Motorola Rhodes,...

3. Ruby i Ruby on Rails

3.1. Programski jezik Ruby

Ruby je objektno orijentisani programski jezik. U sebi kombinuje sintaksu inspirisanu jezicima Perl i Ada, sa objektno orijentisanim osobinama sličnim Smalltalk-u. Ruby je jednoprolazni interpretirani jezik.

Tvorac Ruby programskog jezika je japanac Jukihiro Macumoto (*Yukihiro Matsumoto*). Sa razvojem je počeo 1993. godine, a prva verzija je objavljena 1995. godine. Dugo vremena je Ruby bio čisto japanska pojava i cela dokumentacija je bila napisana samo na japanskom jeziku. Tek krajem devedesetih godina, Macumoto je počeo sa aktivnostima popularizacije ovog jezika van Japana. Veoma brzo je prihvaćen i ubrzo su objavljeni mnogobrojni članci u stručnim časopisima, i urađena je dokumentacija i na drugim jezicima.

Macumoto je izjavio da je Ruby napravljen tako da programeri povećaju produktivnost i da se zabave. On naglašava da dizajn sistema mora da se prilagođava čoveku, a ne čovek sistemu:

> *"Ljudi se često, pogotovo kompjuterski inžinjeri, fokusiraju na mašinu. Oni misle 'Ako napišem ovako, mašina će raditi brže. Ako napišem ovako, mašina će raditi efektivnije. Ako napišem ovako, mašina će nešto, nešto, nešto'. Oni se fokusiraju na mašinu. Ali ustvari, moramo da se fokusiramo na ljude, kako ljudi brinu o programiranju ili o korišćenju mašina. Mi smo gospodari. Mašine su robovi".*

Vodeći se ovom logikom, u Ruby programskom jeziku, promenljive su netipizovane. Programer ne mora da vodi računa da li inicijalizuje broj, nisku, niz,... Druga važna stvar vezana za Ruby jeste ta da se sve posmatra kao objekat. I broj i niska i niz, sve su to objekti i sa svakim od njih se radi isto.

Neke od osobina Ruby programskoj jezika su:

1) Jednostavna i čitljiva sintaksa;

2) Čisto objektno orijentisani jezik (sve je objekat - i broj i niska i klasa...);

3) Netipizovane varijable;

4) Obrada izuzetaka;

5) Automatsko oslobađanje nepotrebno zauzete memorije;

6) Jedinstven interfejs za pristup različitim bazama podataka;

7) Mogućnost i funkcionalnog i proceduralnog programiranja.

Detaljnije o Ruby programskom jeziku možete pronaći na [3].

Primeri:

1) Klasičan *Zdravo svete* primer:

```
puts "Zdravo svete!"
```

Blok 1. Primer Zdravo svete

2) Jedna od najbitnijih osobina je princip “*sve je objekat”*:

```
# Komentari pocinju sa hash tag-om (#)
-199.abs                        # 199
"Zdravo svete!".length          # 13
"Moj master rad".index("j")     # 2

a = [1, 'zdravo', 3.14, 1, 2, [4, 5]]
a.length                        # 5
a[2]                            # zdravo
a.reverse                       # [[4, 5], 2, 1, 3.14, 'zdravo', 1]
```

Blok 2. Rad sa objektima

3) Veoma je bitna razlika između nizova i asocijativnih nizova (koji se nazivaju hash-ovi u Ruby-ju):

```
hash = { :voda => 'mokra', :vatra => 'topla' }
hash[:vatra]                    # topla

hash.each do |key, value|       # for petlja
   puts "#{key} je #{value}"    # odstampace voda je mokra
                                # vatra je topla
end

hash.delete :voda               # uklanja mokra iz niza
```

Blok 3. Niz i Hash (asocijativni niz)

4) Klase:

```
      class Osoba
        attr_reader :ime, :godine   # atributi klase

        def initialize(ime, godine)
          @ime, @godine = ime, godine
        end

        def <=>(osoba)              # operator poredjenja koji se koristi za
sortiranje
          godine <=> osoba.godine
        end

        def to_s                          # to string
          "#{ime} (#{godine})"
        end

      end

      grupa = [                     # ovo je niz objekata
        Osoba.new("Bojana", 23),
        Osoba.new("Marija", 22),
        Osoba.new("Ana", 30)
      ]

      puts group.sort.reverse

      # Ovo ce ispisati sledece:

      Ana (30)
      Bojana (23)
      Marija (22)
```

Blok 4. Definisanje klase i metoda unutar klase

3.2. Ruby on Rails

Ruby on Rails, ili često samo Rails, je alat za razvoj internet aplikacija izrađen na Ruby programskom jeziku. David Hajnemer Henson (David Heinemeier Hanson) je izdvojio Ruby on Rails iz svog projekta Basecamp. Prvu verziju Rails-a, David je objavio u Julu 2004. godine kao otvoreni kod (open source project).

Rails naglašava korišćenje već ustaljenih i prihvaćenih programerskih šablona i principa, kao što su Active Record, Convention over Configuration, Don't repeat yourself, MVC (Model - View - Controller), REST, koji će biti ukratko opisani u nastavku a više možete videti u [4].

Active Record šablon je jedan od ustaljenih šablona za uspostavljanje komunikacije sa relacionom bazom podataka. U Rails-u Active Record se koristi kao standardni model u MVC arhitekturi. Svaki red tabele se predstavlja kao objekat nad kojim je moguće pozvati metod iz predefinisane klase ActiveRecord. Korišćenjem ActiveRecord šablona, programer ne mora da vodi računa o samom SQL upitu, nego za njega to obavlja sam ActiveRecord.

Na primer, komanda:

```
User.find_by_email("test@example.com")
```

će biti prevedena u SQL upit:

```
SELECT * FROM `users` WHERE `users.email` = "test@example.com"
```

Ili, komanda:

```
user = User.new
user.firstname = 'Ivan'
user.lastname = 'Bajalovic'
user.save

# u Rails-u je moglo i mnogo krace da se napise ovaj iskaz
# user = User.new(:firstname => 'Ivan', :lastname => 'Bajalovic').save
# ili
# user = User.create(:firstname => 'Ivan', :lastname => 'Bajalovic')
```

će biti prevedena u SQL upit:

```
INSERT INTO `users` (`firstname`, `lastname`)
       VALUES ('Ivan', 'Bajalovic')
```

Iz prethodnog primera, se može primetiti da klasa `User`, koja, kada se prevede u SQL upit, pretražuje tabelu `users`. Ovo je moguće zahvaljujući šablonu *Convention over Configuration*, odnosno kodiranje po konvenciji. U Rails-u postoje konvencije koje, ako ih se programer pridržava, dosta olakšavaju programiranje. Treba imati u vidu da Rails pravi razliku između velikih i malih slova, tako da promenljive `User` i `user` smatra za dve različite promenljive.

DRY (Don't Repeat Yourself) princip, odnosno *Ne ponavljaj svoj kod*, je princip koji za cilj ima smanjenje ponavljanja informacija bilo koje vrste. Ovo je veoma korisno kod višeslojnih arhitektura. Kada se DRY princip pravilno primeni, kasnija modifikacija bilo kog elementa sistema ne zahteva promenu u drugom, logički nezavisnom, elementu.

3.2.1. MVC Arhitektura

MVC arhitektura (Model - View - Controller) odnosno *Model - Pogled - Kontroler*, je jedna od najpopularnijih višeslojnih arhitektura.

Model se sastoji od podataka, biznis pravila, logike i funkcija nad podacima. Model koji je korišćen u gore pomenutom primeru `User`, će posmatrati tabelu `users` i zahtevaće datoteku `user.rb` u folderu `app/models/`. U ovoj datoteci se definišu i metode za obradu podataka, koje se kasnije mogu pozivati nad tim modelom.

Pogled je izlazna informacija i može biti u različitim formatima (HTML, JSON, xml,..). U Railsu je moguće organizovati informacije posebno za svaki format koji se zahteva. Ekstenzija datoteka je `.[format].erb`. U praksi je najčešće u pitanju HTML format (`.html.erb`), ali teoretski ovo može biti bilo koji format. Ove datoteke se nalaze u `app/views/[controller_name]/` i imaju iste nazive kao i akcije odgovarajućeg kontrolera u čijem su sastavu.

