Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gorazd Rajar

Mobilna informacijska podpora za udeležence jadralnih regat

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 00149/2013
Datum: 14.09.2013

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

Kandidat: **GORAZD RAJAR**

Naslov: **MOBILNA INFORMACIJSKA PODPORA ZA UDELEŽENCE JADRALNIH REGAT**

MOBILE INFORMATION SYSTEM FOR SAILING REGATTA PARTICIPANTS

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Količina podatkov na svetovnem spletu je ogromna, vendar še vedno obstaja največja težava pomanjkanje povezav med njimi. Delno ta problem rešujejo spletni iskalniki, ki z indeksiranjem spleta poskrbijo za nekoliko lažje iskanje podatkov, vendar to še zdaleč ni rešitev, še posebej na točno specifičnih problemskih domenah. Ena izmed takšnih je problemska domena jadralcev, ki pogosto obiskujejo tekmovanja v tujini in doma. Pred odhodom in med bivanjem pogosto potrebujejo različne informacije, ki so povezane s tekmovanjem, prenočiščem, vremenom ipd. Vsi omenjeni podatki obstajajo na različnih spletnih straneh, vendar niso med seboj povezani. V okviru diplomske naloge je potrebno izdelati delujoči prototip mobilne aplikacije, kjer se naj poskrbi za integracijo podatkov iz različnih spletnih virov ter ponudi funkcionalnosti mobilnemu uporabniku.

Mentor:

doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Gorazd Rajar, z vpisno številko **63090104**, sem avtor diplomskega dela z naslovom:

Mobilna informacijska podpora za udeležence jadralnih regat

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 16. septembra 2013 Podpis avtorja:

Za strokovno pomoč in korekten odnos pri izdelavi diplomske naloge se zahvaljujem mentorju doc. dr. Dejanu Lavbiču.

Zahvalil bi se staršem, ki so me porivali naprej skozi celotno obdobje izobraževanja in mi omogočili miren študij brez ostalih skrbi.

Zahvala gre tudi vsem kolegom iz fakultete s katerimi smo zadnjih nekaj let dobro sodelovali. Tu mislim tako na sošolce kot na profesorje in drugo osebje fakultete.

Kazalo

Slike

Kazalo primerov

Seznam kratic in simbolov

- POJO (Plain Old Java Object) - navaden Javanski objekt, ki ne razširja ali implementira nobenega drugega razreda.
- PaaS (Platform as a service) - platforma kot storitev - storitev, ki nudi strojno in programsko opremo za poganjanje uporabnikovih aplikacij v oblaku.
- SQL (Structured Query Language) - standardiziran jezik za iskanje po podatkovnih bazah.
- NoSQL - označuje tipe podatkovnih baz, katerih ne upravljamo z SQL izrazi.
- GQL (Google Query Language) - jezik za poizvedovanje po Googlovem tipu NoSQL podatkovne baze.
- API (Application Programming Interface) - aplikacijski programski vmesnik določa kako bi morali deli programske opreme sodelovati z drugimi deli. Ponavadi je to knjižnica, ki določa rutine, podatkovne strukture, razrede in spremenljivke.
- HTTP (HyperText Transfer Protocol) - aplikacijski protokol, ki določa izmenjavo podatkov na svetovnem spletu.
- HTTPS (HyperText Transfer Protocol Secure) - izvedenka HTTP protokola, ki uporablja varnostne mehanizme, ki podatke šifrirajo.

- HTML (HyperText Markup Language) - glavni opisni jezik za ustvarjanje spletnih strani in ostalih informacij, ki jih lahko prikaže spletni brskalnik.
- JVM (Java Virtual Machine) - Java virtualna naprava. Del platforme Java, ki je zadolžen za izvrševanje kode.
- Java SE (Java Standard Edition) - običajna platforma Java za razvijanje različnih vrst aplikacij.
- Java ME (Java Micro Edition) - platforma Java namenjena vgrajenim sistemom (mobilni telefoni so eni izmed njih).
- LSP - luščenje spletnih podatkov.
- GPS (Global Positioning System) - satelitski sistem, ki omogoča določanje trenutnega položaja GPS sprejemnika.
- SDK (Software Development Kit) - skupek orodij za razvijanje aplikacij za točno določen programski paket, programsko ogrodje, strojno ogrodje, računalniški sistem, igričarsko konzolo itd.
- IDE (Integrated Development Environment) - integrirano razvojno okolje. Skupek programske opreme namenjene razvijanju nove programske opreme.
- XML (Extensible Markup Language) - označevalni jezik z množico pravil za zapis dokumentov v obliki, ki jo znata prebrati tako človek kot računalnik.
- JSON (JavaScript Object Notation) - vrsta zapisa za shranjevanje in izmenjavo človeku razumljivih tekstovnih podatkov. Služi kot alternativa XML.
- URL (Uniform Resource Locator) - spletni naslov.
- ISAF (International Sailing Federation) - mednarodna jadralna zveza.

Povzetek

Osrednja motivacija diplomske naloge je izdelava mobilne aplikacije, ki bo udeležencem jadralnih regat nudila informacije, ki jih potrebujejo. Sistem, ki smo ga razvili na spletu najde za jadralca relevantne podatke in jih integrira v poenoten pogled na mobilni napravi. Celotna rešitev je sestavljena iz dveh delov. Zaledni del, ki teče na sistemu Google App Engine, s tehniko luščenja spletnih podatkov po spletu periodično zbira podatke o kraju, vrsti, terminu jadralnih regat po svetu in podatke o prenočiščih v bližini regate. Vse podatke shranjuje v podatkovno bazo. Osprednji del, ki ga predstavlja aplikacija za sistem Android tem podatkom doda še vremensko napoved. Vremenska napoved mora biti ažurna, zato se osveži iz spletnega vira vsakič, ko uporabnik mobilne aplikacije to zahteva. Osprednji del prejme podatke iz zalednega sistema in jih skupaj z vremensko napovedjo prikaže v intuitivni, poenoteni obliki.

Ključne besede: integracija podatkov, mobilna naprava, luščenje spletnih podatkov, App Engine, Android

Abstract

The main motivation of this thesis is the development of a mobile application, that would help participants of sailing regattas to get the information concerning the regatta. System that we developed finds relevant data, integrates it into a unified view and displays it on a mobile device. Our solution is composed of two parts - frontend and backend. Backend runs on Google App Engine system. It uses web scraping techniques to periodically collect data from web sources. It finds the basic information about the regatta (location, dates, disciplines) and information about nearby accommodation. Everything is saved into the database. An Android mobile application acts as system's frontend. It receives all the data from the backend and adds weather forecast for the location of the regatta. Weather forecast must be up to date, so the mobile application downloads it directly from the source any time the user needs it. Android application displays all the data in an intuitive, unified view.