Kontroleri se bave obradom ulaznih informacija, prosleđuju ih modelima i na kraju šalju ih na poglede. U kontroleru može da bude jedna ili više akcija. One su definisane kao metode unutar klase. Akcije unutar kontrolera odgovaraju na tačno jedan specifičan zahtev od strane internet pretraživača. Kada kontroler primi zahtev, on će proveriti da li postoji odgovarajuća datoteka za zahtevanu akciju, koja je potrebna za ispis pogleda, u direktorijumu `app/views/[controller_name]/[action_name].html.erb`.

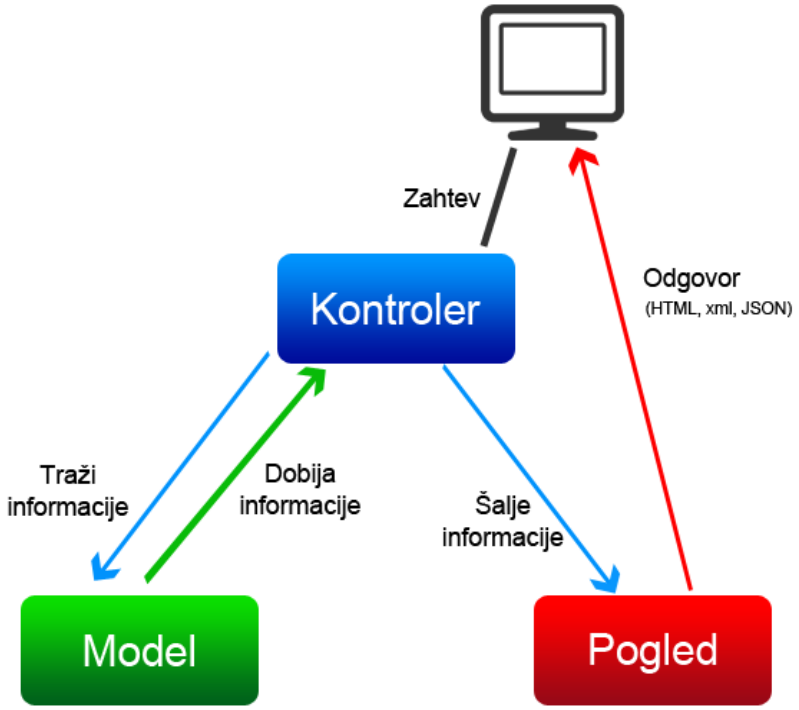
Rails ima posebnu komponentu, *ruter*. Ruter sadrži informacije o kontrolerima i njihovim akcijama, kao i metodama na koje treba da odgovore. Drugim rečima, ukoliko kontroler i njegova akcija nisu definisani u rutama, Rails neće znati kako da odgovori na takav zahtev servera i izbaciće grešku.

```
      # primer definisanja ruta
      # prilikom definisanja rute, nije potrebno navesti koje formate zelite da
podrzite

      get "users/:id"   # u skladu sa REST, ovo ce se mapirati sa show
                        # akcijom u users kontroleru

      post "users/"     # ovo ce se mapirati sa create akcijom u users kontroleru
      get  "users/"     # ovo ce se mapirati sa index akcijom u users kontroleru
```

Blok 5. Definisanje ruta

Rails podstiče programere da koriste rute u skadu sa principima REST arhitekture, kao što su: *create, new, show, edit, update, destroy* i *index*.

Komunikaciju između 3 sloja MVC arhitekture možete videti na Slici 1. Detaljnije možete pročitati o Rails i MVC arhitekturi u [4].



Slika 1. Komunikacija između slojeva u MVC arhitekturi

3.2.2. REST Arhitektura

REST (Representational State Transfer) je stil arhitekture za distribuirane sisteme kao što je internet. REST se izdigao i postao je jedan od najbitnijih stilova za komunikaciju različitih sistema. REST je razvijen od strane W3C konzorcijuma uporedo sa razvojem HTTP 1.1 verzije, a zasnovano na tada postojećem HTTP protokolu verzije 1.0. Više o REST arhitekturi možete pronaći u [5] i [6].

REST arhitektura se koristi dobro definisanim i dobro poznatim metodama i principima HTTP protokola, kao što su:

1) GET (uzmi);

2) PUT (stavi);

3) POST (pošalji);

4) DELETE (izbriši).

Važan koncept u REST arhitekturi je postojanje resursa (odnosno izvora specifičnih informacija), i svaki resurs ima jedinstveni identifikator (npr. URI u HTTP protokolu). Kako bi manipulisali ovim resursima, komponente mreže (npr. internet pretraživači i serveri) komuniciraju preko standardnog protokola (npr. HTTP) i razmenjuju reprezentacije tih resursa. Tako, bilo koja aplikacija može komunicirati sa resursom znajući samo tri stvari:

1) Identifikator resursa (adresu, odnosno URI resursa);

2) Akciju;

3) Format izlazne informacije (da li je u pitanju xml, json, html, csv, ...).

RESTful internet servis je internet servis zasnovan koristeći se HTTP protokolom i principima REST arhitektura. Servis predstavlja kolekciju resursa sa četiri definisana aspekta:

1) Adresa internet servisa (npr. http://www.example.com/api);

2) Internet tip medija koji su podržani od strane internet servisa (xml, html, json, jpeg..);

3) Skup operacija koje su podržane (GET, PUT, POST, ili DELETE);

4) Servis mora biti zasnovan na hipertekstu.

Za razliku od internet servisa zasnovanih na SOAP protokolu, za RESTful internet aplikacije ne postoji standard. Zašto? Zato što je REST arhitekturalni stil a SOAP je protokol.

4. Android i Java

4.1 Android operativni sistem

Android je operativni sistem zasnovan na Linux operativnom sistemu. Razvoj je započela kompanija Android Inc, koju je u početku finansijski podržavao Google da bi je kasnije i kupio 2005. godine. Dve godine kasnije, Android je otkriven javnosti, a prva prodaja telefona zasnovanog na Android operativnom sistemu je bila u oktobru 2008. godine.

Treba pomenuti da je Android open-source projekat (projekat otvorenog koda) i kao takav omogućava proizvođačima da ga prilagode i menjaju. Zahvaljujući ovakvom vidu licenciranja privukao je veliku pažnju programera koji su počeli da razvijaju različite aplikacije. Do maja 2013. godine, aktivirano je 900 miliona uređaja sa Android operativnim sistemom. Google je napravio i internet prodavnicu za Android aplikacije nazvanu Google Play, sa koje su aplikacije preuzete i instalirane 48 milijardi puta.

4.2 Razvoj aplikacija za Android

Najpopularniji alat za razvoj aplikacija je *Android SDK* zasnovan na Java programskom jeziku, ali pored njega postoje i alati za razvoj na C ili C++ programskim jezicima. Android SDK zasnovan na Java programskom jeziku sadrži sveobuhvatan skup razvojnih alata uključujući Debugger (otklanjanje grešaka), softverske biblioteke, emulator Android uređaja, dokumentaciju, primere koda i tutorijale. Sve informacije potrebne za preuzimanje, instaliranje i podešavanja možete pronaći na [7].

Zvanično razvojno okruženje je Eclipse za koji se može preuzeti Android Development Tools - ADT (Android razvojni alati) dodatak.

Zanimljivo je pomenuti da se aplikacije na Android uređajima pokreću u Sandbox-u, odnosno u izolovanom delu sistema koji nema pristup sistemskim resursima, čime se omogućava sigurnost i privatnost korisnika. Da bi aplikacija mogla da pristupa resursima poput internet konekcije, kamere, GPS lokacije, programer mora eksplicitno da zahteva pristup za svaki resurs. Pre instaliranja aplikacije na Android uređaj, korisniku se prikazuje spisak svih resursa kojima će ta aplikacija pristupati.

Pokretanjem novog Android projekta, kreira se određena hijerarhija direktorijuma i predefinisanih klasa. Android je veoma striktan i zahteva ovu hijerarhiju, što dosta programera smatra prilično ograničavajućim. Na primer unutar “res/drawable/” direktorijuma (koji uglavnom sadrži slike) ne mogu se praviti novi poddirektorijumi. Zbog toga programeri nazivima datoteka dodeljuju prefikse koji olakšavaju pretragu direktorijuma. Na primer za ikonice se koriste prefiksi “ic_”, za XML fajlove aktivnosti “activity_” itd.

Direktorijumi koji se nalaze u hijerarhiji novog projekta su:

1) src (sadrži klase koje programer kreira);
2) gen (automatski se kreira od strane Eclipse i Android razvojnog alata i menja se pri svakom čuvanju projketa. U njemu se nalaze klase BuildConfig.java i R.java koji sadrže podešavanja aplikacije i spisak promenljivih definisanih u projektu);
3) assets;
4) bin (sadrži iskompajliranu verziju aplikacije);
5) libs (sadrži privatne biblioteke i eksterne biblioteke koje korisnik želi da pridruži projektu);
6) res.

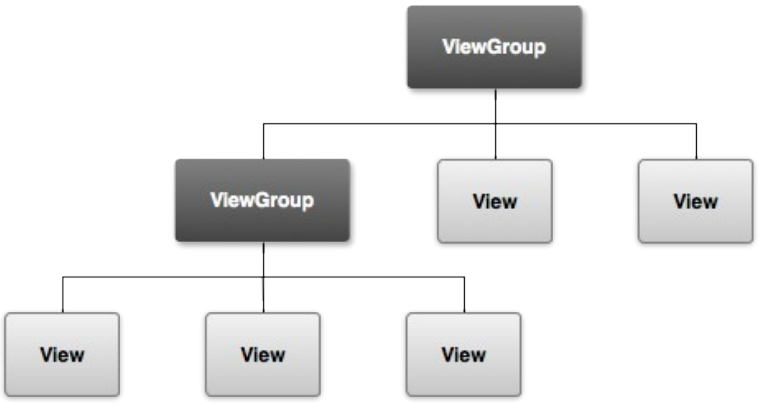
Unutar *res/* direktorijuma se nalaze poddirektorijumi kao što su:

1) anim (sadrži XML datoteke koji definišu animacije)
2) color (XML datoteke koji definišu boje);
3) drawable (sadrži jpg, png ili gif slike, XML datoteke koji dešinišu objekte – krug, dugme, i XML datoteke koji definišu objekte sa nekoliko različitih stanja – dugme pritisnuti, dugme nije

pritisnuto, dugme ima fokus);
4) layout (sadrži XML datoteke koji definišu strukturu različitih ekrana aplikacije);
5) menu (sadrži XML datoteke koji definišu navigaciju aplikacije).

Grafički prizak Android aplikacije se gradi pomoću hijerarhije View i ViewGroup objekata. *View* objekti su obično elementi poput dugmadi, polja za unos teksta, slika itd. *ViewGroup* objekti su nevidljivi kontejneri koji definišu kako se pozicioniraju View objekti unutar grupe...

Najčešće korišćeni ViewGroup objekti su *LinearLayout* (vertikalno ili horizontalno pozicioniranje) i *RelativeLayout*. Kada se koristi LinearLayout, View objekti unutar ove grupe mogu biti ili horizontalno ili vertikalno pozicionirani. Ukoliko je izabrano vertikalno pozicioniranje onda će svi objekti unutar grupe biti pozicionirani jedan ispod drugog (ne može se podesiti da u prvom redu bude jedan element, zatim u drugom dva, pa u trećem opet jedan itd). Za razliku od ove grupe, RelativeLayout omogućava relativno pozicioniranje svakog elementa (elementi se pozicioniraju jedan u odnosu na drugi).



Slika 2. Hijerarhija View i ViewGroup objekata

View i ViewGroup objekti se smeštaju unutar aktivnosti (eng Activity – strana koju korisnik vidi kada upali aplikaciju). Svaka aplikacija mora imati bar jednu aktivnost, i za nju ne postoji univerzalno ime – iako se preporučuje da to bude “activity_main”. Definisanje aktivnosti, View i ViewGroup objekata je ustvari XML datoteka i ona se po Android konvencijama nalazi unutar direktorijuma “res/layout”. Odgovarajuća Java klasa se zove MainActivity.java i nalazi se unutar src/ direktorijuma.

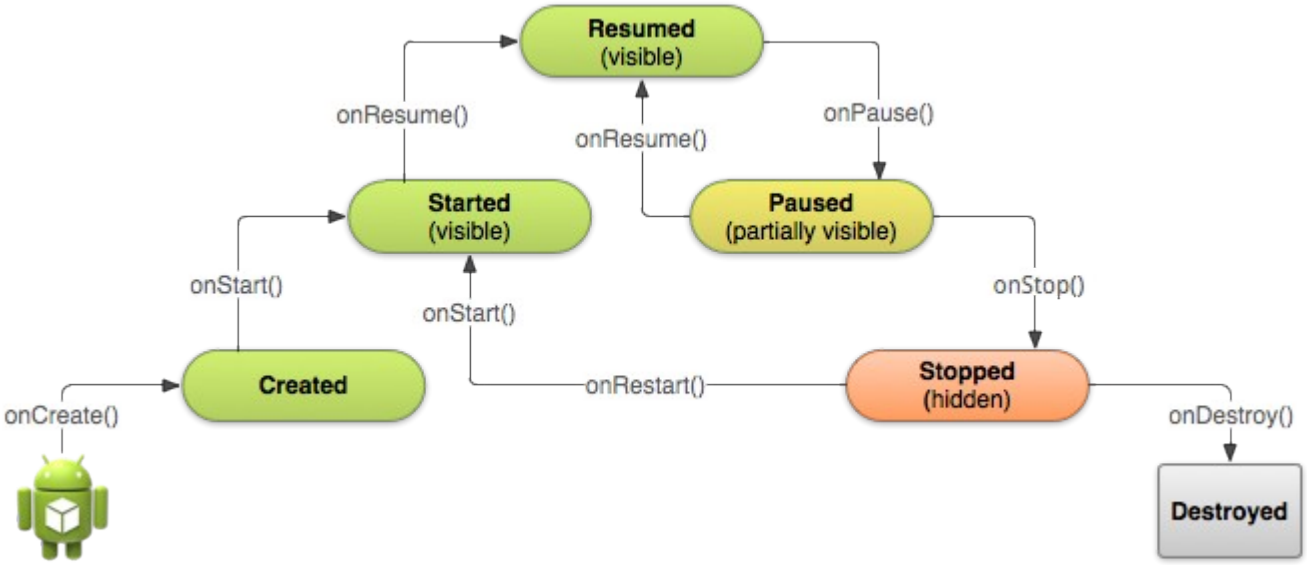
```
 <!-- Definisanje ViewGroup objekta RelativeLayout i definisanje atributa ovog
objekta poput sirine, visine, boje pozadine itd -->
  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
```

```
      android:background="#eeeeee"
      android:focusable="true"
      android:focusableInTouchMode="true"
      tools:context=".HomeActivity" >
  <!-- nacin definisanja View objekta, polja za unos teksta i njegovih atributa
-->
      <EditText
          android:id="@+id/et_query_string"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_alignBottom="@+id/bttn_search"
          android:layout_alignParentLeft="true"
          android:layout_alignParentTop="true"
          android:layout_toLeftOf="@+id/bttn_search"
          android:background="#ffffff"
          android:hint="@string/hint_search"
          android:padding="10dp"
          android:inputType="text"
          android:singleLine="true"
          android:textSize="12sp"
          android:imeOptions="actionSearch"
          />
  <!-- nacin definisanja View objekta, dugmeta i njegovih atributa -->
      <Button
          android:id="@+id/bttn_search"
          style="@style/TextShadow"
          android:layout_width="48dp"
          android:layout_height="wrap_content"
          android:layout_alignParentRight="true"
          android:layout_alignParentTop="true"
          android:background="#f1470c"
          android:drawableLeft="@drawable/ic_search"
          android:padding="10dp"
          android:text="@string/bttn_search"
          android:textColor="#ffffff"
          android:textSize="14sp"
          android:textStyle="bold" />
  <!-- nacin definisanja ViewGroup objekta, liste koja ce prikazivati
       rezultate pretrage -->
      <ListView
          android:id="@+id/lv_places"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_alignParentBottom="true"
          android:layout_below="@+id/et_query_string"
          android:layout_centerHorizontal="true"
          android:layout_margin="10dp"
          android:divider="#00000000"
          android:dividerHeight="10dp" >
      </ListView>
  </RelativeLayout>
```

Blok 6. Sadržaj activity_home.xml datoteke koja predstavlja definiciju forme za pretragu i liste za prikaz rezultata

Slika 3. Grafički prizak aktivnosti za pretragu lokacija

Za razliku od Java aplikacija, aktivnosti u Android razvojnom alatu nemaju čuvenu *main()* metodu, već metodu koja se naziva *onCreate()* i koja je definisana u klasi *android.app.Activity*. Svaka aktivnost nasleđuje klasu Activity iz paketa android.app. Pored ovog postoje i druge metode koji se mogu zameniti (koristeći notaciju *@Override*) kao što su *onResume()*, *onDestroy()*, *onPause()*. Na slici 4 možete videti životni ciklus jedne aktivnosti.