Key words: data integration, mobile device, web scraping, App Engine, Android

Poglavje 1

Uvod

Živimo v informacijski dobi, kjer nas podatki preplavljajo z vseh strani, predvsem po zaslugi svetovnega spleta. Svetovni splet zna odgovoriti na mnoga vprašanja, rečemo lahko tudi na večino vprašanj, ki si jih povprečen človek zastavlja. S pomočjo t.i. iskalnih pogonov (*search engines - Google, Yahoo, Bing...*) zna povprečen uporabnik svetovnega spleta sam poiskati relevantne odgovore na svoje vprašanje ter tudi predvideti ali je vir kjer je informacijo našel zanesljiv in s tem informacija točna.

V mnogih primerih uporabnik spleta išče odgovor na več med seboj povezanih vprašanj (npr. kdaj je izbrana restavracija odprta in na kateri avtobus se moram usesti, da do nje pridem). Na svetovnem spletu mora uporabnik pogosto najti vsak odgovor posebej, kar je zamudno. Poleg tega lahko en napačen ali ne najden odgovor uniči smisel celotnega iskanja. Če je uporabnikov, ki potrebujejo odgovor na isti sklop vprašanj, več, je smiselno te podatke povezati oziroma *integrirati* v neko novo obliko, kjer bodo odgovori na ta sklop vprašanj prikazani skupaj v intuitivni, čitljivi obliki.

Integracija podatkov torej pomeni kombiniranje podatkov najdenih na različnih mestih in povezovanje teh podatkov v poenoten pogled, ki je nato na voljo uporabniku. Ta poenoten pogled na podatke pa lahko prikažemo na različne načine.

Eden od načina prikazov podatkov je prikaz na mobilni napravi. Pametni

telefoni in tablični računalniki so v zadnjih letih naredili pravo revolucijo kar se tiče dostopnosti podatkov kjerkoli in kadarkoli. Prikaz digitalnih podatkov vseh oblik se v vedno večji meri seli na mobilne naprave. V času pisanja tega dela je 10% vseh svetovnih ogledov spletnih strani prihajalo iz mobilnih naprav [10]. Ta številka še naprej narašča.

To je bila tudi osrednja motivacija mojega diplomskega dela - povezati različne na spletu samodejno pridobljene podatke in jih prikazati na mobilni napravi. Za ciljno publiko sem si izbral športnike - jadralce, saj sem tudi sam en izmed njih in poznam težave s katerimi se soočajo. Jadralci, ki pogosto hodijo v tujino na tekmovanja (v nadaljevanju *regate*), pred odhodom in tudi med samim bivanjem na kraju dogodka potrebujejo različne informacije, ki se tičejo regate. Prikaz teh integriranih podatkov je zato smiseln na mobilni napravi, ki jo ima jadralec seboj.

Tu se nam že odpirajo nekatera vprašanja. Katere podatke sploh potrebuje jadralec? Ali potrebni podatki obstajajo in ali so dostopni? Kako zagotoviti, da bodo uporabniku na voljo takrat, ko jih bo rabil?

Na vsa ta vprašanja odgovarjajo naslednja poglavja. V drugem poglavju so predstavljene tehnologije, ki so bile uporabljene pri izdelavi diplomske naloge. V tretjem poglavju nato raziščemo problem, s katerim se ukvarjamo, naredimo pregled že obstoječih rešitev in podrobno opišemo našo rešitev. Sledi še četrto poglavje, ki povzame narejeno in poda možnosti za nadaljnje delo.

Poglavje 2

Uporabljeni koncepti in tehnologije

2.1 Programski jezik Java in Eclipse

Programski jezik Java je od leta 2012 najbolj razširjen programski jezik. Gre za objektno usmerjen jezik, ki temelji na razredih. Razvoj Jave je temeljil na naslednjih konceptih:

- preprost, objektno orientiran in poznan
- robusten in varen
- neodvisen od arhitekture in prenosen
- visoko zmogljiv
- tolmačen, dinamičen in podpira niti

Prva različica Jave je izšla leta 1996 v podjetju Sun Microsystems, danes pa nosi oznako Oracle, saj je ta leta 2010 kupil Sun Microsystems.

Eclipse je ta trenutek najbolj priznano integrirano razvojno okolje. Uporablja se predvsem za razvijanje aplikacij v programskem jeziku Java, a se

z mnogimi razširitvami, ki so na voljo, da programirati tudi v večini ostalih priznanih jezikih (*C, C++, JavaScript, PHP, Python, Ruby,...*). Eclipse ima dober sistem razširitev, ki jih izdelujejo tako posamezniki kot podjetja. Razširitve pohitrijo proces razvoja aplikacij. Eclipse je za svojega privzel tudi Google. Proces razvoja aplikacij za njihove sisteme temelji na Eclipsu in Googlovih razširitvah zanj.

2.2 Google App Engine

Google App Engine je storitev oblike PaaS (2.2.1). Google je prvo verzijo za javnost objavil Aprila 2008. Do konca leta 2012 je App Engine nabral že 250.000 uporabnikov.

2.2.1 Platform as a service (PaaS)

Google AppEngine spada med t.i. platforme kot storitev (*Platform as a Service*). To so storitve, ki se nahajajo v oblaku in služijo kot programska platforma na katere lahko uporabnik naloži svojo programsko kodo in jo nato zažene s spletno (HTTP) zahtevo. PaaS storitev zagotavlja vse elemente, ki so potrebni za zaganjanje uporabnikove programske kode. To so operacijski sistem, spletni strežnik, podatkovna baza in podpora nekemu programskemu jeziku.

Ponudnikov PaaS storitev je na trgu več. Dominira Amazonov sistem EC2, AppEngine pa vztrajno pleza po lestvici. Dominanca Amazona izhaja iz tega, da je bil eden izmed prvih, ki je začel ponujati oblačno infrastukturo kot storitev (prva verzija EC2 je izšla že leta 2002). Google je imel leta 2012 20% povečanje števila uporabnikov na področju PaaS in IaaS (*infrastructure as a service*) in obvladuje slabih 5% trga, medtem ko je imel Amazon v istem obdobju 10% povečanje in obvladuje 27% trga [6].

2.2.2 App Engine

Google App Engine velja za eno od najbolj preprostih za uporabo storitev PaaS. Uporabnik svoj App Engine projekt naloži na Googlovo infrastrukturo, ta pa poskrbi za vse ostalo.

Največja prednost pred ostalimi razširjenimi PaaS storitvami je učinkovito samodejno upravljanje s količino prometa (horizontalno skaliranje), ki ga je App Engine aplikacija deležna. To pomeni, da ko naraste število zahtev, ki jih mora aplikacija obdelati, se samodejno povečajo tudi potrebni viri, ki jih aplikacija potrebuje za obratovanje (procesorska moč, spomin, pasovna širina). App Engine posrbi tudi za varnost, internetno povezavo, vzdrževanje strojne opreme in ostale stvari nujno potrebne za poganjanje spletnih aplikacij.

Slabost App Enginea je, da lahko naložimo le projekte prirejene za App Engine, med tem ko nekateri drugi ponudniki ponujajo večjo svobodo pri obliki kode, ki jo naložimo. S tem, ko je sistem manj omejen pa pride tudi več dela za razvijalca, saj mu je pogosto prepuščeno delo z veliko podrobnostmi za katere App Engine poskrbi samodejno.