Slika 4. Životni ciklus aktivnosti u Android aplikaciji

Razvoj Android aplikacije se svodi na kreiranje grafičkog prikaza aktivnosti pomoću XML datoteka koje se nalaze unutar *res/layouts/* direktorijuma i kreiranjem pratećih Java klasa za svaku aktivnost.

Jedna od najvažnijih datoteka u Android projektu jeste *AndroidManifest.xml.* U njemu se definišu permisije za željene resurse (kameru, Internet, GPS), aktivnosti koje se koriste u aplikaciji, verzija aplikacije, minimalna i maksimalna verzija podržanih android operativnih sistema.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- paket Java klasa koje pripadaju projektu, i broj verzije -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bajalovic.spotty"
    android:versionCode="1"
    android:versionName="1.0" >

<!-- Minimalna i ciljana SDK Verzija. Verzija 9 odgovara Android 2.3, a SDK
verzija 17
     odgovara Android 4.2 operativnom sistemu -->
    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="17" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

     <!-- Resursi koje “Spotty” aplikacija zahteva za rad -->
```

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <permission
        android:name="com.bajalovic.spotty.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />

    <uses-permission
android:name="com.bajalovic.spotty.permission.MAPS_RECEIVE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <uses-library android:name="com.google.android.maps" />

         <!-- definisanje glavne akcije, koja se zove LoginActivity -->
        <activity
            android:name="com.bajalovic.spotty.LoginActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Spisak ostalih aktivnosti koje se koriste u aplikaciji -->
        <activity
            android:name="com.bajalovic.spotty.NoInternetActivity"
            android:label="@string/title_activity_no_internet" >
        </activity>
        <activity
            android:name="com.bajalovic.spotty.HomeActivity"
            android:label="@string/title_activity_home" >
        </activity>
        <activity
            android:name="com.bajalovic.spotty.MapActivity"
            android:label="@string/title_activity_map" >
        </activity>
  </aplication>
</manifest>
```

Blok 7. Deo AndroidManifest.xml datoteke preuzet iz “Spotty” aplikacije

5. RAZVOJ “SPOTTY” INTERNET APLIKACIJE

Za izradu "Spotty" internet aplikacije korišćen je Ruby on Rails. Razvojem aplikacije na ovoj platformi dobijena je ne samo internet prezentacija, već i API (Application Programming Interface - Interfejs za programiranje aplikacija) koji je kasnije iskorišćen za komunikaciju aplikacije za mobilne uređaje sa Android operativnim sistemom i internet aplikacije. Više o komunikaciji između Internet i Android aplikacije možete videti u *odeljku 6*.

Za skladištenje podataka aplikacija koristi *PostgreSQL* bazu podataka. Glavni Rails modeli koji se koriste u radu aplikacije su: Category, Checkin, Place, Price, User i CategoriesPlace. U skladu sa Rails konvencijama, modeli su napisani u jednini dok su nazivi tabela u bazi u množini. Sledeće tabele u bazi odgovaraju gore pomenutim modelima: categories, checkins, places, prices, users i categories_places.

Category model sadrži informacije o kategorijama (lokacije se grupišu u određene kategorije). Relacija koja postoji u ovom modelu jeste `has_and_belongs_to_many :places`, što znači da jedna kategorija može imati više lokacija, ali i da neka lokacija može biti u više kategorija. U trenutnoj verziji aplikacije, svaka lokacija pripada samo jednoj kategoriji, i to odgovara postavljenoj relaciji has_and_belongs_to_many. U budućnosti je planirano da određene lokacije mogu pripadati različitim kategorijama, npr. jedan ugostiteljski objekat može imati i restoran i kafić kao dve razdvojene celine.

Place model sadrži informacije o samim lokacijama. Ovaj model skladišti naziv lokacije, njenu geolokaciju (zapisanu u koordinatama latitude i longitude), adresu, grad i državu. Relacije koje postoje u ovom modelu su `has_and_belongs_to_many :categories` i `has_many :prices`. Pored što pripada različitim kategorijama, ova lokacija ima još jednu relaciju, i to prema modelu Prices. Jedna lokacija može imati više cena (cene su artikli, odnosno ponude te lokacije).

Veze između modela Category i Place se skladište u tabeli categories_places i njoj odgovara model *CategoriesPlace*.

User model sadrži informacije o korisnicima aplikacije, njihovo ime i prezime, email adresu, šifru, ...

Price model sadrži informacije o ponudama lokacija. Svaki zapis u ovom modelu može da sadrži naziv, korisnikov utisak ili detaljniji opis ponude, cenu, sliku... Relacija `belongs_to :place` označava da su informacije u ovom modelu usko vezane za lokaciju.

Checkin model služi kao veza izmedju korisnika i lokacija. Ona pamti kada se i koji korisnik prijavio na određenoj lokaciji. Skladištenje ovih informacija omogućava kreiranje

različitih statistika o korisnikovim navikama, posećenosti određenih lokacija itd...

Kada se prati konvencija vezana za nazive modela i tabela i kada se naprave relacije između modela, onda je veoma jednostavno dobijanje potrebnih informacija.

Ispod možete videti neke primere kako pomoću relacija između modela možete dobiti potrebne informacije:

```
# Primer: 1
# Za lokaciju sa poljem ID (identifikator) 3 nadji sve cene
Place.find(3).prices.all

# Primer: 2
# prijavljivanje korisnika na odredjenoj lokaciji
place_id = params[:p] # preuzmi identifikator lokacije
place = Place.find(place_id)
# pronadji lokaciju sa zadatim identifikatorom

# current_user je instanca klase User i sadrzi informacije o
# trenutno prijavljenom korisniku
current_user.checkins.create place_id: place.id

# u zavisnosti od formata zahteva, definise se izlazna informacija
respond_to do |format|
# ukoliko je html, onda redirektuj na rutu za prikaz zadate lokacije
  format.html { redirect_to place_path(place) }
# ukoliko je json, onda ispisi u JSON formatu informacije o lokaciji
  format.json { render json: place }
end
```

Blok 8. Dobijanje potrebnih informacija pomoću relacija između modela

Predstavićemo neke od akcija koje korisnik može koristiti na internet aplikaciji i prikazati kod koji to omogućava.

5.1 Pretraga objekata

Korisnik na ovoj stranici može da izabere maksimalnu distancu koju bi želeo da prepešači (15, 30, 45 minuta ili sat vremena šetnje), i da izabere samo određenu kategoriju (ukoliko želi kafu, onda je verovatnije zainteresovan za kafić nego za restoran). Početna podešavanja su: za distancu 15 minuta šetanja i sve kategorije.

Pretraga lokacija u izabranom radijusu se vrši tako što se od korisnikove trenutne lokacije prvo odredi kvadrat, takav da je centar kvadrata korisnikova lokacija (u koordinatama latitude/longitude) a dužina ivice je radijus/2. Na ovaj način dobijemo donje levo i gornje desno teme kvadra, što nam je dovoljno za restrikciju nad tabelom `places`. Zatim, pravi se restrikcija tabele `places` i traže se lokacije čije koordinate pripadaju gore definisanom kvadratu. U sledećoj iteraciji se za svaki rezultat računa distanca izmedju koordinata rezultata i korisnikove trenutne lokacije. Distance koje su veće od zadate, odnosno koordinate koje pripadaju ćoškovima kvadra se odbacuju. Grafički prikaz možete pogledati na slici 5.

Slika 5. Restrikcija tabele uz pomoć kvadrata i kruga

Ukoliko želi konkretno piće ili obrok, može i to da unese u pretragu i u tom slučaju rezultati pretrage sadržaće i cene traženog pića odnosno obroka. Ovo sa korisnikove strane deluje veoma jednostavno, ali se u pozadini dešava mnogo više. Deo koda možete videti u Bloku 9.

```
    # preuzimanje vrednosti promenljivih iz POST zahteva
    radiusDistance = params[:r] # distanca za setanje
    lat = params[:la]           # latitude korisnikove lokacije
```

```
      lng = params[:ln]           # longitude korisnikove lokacije
      cat = params[:c]            # izabrana kategorija

      # poziva se pomocni metod koji racuna latitude/longitude koordinate donjeg
      # levog i gornjeg desnog ugla kvadrata
      @squarePlaces = nil
      findCorners lat, lng, radiusDistance

      # ukoliko je izabrano “Sve kategorije” onda se ovaj uslov ne ukljucuje u
      # pretragu, u suprotnom se ukljucuje
      if cat.to_i == 0
        @squarePlaces = Place.all(:include => :cats,
          :conditions => ["lat > ? AND lat < ? AND long > ? AND long < ? ",
            @lowerLeft[0], @topRight[0], @lowerLeft[1], @topRight[1]])
      else
        @squarePlaces = Place.all(:include => :cats,
         :conditions => ["cats.id = ? AND lat > ? AND lat < ? AND long > ? AND
long < ? ",
            cat, @lowerLeft[0], @topRight[0], @lowerLeft[1], @topRight[1]])
      end

 # nakon sto smo dobili sve lokacije unutar kvadrata, izbacujemo one koje ne
 # pripadaju krugu
 @places = []
 place_ids = [];
 @squarePlaces.each do |place|
   if( distance_between([lat.to_f, lng.to_f], [place.lat, place.long]) <
      radiusDistance.to_f )
     place.category_id = place.cats.first.id
     @places.push(place)
     place_ids.push(place.id)
   end
 end

 # ukoliko je korisnik uneo zeljeno pice ili obrok, vrsi se PostgreSQL pretraga
 # celog teksta
 prices = []
 if place_ids.any?
   prices = Price.tsearch_query(params[:q].downcase, 25, place_ids)
 else
   prices = []
 end

 # ukoliko je korisnik uneo zeljeno pice ili obrok, i pronadjene su lokacije koje
 # zadovoljavaju rezultate pretrage, tada se one skladiste u novi niz koji je
 # asocijativno povezan sa spiskom lokacija
 prices_by_place   = {}
 if prices.length > 0
   prices.each do |price|
     prices_by_place[price.place_id] = [] unless prices_by_place.has_key?
                                                              price.place_id
     prices_by_place[price.place_id] << price
   end
```

```
 end

respond_to do |format|
      format.json { render json: {places: newPlaces, prices: prices_by_place } }
end
```

Blok 9. Pretraga i sortiranje rezultata pretrage

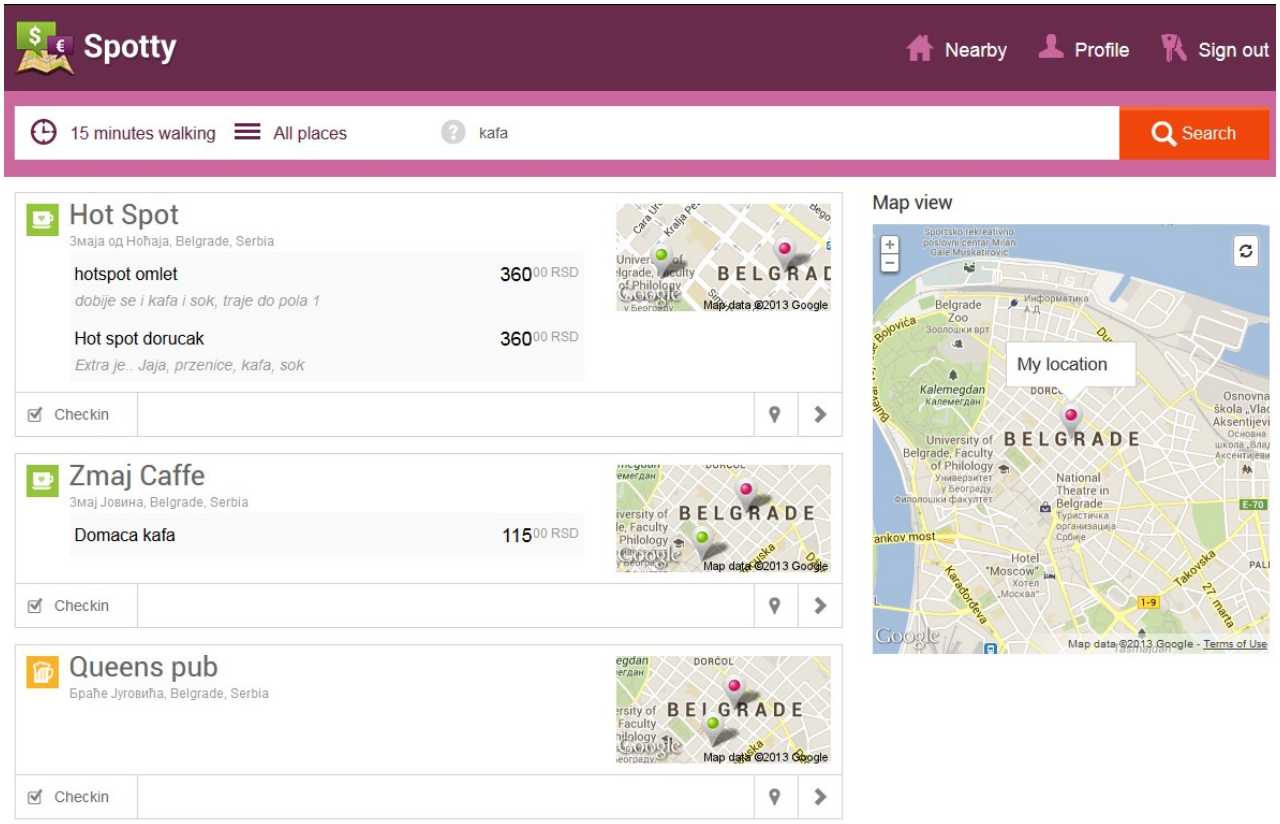
Kao što se može primetiti pretraga vraća informacije samo u JSON formatu. Rezultat pretrage u JSON formatu izgleda ovako:

```
{
"places":{
      # ID lokacije i informacije o toj lokaciji
      "15":{
            "address":"3маја од Ноћаја",
            "city":"Belgrade",
            "country":"Serbia",
            "created_at":"2013-02-14T10:39:01Z",
            "id":15,
            "lat":"44.820385149888175",
            "long":"20.457285940647125",
            "name":"Hot Spot"
            },
      "16":{
            "address":"Браће Југовића",
            "city":"Belgrade",
            "country":"Serbia",
            "created_at":"2013-02-16T14:30:43Z",
            "id":16,
            "lat":"44.81845165664683",
            "long":"20.460497215390205",
            "name":"Queens pub"
            },
      .........
      },
"prices":{
      # ID lokacije i niz koji sadrzi cenovnik te lokacije
      "15":[
            {
            "name":"hotspot omlet",
            "description":"dobije se i kafa i sok, traje do pola 1 ",
            "created_at":"2013-02-14T10:39:54Z",
            "currency":"RSD",
            "id":21,
            "price":"360.0",
            },
            {
            "name":"Hot spot dorucak",
            "description":"Extra je.. Jaja, przenice, kafa, sok",
            "created_at":"2013-04-11T10:11:10Z",
            "currency":"RSD",
            "id":77,
```

```
            "price":"360.0"
            }
            ]
      }
}
```

Blok 10. Rezultati pretrage formatirani u JSON formatu

Grafički prikaz rezultata pretrage možete videti na slici 6. Za svaki objekat, korisnik ima informacije o nazivu objekta, adresi, mapu sa iscrtanom pozicijom gde se nalazi objekat u odnosu na korisnikovu lokaciju i ukoliko je uneo željeno piće ili obrok, cenovnik za ponude koje sadrže traženi pojam.