App engine nudi podporo štirim programskim jezikom: Java, Python imata podporo vseh komponent App Enginea, v preizkusno fazo za nekatere komponente pa so maja 2013 dali tudi PHP in Googlov lasten jezik Go.

Google App Engine je na voljo brezplačno do nekega obsega porabe virov. V praksi to pomeni, da lahko aplikacijo razviješ in sam uporabljaš brezplačno, ko pa jo začne redno uporabljati več ljudi in rahlo naraste število zahtev, vsaj en od kazalcev hitro preseže mejo. Kazalci se ponastavijo enkrat dnevno.

2.2.3 Datastore

Google App Engine za podatkovno bazo uporablja lasten tip NoSQL podatkovne baze, ki so jo poimenovali Datastore.

NoSQL podatkovne baze se uporabljajo v distribuiranih sistemih, kot je Google App Engine, saj so lažje horizontalno skalabilne - spreminjanje,

odstranjevanje, dodajanje novih vozlišč (strežnikov) v distribuirano oblačno strukturo je veliko bolj preprosto, kot pri uporabi SQL podatkovne baze. V taki bazi lahko shranjujemo podatke, ki ne potrebujejo močne konsistence (*strong consistency*).

Datastore zagotavlja:

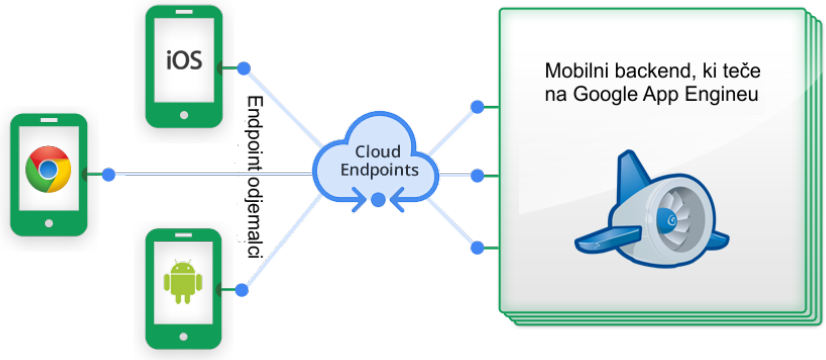
- Močno konsistenco pri direktnem branju in pri poizvedbah po starših
- Zakasnjeno konsistenco (*eventual consistency*) pri vseh ostalih poizvedbah

NoSQL baze zaradi svoje zasnove ne morejo uporabljati poizvedovalnega jezika SQL. App Engineov Datastore namesto tega uporablja jezik GQL, ki je SQL-u po sintaksi zelo podoben, a ne vsebuje funkcij, ki niso primerne za NoSQL podatkovno bazo. Predvsem manjka JOIN operator, katerega uporaba bi bila zelo neučinkovita, če se poizvedba izvaja na različnih strežnikih distribuiranega sistema.

2.2.4 Google Cloud Endpoints

Google Cloud Endpoints so orodja, knjižnice in zmogljivosti, ki omogočajo ustvarjanje API-jev in knjižnic za odjemalce App Engine aplikacije. Kot je vidno na sliki 2.1, so Cloud Endpoints posrednik med strežnikom in odjemalci. Olajšajo razvijanje spletnega zalednega sistema, ki je namenjen mobilnim ali spletnim odjemalcem. Googlova orodja znajo na podlagi App Engine projekta sama ustvariti knjižnice, ki jih potem uporabimo v aplikaciji za Android (knjižnice v jeziku Java), iOS (jezik Objective-C) ali spletni aplikaciji (jezik JavaScript). Z uporabo Cloud Endpoints se izognemo pisanju ovojnic za komunikacijo med zalednim sistemom (App Engine) in osprednjim sistemom (Android, iOS, splet).

Na strani odjemalca z direktnim klicem API funkcije pošljemo zahtevek na App Engine strežnik. Strežnik nato odgovori z nekim objektom, ki se



Slika 2.1: Shema pretoka podatkov preko Cloud Endpoints

samodejno pretvori v obliko primerno za prenos po HTTP protokolu. Samodejno ustvarjene knjižnice na strani odjemalca pa znajo ta odgovor pretvorit nazaj v objekt. Komunikacija preko spleta je za razvijalca skoraj neopazna.

2.3 Android

Android je Googlov odprtokodni celoviti skupek programske opreme za prenosne naprave. Zajema operacijski sistem, sistemski srednji sloj in aplikacije. Operacijski sistem je namenjen predvsem prenosnim napravam z zaslonom na dotik. Ker je Android odprtokoden je najljubša izbira večine proizvajalcev mobilnih naprav, ki želijo cenovno ugoden, prilagodljiv, lahek a še vedno napreden operacijski sistem. Dodano vrednost naštetemu da še okoli 1.000.000 aplikacij (julij 2013), ki so že na voljo na Play Store - uradni trgovini aplikacij za operacijski sistem Android.

V drugem četrtletju leta 2013 je imel Android 79,3% tržni delež [11] (leta 2009 le 2,8%). Na drugem mestu mu s 13% sledi Applov iOS, tretji je Microsoft z Windows Phone (3,9%). Od naštetih je le Android odprtokoden. Podoben delež ima Android tudi med razvijalci aplikacij - 71%.

Android je zastavljen tako, da je razvijanje aplikacij čim bolj preprosto.

Posledica tega in celotne Googlove filozofije je veliko število aplikacij in njihova nizka povprečna cena. 82% aplikacij za Android je brezplačnih, 13 % pa jih stane manj kot $2,5 [12]. Povprečen prenos aplikacije na napravo z operacijskim sistemom Android je uporabnika leta 2013 stal $0,06 [13].

2.3.1 Arhitektura operacijskega sistema

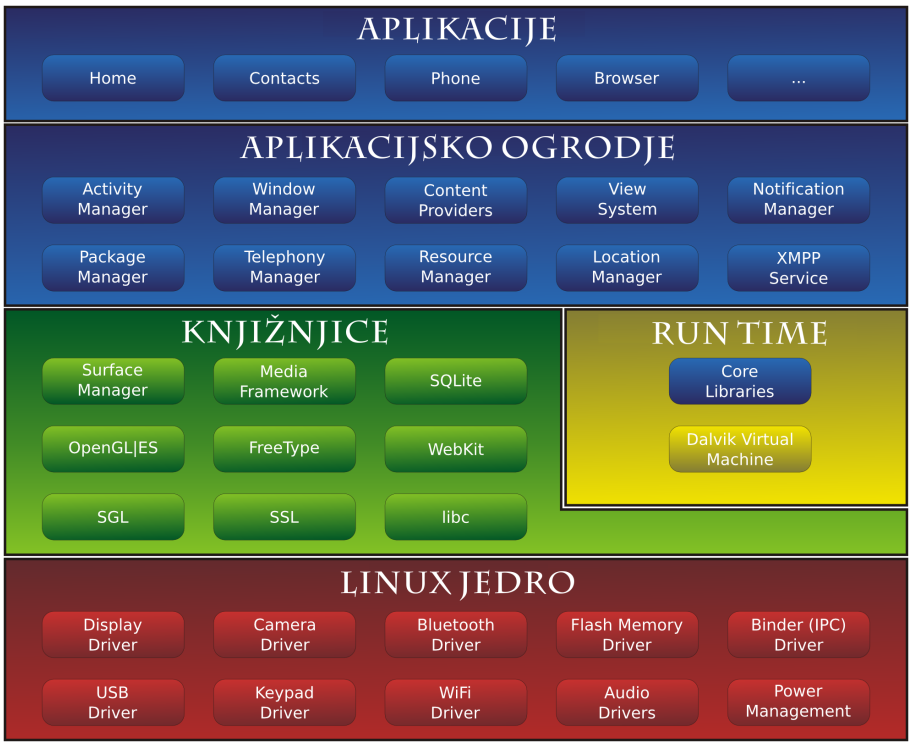
Kot je prikazano na sliki 2.2 arhitekturo Androida ločimo na pet nivojev:

- Linux jedro
- Knjižnice
- Android Runtime
- Aplikacijsko ogrodje
- Aplikacije

Android je torej skupek programske opreme naložene ena na drugo, vsaka od teh plasti ima več komponent. Vsaka plast v arhitekturi zagotavlja določene storitve plasti nad njo.

Linux jedro

Linux jedro je osnovna plast. Celoten operacijski sistem Android je zgrajen na Linux jedru verzije 2.6 z nekaterimi popravki, ki jih je naredil Google. Ta plast sodeluje s strojno opremo in zajema vse osnovne gonilnike (*drivers*). Gonilniki so programi, ki nadzirajo in komunicirajo s strojno opremo. Linux jedro igra vlogo abstraktne plasti med strojno opremo in višjimi plastmi programske opreme. Android uporablja Linux za osnovne funkcionalnosti kot je upravljanje s spominom, upravljanje s procesi, omrežne povezave, varnostne nastavitve itd. Ker je Android zgrajen na priljubljenih in preizkušenih temeljih je prenos sistema na različne mobilne naprave relativno hiter in preprost.



Slika 2.2: Arhitektura sistema Android.

Knjižnice

Naslednja plast v arhitekturi so Androidove knjižnice, ki napravi omogočajo ravnanje z različnimi tipi podatkov. Te knjižnice so napisane v programskih jezikih C in C++ in so specifične za strojno opremo, ki jo ima naprava. Nekatere pomembne knjižnice:

- *Surface Manager* - Odgovoren je za ustvarjanje grafičnih površin v sistemskem spominu in za upravljanje dostopa do njih. Grafične površine (*surfaces*) v Androidu so t.i. *bufferji* bitnih slik in njihovih meta podatkov (širina, višina, format, enolični identifikator) in se uporabljajo kot vhodni podatki komponentam, ki površine sestavljajo skupaj v nek prikaz.
- *Media framework* - Ogrodje, ki zagotavlja različne kodeke, ki omogočajo predvajanje in snemanje različnih multimedijskih formatov.
- *SQLite* - Pogon za podatkovne baze, ki se uporabljajo za shranjevanje podatkov na Android napravah.
- *WebKit* - Pogon za spletni brskalnik. Uporablja se za prikaz HTML vsebine.
- *OpenGL* - Knjižnica za izrisovanje 2D in 3D grafične vsebine na zaslon.

Android Runtime

Android Runtime sestavljata Dalvik Virtual Machine in Core Java libraries.

Dalvik Virtual Machine (DVM) je tip Java Virtual Machine (JVM) virtualne naprave optimiziran za majhno porabo procesorske moči in spomina. Za razliko od JVM Dalvik VM ne poganja *.class* datotek ampak *.dex* datoteke, ki so zgrajene iz *.class* datotek in zagotavljajo bolšjo učinkovitost v okoljih z majhnim obsegom sistemskih virov. Dalvik VM dopušča da je več virtualnih naprav ustvarjenih sočasno kar omogoča dobro varnost, izolacijo, upravljanje s pomnilnikom in podporo nitim.

Java Core Libraries so osnovne Java knjižnice, ki se sicer razlikujejo od tistih definiranih v Java SE in Java ME, a zagotavljajo večino funkcionalnosti definiranih v Java SE knjižnicah.

Aplikacijsko ogrodje

Aplikacijsko ogrodje so gradniki, ki jih naše aplikacije direktno uporabljajo. Za razvijalca so to osnovna orodja s katerimi gradi aplikacije.
Nekateri pomembni gradniki, ki spadajo v aplikacijsko ogrodje:

- *Activity Manager* - Upravlja z življenjskim krogom aktivnosti v aplikacijah.
- *Content Providers* - Zadolženi so za izmenjevanje podatkov med različnimi aplikacijami. Android iz varnostnih razlogov ne dovoljuje direktnega izmenjevanja podatkov med aplikacijami.
- *Telephony Manager* - Upravlja z vsemi telefonskimi klici. Uporabimo ga, če želimo v naši aplikaciji dostopati do telefonskih klicev.
- *Location Manager* - Določanje trenutne lokacije preko GPS-a ali glede na oddajnike telefonskega signala.
- *Resource Manager* - Upravljanje z različnimi tipi virov, ki jih uporabljamo v naši aplikaciji

Aplikacije

Aplikacije so najvišji sloj v Androidovi arhitekturi in so tisto kar vidi in uporablja uporabnik naprave. Nekaj standardnih aplikacij je že prednaloženih na vse Android naprave (aplikacije za telefonske klice, SMS sporočila, spletni brskalnik, upravljalec kontaktov,...), ostale aplikacije pa namestijo proizvajalci naprav ali pa uporabnik sam.

2.3.2 Razvijanje aplikacij za Android

Razvijalec rabi za osnovno razvijanje aplikacije le računalnik s priljubljenim operacijskim sistemom (Windows, Linux, OSX), urejevalnik kode Eclipse (glej 2.1), Android SDK in Android Developer Tools (*ADT*) vtičnik za Eclipse. Razvijalec nujno ne potrebuje dejanske Android naprave, saj je v ADT vključen tudi simulator Android naprave. Aplikacijo je možno razvijati tudi brez ADT, a je v praksi to slaba izbira, saj ADT močno oljaša in pohitri razvoj.

Android Developer Tools (ADT)

ADT je vtičnik za Eclipse in zajema paleto orodij, ki so integrirana z Eclipse IDE. Omogoča dostop do mnogih funkcij, ki pripomorejo k hitremu razvijanju Android aplikacij. Del ADT-ja je tudi grafični vmesnik za mnoge ukaze SDK orodij in orodje za vizualno oblikovanje aplikacije, ki omogoča hitro oblikovanje in sestavljanje uporabniškega vmesnika aplikacije.

Nekatere glavne prednosti uporabe ADT:

- **Integrirano ustvarjanje projekta, razvijanje, pakiranje in razhroščevanje**
  ADT v Eclipse integrira veliko neobhodnih razvijalčevih nalog, kar olajša delo pri večini stopenj razvoja aplikacije.