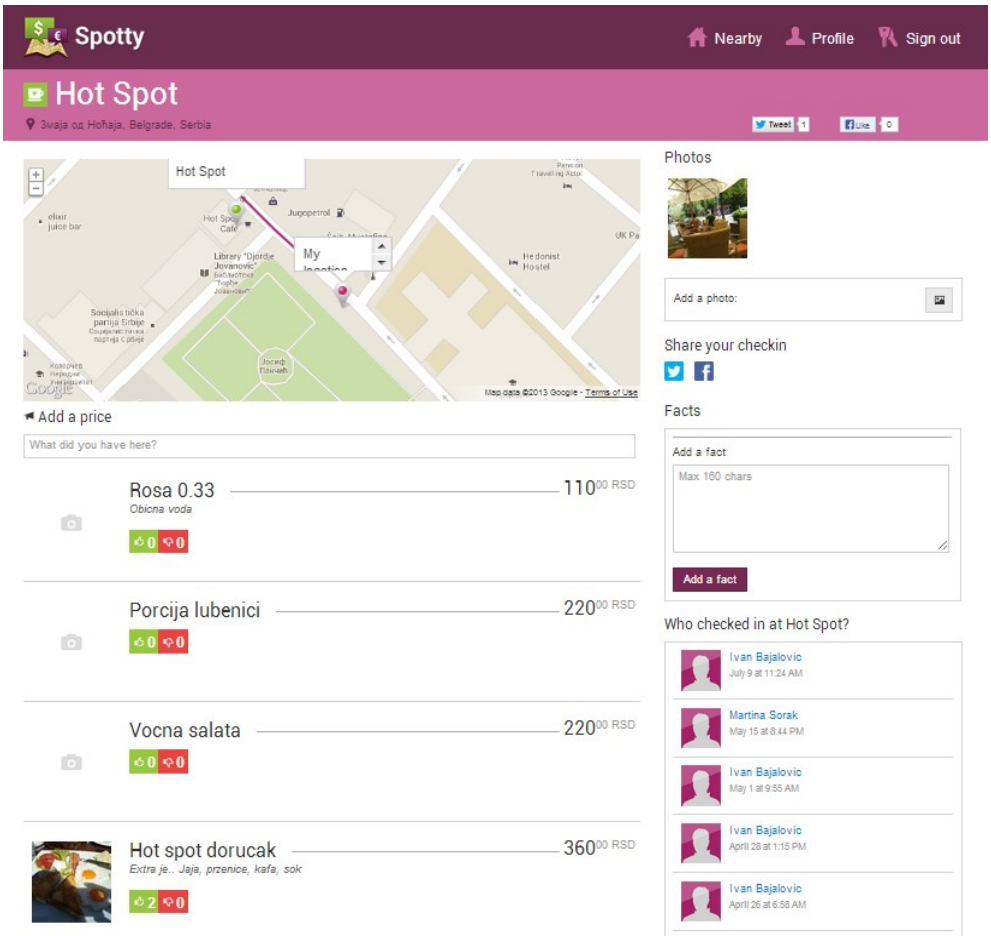
Slika 6. Grafički prikaz rezultata pretrage korisnika internet aplikacije “Spotty”

5.2 Prikaz pojedinačnog objekta

Bez umanjenja opštosti, uzećemo za primer ugostiteljski objekat “Hot Spot”. Prezentaciju “Hot Spot” objekta možete videti na slici 7.

Informacije koje su dostupne na prezentaciji objekta su:

1) Naziv i adresa objekta (ulica, grad, država);
2) Mapa sa iscrtanom putanjom kako korisnik da peške dođe od svoje trenutne lokacije do objekta, uz informaciju kolika je distanca koju treba da prepešači i aproksimaciju vremena koje je potrebno da se ta distanca prepešači;
3) Cenovnik objekta;
4) Galeriju slika objekta;
5) Činjenice (opšte informacije koje ostavljaju drugi korisnici a mogu biti generalan utisak o objektu, radno vreme, ...);
6) i spisak korisnika koji su prijavili da su posetili objekat.



Slika 7. Internet prezentacija objekta “Hot Spot”

Ono što se nalazi u pozadini ove strane jeste Ruby on Rails akcija `show` unutar kontrolera `places` (Blok 11).

```
    @place = Place.find(params[:id])
    # spisak osoba koje su prijavile svoje prisustvo
    @checkins = Checkin.where("place_id = ?", params[:id]).
                        order("created_at DESC").
                        limit("30").
                        includes(:user)
    # cenovnik
    @prices = @place.prices.order("created_at DESC")

    # spisak cinjenica
    @facts = @place.facts.order("created_at desc ").all
    # spisak slika, galerija
    @place_photos = @place.place_photos.order("created_at DESC").all

    .... # ovde se nalaze while petlje koje iteriraju spiskove slika, cena,
cinjenica
    .... # i pravi asocijativne nizove kako bi se lakse koristile informacije
    .... # kada se zahteva JSON format

    respond_to do |format|
      format.html { } # sve promenljive koje pocinju sa @ su dostupne u pogledu,
tako da
                      # nema potrebe da se prosledjuju ovde
      format.json { render json: {prices: prices_json,
                                  photos: photos_json,
                                  facts: facts_json}
                  }
    end
```

Blok 11. Prikaz pojedinačnog objekta

Dodavanje nove cene je jako jednostavno u Ruby on Rails-u.

```
     place = Place.find(params[:price][:place_id])
      # kreiranje nove stavke u cenovniku
      @Price = Price.create( params[:price] )
      @Price.user_id = current_user.id
      @Price.save
      # ukoliko je zahtev za HTML formatom, korisnik ce biti preusmeren na
prezentaciju
      # objekta i ispisace se poruka o uspesnosti
      respond_to do |format|
        format.html { redirect_to place_path(place),
                      :notice => "You added a price to #{place.name}" }
      # ukoliko je zahtev za JSON formatom, onda ce se samo prikazati informacije
      # o objektu, u JSON formatu naravno
        format.json { render json: place }
```

```
    end
```

Blok 12. Dodavanje nove cene

6. RAZVOJ “SPOTTY” ANDROID APLIKACIJE

Gore je pomenuto da je izbor Ruby on Rails alata za razvoj, kao “neželjeni efekat” proizveo i interfejs za razvoj aplikacija – API. U prethodnoj glavi mogli ste da primetite da akcije odgovaraju na 2 različita zahteva za formatiranje: HTML i JSON.

Ukoliko je internet adresa za prezentaciju objekta “Hot Spot”: http://spotty.rs/places/15, Ruby on Rails će ovo tretirati kao zahtev za HTML formatiranje. Ukoliko na kraj dodamo ekstenziju `.json` (URL postaje http://spotty.rs/places/15.json) Rails će to tretirati kao JSON zahtev i izlazne informacije biće predstavljene u JSON formatu. Java programski jezik sadrži klase za obradu JSON formata i nalaze se u klasi `org.json`.

Zahvaljujući Ruby on Rails-u, sve potrebne informacije za Android aplikaciju su sada dostupne kroz postojeće URL-ove uz jednostavno dodavanje .json na kraju svakog URL-a. Programer ne mora da piše nove akcije za iste informacije koje želi da dobije i to je velika prednost ovog alata za razvoj, i u skladu je sa principima Ruby programskog jezika – Do Not Repeat Yourself (Ne ponavljaj već napisano).

Sve metode za dobijanje informacija potrebnih za Android aplikaciju se nalaze u Java klasi `ApiClient.java`. Ispod je navedeno nekoliko metoda iz ove klase kao i primeri kako se dobijene informacije obrađuju i predstavljaju korisniku.

```
	/*
	Pretraga objekata na osnovu korisnikove lokacije i podesenih parametara
(distance,
	kategorije, i termina za pretragu – zeljeno pice ili obrok)
	 */
	public JSONObject searchPlacesAndArticles(Double lat, Double lng,
			String radius, String category, String query) throws
JSONException {

		HttpPost post = new HttpPost(
				"http://spotty.rs/price/search.json");
		List<NameValuePair> nameValuePair = new ArrayList<NameValuePair>(6);
		// String token potreban za proveru da li je korisnik prijavljen na
sistem
		nameValuePair.add(new BasicNameValuePair("auth_token", authToken));
		nameValuePair.add(new BasicNameValuePair("la", lat + ""));
		nameValuePair.add(new BasicNameValuePair("ln", lng + ""));
		nameValuePair.add(new BasicNameValuePair("r", radius));
		nameValuePair.add(new BasicNameValuePair("c", category));
		nameValuePair.add(new BasicNameValuePair("q", query));
		String responseBody = "";
		try {
```

```
                post.setEntity(new UrlEncodedFormEntity(nameValuePair, "UTF-8"));
                post.setHeader("Content-Type",
"application/x-www-form-urlencoded");
          } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
          }

          // Pravljenje HTTP zahteva
          JSONObject prices = null;
          try {
                HttpResponse response = client.execute(post);
                int responseCode = response.getStatusLine().getStatusCode();
                switch (responseCode) {
                case 200:
                      HttpEntity entity = response.getEntity();
                      if (entity != null) {
                            responseBody = EntityUtils.toString(entity);
                            prices = new JSONObject(responseBody);
                      }
                      break;
                }
          } catch (ClientProtocolException e) {
                e.printStackTrace();
          } catch (IOException e) {
                e.printStackTrace();
          }
          return prices;
     }

     /* Metod koji preuzima informacije o prezentaciji */
     public JSONObject getPlaceInfo(String p) throws JSONException {
          HttpGet get = new HttpGet("http://spotty.rs/places/"
                      + p + ".json?auth_token=" + authToken);
          String responseBody = "";

          // Pravljenje HTTP zahteva
          try {
                HttpResponse response = client.execute(get);

                int responseCode = response.getStatusLine().getStatusCode();
                switch (responseCode) {
                case 200:
                      HttpEntity entity = response.getEntity();
                      if (entity != null) {
                            responseBody = EntityUtils.toString(entity);
                      }
                      break;
                case 401:
                      signOut();
                      responseBody = null;
                      break;
                }
          } catch (ClientProtocolException e) {
```

```
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }

            JSONObject jObject = new JSONObject(responseBody);
            return jObject;
      }

      /*
PRETRAGA LOKACIJA U BLIZINI KORISNIKA

Pravljenje HTTP zahteva u Android aplikacijama mora biti asinhrono – odnosno
aplikacija ne moze da ceka da se HTTP zahtev zavrsi i da se onda tek ispise
sadrzaj. Zbog toga se koristi Java Thread pod nazivom AsyncTask (nalazi se u
android.os paketu) */

      new AsyncTask<Boolean, Integer, Boolean>() {
            private ProgressDialog progress;
            private ArrayList<HashMap<String, String>> placesList = new
                                                      ArrayList<HashMap<String,
String>>();
            private ArrayList<ArrayList<HashMap<String, String>>> placePrices = new
                                          ArrayList<ArrayList<HashMap<String,
String>>>();
            private JSONObject allPlaces = null;
            private JSONObject allPrices = null;

            @Override
            protected void onPreExecute() {
                  /* prvo se izvrsava ovaj metod. Podesen je dijalog napredovanja
                     sa porukom da se pretrazuje baza */
                  final String content = getString(R.string.searching_title);
                  progress = ProgressDialog.show(activity, null, content);
                  query = etSearchQuery.getText().toString().trim();
            }
            @Override
            protected Boolean doInBackground(final Boolean... args) {
                  /* drugi metod po redu izvrsavanja je doInBackground. Unutar ove
                     metode nije moguce pristupati View ili ViewGroup objektima.
                     Poziva se metoda searchPlacesAndArticles iz klase apiClient
koja
                     ce vratiti objekat tipa JSONObject koji unutar sebe sadrzi dva
                     objekta JSONObject tipa – Sve lokacije koje zadovoljavaju
                     kriterijume i spisak cena koje zadovoljavaju kriterijume.
                  */
                  try {
                        JSONObject places = apiClient.searchPlacesAndArticles(
                                          myLat, myLong, distance, category,
query);
                        allPlaces = places.getJSONObject("places");
                        allPrices = places.getJSONObject("prices");
                  } catch (JSONException e) {
```

```
                        e.printStackTrace();
                  }
                  return true;
            }
            @Override
            protected void onPostExecute(final Boolean result) {
                  /* treci metod koji se izvrsava unutar AsyncTask-a je
onPostExecute.
                     Unutar ove metode sortiracemo i pripremiti rezultate iz dva
                     JSONObject objekta: allPlaces i allPrices, i na kraju popuniti
listu
                     sa rezultatima */
                  Iterator<?> keys = allPrices.keys();
                  int[] usedPlaces = new int[allPrices.length()];
                  int j = 0;
                  while (keys.hasNext()) {
                        String placeId = (String) keys.next();
                        JSONObject placeInfo = null;
                        ArrayList<HashMap<String, String>> placePrice = new
                                          ArrayList<HashMap<String, String>>();
                        try {
                              placeInfo = allPlaces.getJSONObject(placeId);
                              if (placeInfo != null) {
                                    usedPlaces[j] = Integer.parseInt(placeId);
                                    j++;
                                    HashMap<String, String> map = new
HashMap<String,
                                                                        String>();
                                    map.put("id", placeInfo.getString("id"));
                                    map.put("name", placeInfo.getString("name"));
                  /* ... popunjavanje HashMap objekta sa informacijama o lokaciji
*/
                                    placesList.add(map);
                              }
                        } catch (JSONException e2) {
                              e2.printStackTrace();
                        }
                        JSONArray place;
                        try {
                              place = allPrices.getJSONArray(placeId);
                              for (int i = 0; i < place.length(); i++) {
                                    JSONObject price;
                                    try {
                                    price = place.getJSONObject(i);
                                    HashMap<String, String> priceMap = new
                                                                  HashMap<String,
String>();
                                    priceMap.put("id", price.getInt("id") + "");
                                    priceMap.put("name",price.getString("name"));
                  /* ... popunjavanje HashMap objekta sa informacijama o cenovniku
*/
                                    placePrice.add(priceMap);
                                    } catch (JSONException e) {
```

```
                                        e.printStackTrace();
                                    }
                                }
                                placePrices.add(placePrice);
                            } catch (JSONException e1) {
                                e1.printStackTrace();
                            }
                        }
                        /* prvo smo popunjavali lokacije koje sadrze trazeni pojam, a
sada
                           popunjavamo listu sa ostalim lokacijama koje nemaju traženi
pojam
                           ali su u blizini korisnika */
                        Iterator<?> places = allPlaces.keys();
                        while (places.hasNext()) {
                            String placeId = (String) places.next();
                            if (!inArray(usedPlaces, Integer.parseInt(placeId))) {
                                JSONObject pl;
                                try {
                                  pl = allPlaces.getJSONObject(placeId);
                                  HashMap<String, String> map = new HashMap<String,
                                                                          String>();
                                  map.put("id", pl.getString("id"));
                                  map.put("name", pl.getString("name"));
                        /* popunjavanje HashMap objekta sa informacijama o lokaciji */
                                  placesList.add(map);
                                } catch (JSONException e) {
                                  e.printStackTrace();
                                }
                            }
                        }

                        ListView myListView = (ListView)
activity.findViewById(R.id.lv_places);


      /* LazyAdapter je klasa koja prosiruje klasu BaseAdapter iz android.widget
paketa.
         Ukoliko lista sadrzi 10 lokacija, a na ekranu mogu da se vide samo 3
lokacije
         (zavisno o dimenzijama ekrana mobilnog uređaja), tada ce lista izbrisati
ostalih
         7 lokacija i na taj nacin osloboditi memoriju. Tek kada korisnik prstom
pomera
         listu, sledeca npr. cetvrta lokacija ce se kreirati u listi, a prva ce se
         izbrisati iz liste. Moja klasa LazyAdapter prihvata JSONObject objekte
koji
         sadrze rezultate pretrage i brinuce se o popunjavanju liste i ustedi
         memorije.
      */
                        lazyAdapter = new LazyAdapter(activity, placesList, placePrices);
                        lazyAdapter.setListView(myListView);
                        lazyAdapter.setMyLatLong(myLat + "", myLong + "");
```

```
                lazyAdapter.setApiClient(accessToken);
                myListView.setAdapter(lazyAdapter);

                progress.dismiss();
            }
        }.execute();
```

Blok 13. Metode iz klase ApiClient.java

Princip koji je korišćen za prikaz rezultata pretrage je isti za prikaz svih ostalih informacija unutar “Spotty” Android aplikacije. Jedina razlika između nekih metoda je što jedni popunjavaju liste (Blok 14), a druge metode direktno menjaju View objekte unutar aktivnosti (Blok 15).

```
 public class FactAdapter extends BaseAdapter {

    private Activity activity;
    private ArrayList<HashMap<String, String>> data;
    private static LayoutInflater inflater = null;
    protected ListView listView;
    ImageLoader imageLoader;

    public FactAdapter(Activity a, ArrayList<HashMap<String, String>> d) {
          activity = a;
          data = d;
          inflater = (LayoutInflater) activity
                      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
          imageLoader = new ImageLoader(activity);
    }

    public void setListView(ListView lv) {
          listView = lv;
    }

    public int getCount() {
          return data.size();
    }

    public Object getItem(int position) {
          return position;
    }

    public long getItemId(int position) {
          return position;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
          View vi = convertView;
          if (convertView == null)
                vi = inflater.inflate(R.layout.row_list_fact, null);
          /*
          Svaka stavka liste je View objekat vi, i pomocu njega pristupamo drugim
          View objektima unutar njega, kako bismo mogli da prikazemo potrebne
          informacije
          */
          TextView tvFactFact = (TextView) vi.findViewById(R.id.tv_fact_fact);
          TextView tvFactAuthor = (TextView)
vi.findViewById(R.id.tv_fact_author);
          TextView tvFactDate = (TextView) vi.findViewById(R.id.tv_fact_date);

          HashMap<String, String> fact = new HashMap<String, String>();
          fact = data.get(position);

          // Setting all values in listview
          tvFactFact.setText(fact.get("text"));
          tvFactAuthor.setText("Added by: " + fact.get("user"));
```