- **Integracija SDK orodij**
  Veliko orodij, ki jih nudi Android SDK je integriranih v Eclipsove menije, poglede ali pa so del procesov v ozadju, ki jih vodi ADT.

- **Programski jezik Java in XML urejevalniki**
  Urejevalnik Java programske kode vključuje običajne lastnosti IDE-jev, kot so preverjanje sintakse ob prevajanju, samodejno zaključevanje (*auto-completion*) in integrirano dokumentacijo za Android API-je. Del ADT-ja je tudi urejevalnik XML dokumentov, ki omogoča urejanje aplikacijskih XML datotek v obliki obrazca (namesto ročnega pisanja XML

kode). Urejevalnih grafične podobe aplikacije omogoča oblikovanje uporabniškega vmesnika s tehniko *primi in spusti* (*drag and drop*).

- **Integrirana dokumentacija za Android API-je**
  Do dokumentacije lahko dostopamo s preletom miške čez razrede, metode in spremenljivke v kodi.

2.4 Luščenje spletnih podatkov in HTMLUnit

2.4.1 Luščenje spletnih podatkov

Luščenje spletnih podatkov (LSP, ang. *web scraping*) je programska tehnika samodejnega pridobivanja informacij s svetovnega spleta. LSP je del širšega področja, kjer potekajo aktivne raziskave s ciljem uresničitve vizije t.i. semantičnega spleta, kjer bi računalniki znali sami pametno iskati po mreži podatkov, ki ji danes pravimo svetovni splet. Na spletu so danes podatki večinoma nestrukturirani, namenjeni človeškemu načinu razmišljanja in branja. Do uresničitve semantičnega spleta manjkajo še znanstveni preboji na področjih obdelovanja tekstovnih podatkov, umetne inteligence in interakcije človek-računalnik.

LSP je aplikativno področje in strmi k avtomatičnemu pridobivanju podatkov na obstoječi obliki spleta in s trenutnimi tehnologijami. V večini primerov je LSP računalniška simulacija človeškega brskanja po svetovnem spletu. To je doseženo ali z implementacijo HTTP protokola ali pa z implementacijo celotnega spletnega brskalnika brez grafičnega vmesnika.

LSP je tesno povezan s spletnim indeksiranjem, kjer prav tako računalniški sistem (*bot, crawler*) pregleduje spletne strani in jih vpisuje v svoje indeksne datoteke (kazalo besed, ki se pojavljajo na spletni strani). Za razliko od spletnega indeksiranja je LSP osredotočen na preoblikovanje nestrukturiranih podatkov v strukture in ne le indeksiranje nestrukturiranih podatkov.

2.4.2 HTMLUnit

HTMLUnit [17] je brskalnik brez grafičnega vmesnika. Omogoča manipulacijo s spletnimi stranmi iz programov napisanih v Javi. S programsko kodo lahko po spletni strani klikamo, beremo vsebino in strukturo spletne strani, izpolnjujemo in pošiljamo obrazce. HTMLUnit omogoča tudi delo z JavaScriptom, Ajax zahtevami, piškotki, HTTPS varnostjo in osnovno HTTP preverjanje pristnosti (*authentication*). Skratka zajema večino sposobnosti modernih brskalnikov. Razvijalci so šli celo tako daleč, da se da delovanje brskalnika nastaviti na simulacijo Internet Explorerja ali Mozilla Firefoxa.

Najbolj pogosto se HTMLUnit uporablja za samodejno testiranje spletnih strani, uporablja pa se tudi za luščenje spletnih podatkov.

Za navigacijo po HTML dokumentu HTMLUnit uporablja XPath notacijo.

XPath

XPath oziroma *XML path language* je poizvedovalni jezik za izbiranje elementov v XML dokumentu. Definiral ga je World Wide Web Consortium (W3C) - največja svetovna organizacija za določanje standardov svetovnega spleta.

Primer 2.1 prikazuje XPath notacijo za iskanje elementa v XML dokumentu.

Primer 2.1: Iskanje elementa z XPathom

```
//*[@id='container']/div[2]/table/
```

Sintaksa XPath izraza 2.1

- // - označuje potomca, direktnega ali globljega. Ker pred tem ni nobenega izraza gre za kateregakoli potomca korenskega elementa, torej se element išče kjerkoli v dokumentu.

- **[@id='container']* - označuje vse najdene elemente (*) z atributom id enakemu 'container'.
- */* - označuje direktnega potomca
- *div[2]* - označuje drugega potomca tipa *div*.
- *table* - označuje vse potomce tipa *table*.

Izraz najprej poišče vse potomce korenskega elementa z atributom *id* enakim *'container'*. Nadalje pri vseh potomcih poišče drugi element tipa *div* (če obstaja) ter nato na teh elementih izbere vse direktne potomce tipa *table*. Poizvedba s tem izrazom bo vrnila seznam izbranih XML elementov tipa *table*, torej seznam HTML tabel.

2.5 Geokodiranje in Google Geocoding API

2.5.1 Geokodiranje

Geokodiranje je proces iskanja geografskih koordinat (geografska širina in dolžina) na podlagi drugih geografskih podatkov kot so naslov, ime mesta ali poštna številka. Praksa je pogosto uporabljena v informacijskih sistemih, kjer je naslov ponavadi znan, koordinate pa ne. Koordinatni zapis je računalniku bolj razumljiv kot naslov (verjetno tudi človeku, če ima pred seboj zemljevid). Na njihovi podlagi lahko računalnik določi razdalje med kraji, poišče interesantne lokacije v bližini, prikaže točko na zemljevidu...

2.5.2 Google Geocoding API

Google ponuja brezplačni spletni API za geokodiranje. Dostopen je preko spletnega naslova oblike

http://maps.googleapis.com/maps/api/geocode/[tip]?address=[naslov]

- *[tip]* - tip odgovora, ki je lahko XML ali JSON.

- *[naslov]* - dejanski naslov, ime kraja, ime države,... Google Geocoding podpira večino oblik zapisa naslova, ki bi jih naključni uporabnik uporabil.

Odgovor na tako zahtevo je torej lahko v obliki XML ali JSON. Primer 2.2 prikazuje odgovor tipa JSON na zahtevani naslov "Ljubljana, Slovenija". JSON objekt *lat* predstavlja geografsko širino, objekt *lng* pa geografsko dolžino.

Primer 2.2: Google Geocoding odgovor oblike JSON

```
{
   "results" : [
      {
         ...
         "geometry" : {
            ...
            "location" : {
               "lat" : 46.05645089999999,
               "lng" : 14.5080702
            }
         },
         ...
      }
   ]
}
```

2.6 Forecast.io API

Na spletu je ogromno ponudnikov vremenske napovedi. Večinoma so to spletne strani, kjer so podatki o vremenu k uporabniku poslani v obliki HTML strani. Če bi želeli te nestrukturirane podatke prevesti v strukture in jih nekam zapisati v poenoteni obliki bi se morali poslužiti tehnik luščenja spletnih podatkov (glej poglavje 2.4.1). Nekaj pa je tudi ponudnikov, ki nudijo spletne API-je za vremensko napoved. Za samodejno zbiranje podatkov o

vremenu je to najbolj primerna oblika, saj so podatki o vremenu že strukturirani, vseh skupaj pa je precej manj kot podatkov na tipični spletni strani, ki ima poleg podatkov o vremenu še meta podatke o prikazu teh podatkov (tabele, vremenska napoved v sliki, drugi HTML gradniki).