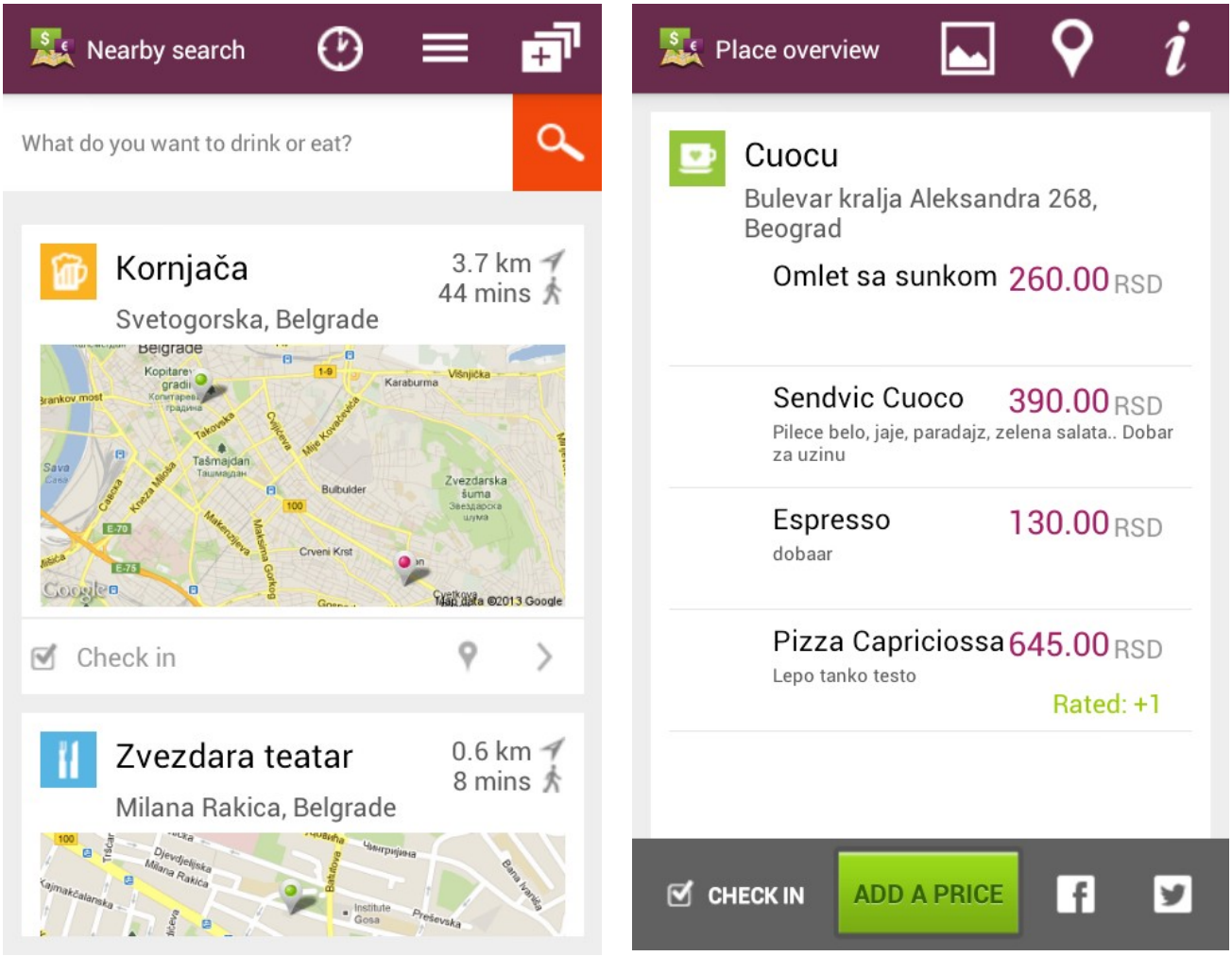
```
            DateFormat dateFormat = new SimpleDateFormat(
                        "yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault());
            Date created_at = new Date();
            try {
                  created_at = dateFormat.parse(fact.get("updated_at"));
            } catch (ParseException e1) {
                  e1.printStackTrace();
            }
            String newTimeString = new SimpleDateFormat("HH:mm",
                        Locale.getDefault()).format(created_at);
            String newDateString = new SimpleDateFormat("dd MMM yyyy",
                        Locale.getDefault()).format(created_at);

            tvFactDate.setText("Added at: " + newTimeString + ", " +
newDateString);
            // ako se pogleda kod za formatiranje datuma i vremena, u Javi je
potrebno
            // 9 linija koda. U Rails-u je dovoljna jedna:
            // price.created_at.strftime(“%d %b %Y”)
            return vi;
      }
}
```

Blok 14. FactAdapter klasa pomoću koje se popunjava lista činjenica za određeni objekat

```
      /* Da bi se moglo promeniti vrednost nekog View objekta prvo je potrebno
napraviti instancu tog objekta */
      TextView nazivObjekta = getActivity().findViewById(R.id.tv_naziv_objekta);
      // TextView je View objekat
      /* Mozemo videti kako se koriste identifikatori koji se automatski skladiste
u R klasi na osnovu XML fajlova iz res/ direktorijuma */
      nazivObjekta.setText(“neki tekst”);
      /* kada imamo instancu, onda mozemo da koristimo metode za podesavanje
teksta ili da promenimo neke od atributa kao sto su velicina slova, boja slova
itd... */
```

Blok 15. Izmena teksta View objekta direktno iz aktivnosti

Na slici 8 možete videti kako izgledaju rezultati pretrage i prikaz prezentacije na Android aplikaciji.



Slika 8. Rezultati pretrage i prezentacija lokacije na “Spotty” Android aplikaciji

Dobro organizovana struktura internet aplikacije je omogućila da se mobilna aplikacija svede na jednostavnu klijent aplikaciju koja preuzima informacije sa unapred definisanih URL-ova, obradu i prikaz tih informacija.

Na ovaj način ni u jednom trenutku mobilna aplikacija ne komunicira direktno sa bazom podataka tako da je zaštita ostala na visokom nivou i za to se takođe brine internet aplikacija. Ovde je akcenat stavljen na “mobilna aplikacija” a ne na “Android aplikacija” jer je sada moguće razviti aplikacije i za druge operativne sisteme poput Apple iOS, Ubuntu Mobile, Winows Phone koristeći postojeći API..

7. Zaključak

Trendovi i moderne navike ljudi širom planete, omogućili su da aplikacije kao što je "Spotty" pomognu najrazličitijim korisnicima. Sve više turista preko interneta rezerviše i organizuje svoja putovanja, prave planove ne samo za smeštaj i znamenitosti koje treba posetiti, već i o restoranima i kafićima gde se mogu odmoriti, uživati, hraniti...

Imajući u vidu mnogobrojnost ugostiteljskih objekata u svakom gradu, oni su za sada jedini objekti koje će se nalaziti u aplikaciji. U budućnosti kategorije objekata biće proširene tako da obuhvataju i različite znamenitosti (muzeje, biblioteke, parkove, spomenike, ...) tako da i turisti koji nisu do detalja isplanirali svoje rute mogu u realnom vremenu lako dobiti informacije šta je u njihovoj blizini, kao i informacije o cenama ulaznica, radnom vremenu, trenutnim postavkama, izložbama...

Ideja "Spotty" aplikacije je da korisnik jednostavno i brzo dođe do željenih informacija. Iz tog razloga pretraga lokacija je svedena samo na željenu distancu koju je korisnik voljan da pređe i eventualno da filtrira rezultate po kategorijama ili određenom pojmu. Da bi ovo bilo moguće, potreban je jedan siguran, stabilan i fleksibilan sistem i to su neki od razloga za izbor Ruby on Rails alata.

Zahvaljujući MVC i REST arhitekturi aplikacije i dobijenog interfejsa za programiranje aplikacija (API) u budućnosti je moguće pravljenje aplikacija za druge operativne sisteme i platforme što će privući još više korisnika.

8. Literatura

[1] Android Operativni sistem, Wikipedia, *wikipedia.org/wiki/Android_(operating_system)*

[2] Mobilni uređaji, Wikipedia, *wikipedia.org/wiki/Mobile_device*

[3] Ruby programski jezik, Wikipedia, *wikipedia.org/wiki/Ruby_(programming_language)*

[4] Sam Ruby, Dave Thomas, David Heinemeir Hansson, *Agile Web Development with Rails (4 edicija)*

[5] Michael Hartl, *Ruby on Rails Tutorial, Learn Web Development with Rails*

[6] Ruby on Rails, Wikipedia, *wikipedia.org/wiki/Ruby_on_Rails*

[7] Android SDK elektronsko uputstvo, *developer.android.com/develop*