Forecast.io [22] je spletni ponudnik vremenske napovedi, ki uporabnikom ponuja tudi API vremenske napovedi. Brezplačno lahko pošljemo 1000 zahtev za vremensko napoved dnevno. Zahtevane parametre pošljemo v obliki URL naslova (primer 2.3), strežnik pa odgovori z JSON dokumentom, kjer je strnjena vremenska napoved za vsako uro v naslednjem dnevu in za vsak dan v naslednjem tednu.

Primer 2.3: URL vremenskega API-ja ponudnika forecast.io

```
https://api.forecast.io/forecast/[apikey]/[latitude],[longitude
    ],[time]
```

- *[apikey]* - Uporabnikov API ključ, ki je lahko uporabljen do tisoč krat na dan. Gre za niz znakov, ki ga uporabniku dodeli Forecast.io.
- *[latitude]* - Geografska širina v decimalni obliki s poljubno natančnostjo (npr. '43.0156').
- *[longitude]* - Geografska dolžina v decimalni obliki s poljubno natančnostjo (npr. '13.3372')
- *[time]* - Čas za katerega želimo izvedeti vremensko napoved. Za nekatere kraje je možna poizvedba za 60 let v preteklost. Čas je lahko UNIX oblike (število sekund od polnoči 1.1.1970 GMT) ali pa oblike [YYYY]-[MM]-[DD]T[HH]:[MM]:[SS] - oznake v oklepajih po vrsti: leto, mesec, dan, ure, minute, sekunde.

Poglavje 3

eRegatta - aplikacija za udeležence jadralnih regat

3.1 Problem

Kot vsi športniki tudi jadralci hodijo na tekmovanja. Perspektivni jadralec v tipični sezoni poleg treningov obišče od 10 do 25 regat doma in v tujini. Pri izbiri regat, ki jih bo obiskal ter kasneje pri udeležbi na regatah potrebuje določene informacije, ki so dostopne na svetovnem spletu:

- **Osnovni podatki o regati**
  Jadralci tekmujejo v različnih disciplinah na različnih tipih jadrnic na različnih nivojih tekmovanj. Vedeti morajo torej v kateri disciplini se bo jadralo, na kakšnih jadrnicah in kakšen je nivo (*grade*) ter s tem konkurenca na določeni regati. Vedeti morajo tudi kdaj in kje se bo regata odvijala, da lahko naredijo časovni in finančni plan.

- **Lokacija regate**
  Ker je potovanje drago, tako kot vsi individualni športniki tudi jadralci dajejo prednost regatam v bližini. Pred sezono torej izbirajo regate, ki so blizu, med obiskom regat pa je lokacija tako ali tako nujno potrebna.

- **Vremenska napoved**
  Za jadralce je vremenska napoved zelo pomembna. Če je vetra premalo ali preveč se regata prekini ali odpove in je ob slabi napovedi sploh nima smisla obiskati. Ko je napoved ugodna pa jadralcu vsaka informacija o spremembi smeri in moči vetra zlata vredna, saj lahko tako izdela boljšo taktiko. Tudi podatek o temperaturi in padavinah prav pride pri izbiri opreme, ki jo jadralec vzame na regato.

- **Prenočišče**
  Regate tipično trajajo od 2 do 5 dni, če se jadralec pripravlja na nastop v kraju regate pa bivanje tam traja še dlje. Kadar se regata ne odvija v domačem akvatoriju je na kraju regate treba prespati v lokalnem hotelu, apartmaju, hostlu,...

Problem, ki se tu odpira je kako jadralcu ob pravem času na pravem mestu zagotoviti te informacije. Jadralec jih mora imeti na voljo tako pred kot med regato.

3.2 Obstoječe rešitve

Za zdaj ne obstaja enotna rešitev, ki bi jadralcu zagotovila potrebne informacije. Edini način je iskanje vsake informacije posebej:

- **Osnovni podatki o regati**
  Podatke o regatah v tujini jadralci iščejo na spletni strani mednarodne jadralne zveze [23]. Mednarodna jadralna zveza ima v svoji bazi okoli 500 prihajajočih regat po katerih uporabnik poizveduje preko države, datuma, razreda, discipline, tipa, nivoja in imena regate - skratka preko vseh za jadralca pomembnih kategorij.

  Pri iskanju regat v domači državi jadralci uporabijo spisek regat, ki jih objavi nacionalna jadralna zveza na svoji strani. Ti podatki so ponavadi slabše strukturirani in redko katera zveza na svoji strani omogoča iskanje glede na vrsto regate.

- **Lokacija regate**
  Okvirno lokacijo regate (država in mesto) jadralec izve z ostalimi osnovnimi podatki iz prejšnje točke. Točna lokacija (naslov jadralnega kluba, ki je organizator) pa je napisana v t.i. razpisu. To je dokument, ki ga jadralni klub izda kot neke vrste vabilo z vsemi potrebnimi podatki. Ponavadi je objavljen na spletni strani kluba, ali pa ga moraš posebej zahtevati od kluba.

- **Vremenska napoved**
  Spletnih in mobilnih aplikacij, ki ponujajo vremensko napoved ne manjka. Nekatere znajo poiskati vremensko napoved za trenutni položaj, kar je uporabno, ko pride jadralec na regato. Pred regato pa je še vedno primoran poiskati lokacijo regate in jo vpisati ali najti v želeni aplikaciji.

- **Prenočišče**
  Tudi ponudnikov prenočišč na spletu ne manjka. Uporabnik lahko uporabi iskalnik kot je *Google* za direktno iskanje prenočišča ali pa obišče eno izmed spletnih strani, ki delujejo kot posredniki pri prenočiščih. Nekateri izmed bolj priljubljenih so *Booking.com* (*www.booking.com*), *TripAdvisor* (*www.tripadvisor.com*), *Expedia* (*www.expedia.com*) in *Hotels.com* (*www.hotels.com*). Vsi omogočajo tudi takojšnjo rezervacijo prenočišča.

3.3 Konceptualna ideja

Osnovna ideja diplomskega dela je združiti podatke, ki jih jadralec potrebuje pred in med udeležbo na regatah, v poenoten pogled, ki bo na voljo jadralcu takrat, ko bo podatke rabil. Ugotovili smo že, da podatki obstajajo in tudi dostop do njih je mogoč in brezplačen. Ostaneta dve težavi:

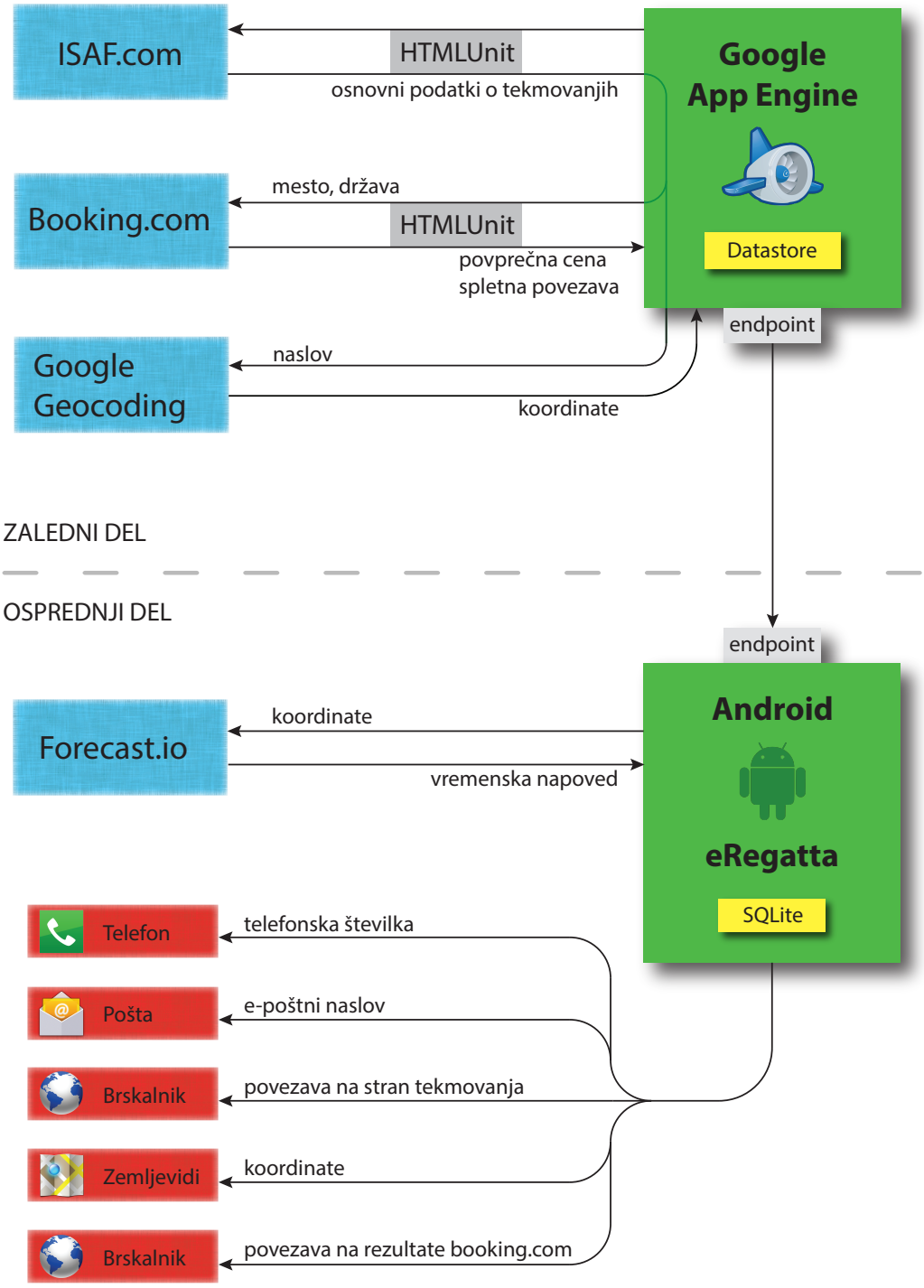
– Za iskanje podatkov jadralec porabi preveč časa, saj so ti podatki razdrobljeni po spletu

– Dostop do interneta je na regatah v tujini pogosto plačljiv ali pa ga ni, zato podatkov, ki se ne osvežujejo pogosto ni smiselno vsakič prejeti s spleta ampak jih shraniti.

Rešitev sem videl v izdelavi aplikacije na kateri so združeni in shranjeni vsi podatki. Podatke bi zbral s pomočjo tehnike samodejnega luščenja spletnih podatkov iz spletnih strani na katerih se nahajajo podatki. Te podatke bi nato uporabniku prikazal na enem mestu v poenotenem pogledu. To reši prvo težavo.

Najboljša rešitev za drugo težavo je mobilna aplikacija. Druga možnost je bila spletna stran prilagojena prikazu na mobilni napravi, a sem jo zavrgel, saj se bo aplikacija uporabljala tudi v tujini kjer je dostopnost do internetne povezave pogosto dražja ali pa je sploh ni. Podatki, ki so v strukturirani obliki shranjeni v mobilni aplikaciji so dostopni tudi brez spletne povezave. Ta je potrebna le za osveževanje podatkov. Ker se večina podatkov le malo spreminja, njihovo osveževanje ni potrebno vsak dan, sploh pa ne ob vsakem ogledu podatkov. Podatki na mobilni napravi so dostopni praktično kadarkoli in kjerkoli, saj ima lahko uporabnik mobilno napravo ves čas ob sebi. Odločil sem se za mobilni operacijski sistem Android, saj ima največji delež uporabnikov na trgu.

Arhitektura celotne rešitve bo razdeljena na dva dela. Zaledni del sistema eRegatta predstavlja aplikacija na podlagi Googlovega sistema AppEngine, ki bo delovala kot strežnik, ki na spletu periodično pridobiva in hrani podatke, za katere ni potrebno, da so popolnoma ažurni. Ti podatki se prenesejo v osprednji del sistema, ki ga predstavlja Android aplikacija. Ta podatke prikaže in jih sama obogati še s podatki, ki morajo biti ažurni. V mojem primeru so to podatki o vremenski napovedi, ki je za jadralce zelo pomembna.



Slika 3.1: Arhitektura aplikacije eRegatta.

3.4 Arhitektura sistema

Kot je prikazano na sliki 3.1 je aplikacija razdeljena na dva dela - zaledni in osprednji del.

- **Zaledni del** predstavlja spletna aplikacija na spletnem strežniku App Engine, ki dnevno zbira podatke. Najprej zbere osnovne podatke o regatah iz spletne strani mednarodne jadralne zveze, nato pa jih uporabi pri iskanju prenočišč in koordinat mesta, kjer se bo posamezni dogodek odvijal. Vse skupaj shrani v podatkovno bazo.

- **Osprednji del** predstavlja aplikacija eRegatta za operacijski sistem Android, ki na uporabnikovo zahtevo prenese podatke iz zalednega sistema in jih shrani v svojo SQLite podatkovno bazo. Sama aplikacija eRegatta na uporabnikovo zahtevo v podatkovni bazi osveži še vremensko napoved iz spletne strani *Forecast.io*. Vse skupaj prikaže v poenotenem pogledu in da uporabniku možnost, da uporabi podatke tudi v drugih funkcijah, ki jih ponuja mobilna naprava. V primeru, da so podatki znani lahko uporabnik:

  - Pokliče organizatorja regate
  - Pošlje e-pošto organizatorju
  - Odpre spletno stran regate
  - Prikaže lokacijo regate na zemljevidu
  - Odpre podstran Booking.com, kjer so izpisani najdeni rezultati za prenočišča v bližini

Sliki 3.2 in 3.3 prikazujeta shemi podatkovnih baz na strežniku in na mobilni aplikaciji. Kot vidimo je struktura podobna. Obe podatkovni bazi imata tabeli *regatta* in *classes*, kjer se nahajajo vsi podatki o regati. V NoSQL oblačni podatkovni bazi pravzaprav podatki niso shranjeni v tabelo ampak kot določen tip. Iskanje po tipu podatka je zelo učinkovito zato

Slika 3.2: Shema podatkovne baze na App Engine strežniku.

si lahko vse podatke istega tipa predstavljamo kot neko tabelo v relacijski podatkovni bazi.

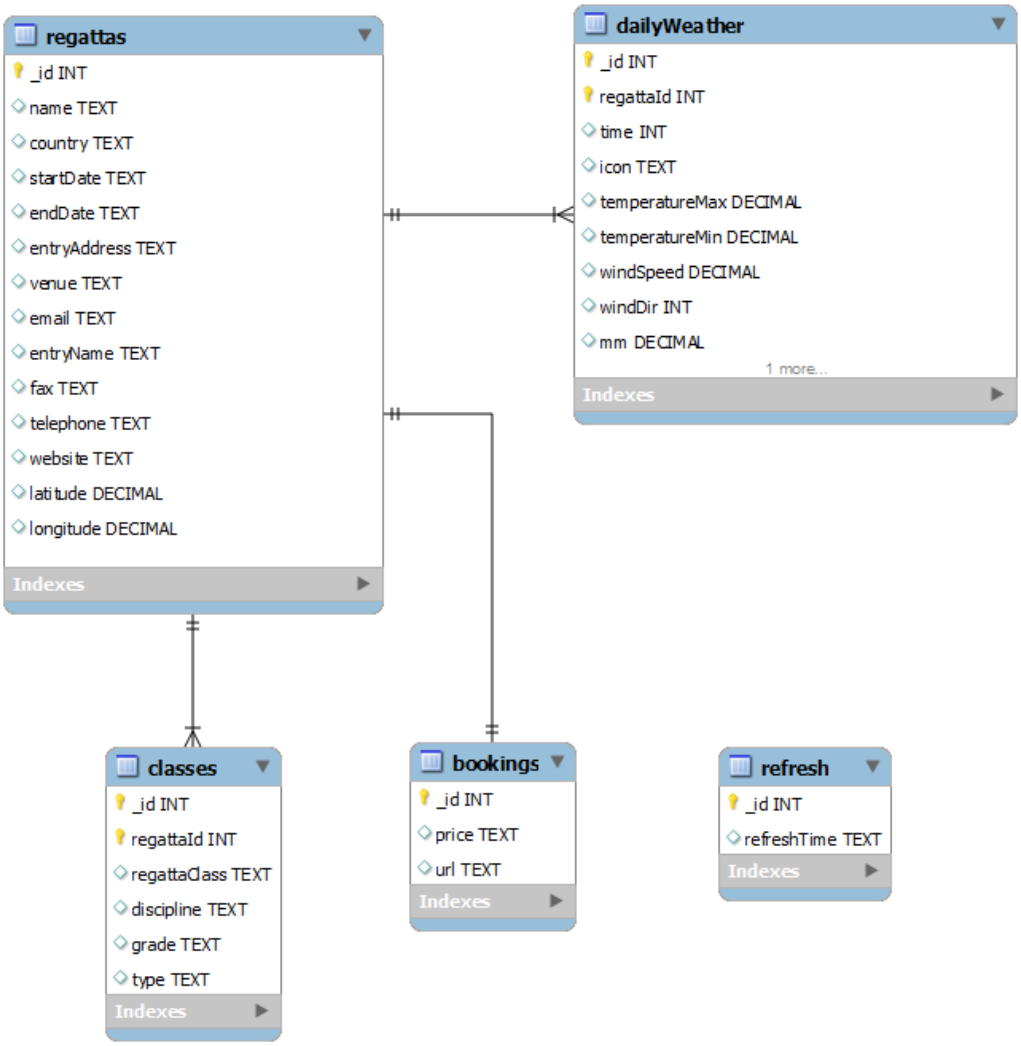
Vsaka regata lahko zajema več jadralnih disciplin, razredov, nivojev, zato sta ti dve tabeli v razmerju 1:n. Obe podatkovni bazi imata tudi tabelo *bookings*, kjer je za vsako regato največ en zapis o povprečni ceni prenočišča s tremi zvezdicami za dve osebi v času regate ter spletna povezava do najdenih rezultatov prenočišč v bližini regate.

Poleg tega je v podatkovni bazi na mobilni napravi še tabela *dailyWeather*, v kateri vsak zapis opisuje vremensko napoved za en dan določene regate. Vsaka regata ima največ pet zapisov vremenske napovedi, za naslednjih pet dni. Podatki o vremenski napovedi ne pridejo iz strežnika, temveč jih mobilna naprava sama poišče.

Tabela *refresh* na mobilni napravi ima zapis o tem kdaj so bili podatki nazadnje preneseni iz strežnika v mobilno napravo.

3.5 Implementacija

Celotna izvorna koda rešitve je dostopna na spletnem naslovu *https://github.com/eurekast/eRegatta*



Slika 3.3: Shema podatkovne baze v mobilni aplikaciji eRegatta.

3.5.1 Zaledni sistem - App Engine

Zaledni sistem je v celoti postavljen na Googlovo storitev App Engine (glej 2.2).

Cilj zalednega sistema je zbrati, shraniti in nuditi podatke, ki se s časom ne spreminjajo veliko. To so:

- osnovni podatki o regatah - dostopno na spletni strani *www.sailing.org*
- podatki o povprečni ceni prenočišča v bližini regate in spletna povezava do njih - dostopno na spletni strani *www.booking.com*
- koordinate kraja regate - pretvorba preko spletnega APIja dostopnega preko *maps.googleapis.com/maps/api/geocode/*

Na App Enginu uporabnik izvajanje kode sproži s HTTP zahtevo. Ob tem se zažene določen proces, ki lahko dalje proži zagon drugih procesov oziroma t.i. nalog (*tasks*). Naloge se izvajajo v ločenih nitih. Cikel podatkovnih prenosov od spletnih virov k končnemu uporabniku mobilne aplikacije eRegatta se začne s proženjem HTTP zahteve za naslov *eregatta.appengine.com/putregattas*, ki zažene kodo napisano v *putregattas.class* ta pa poskrbi za prejem podatkov o ovanjih in o koordinatah.

Podatki o regatah

Mednarodna jadralna zveza nudi seznam vseh jadralnih regat, ki se bodo odvijala v (bližnji) prihodnosti. V bazi imajo ves čas okoli 500 regat. Vsaka od regat ima svojo podstran na kateri so napisani vsi podatki, ki so jih organizatorji regate posredovali Mednarodni jadralni zvezi. Seznam regat je dostopen na spletnem naslovu http://www.sailing.org/regattas.php.

Poslužil sem se tehnik luščenja spletnih podatkov in uporabil spletni brskalnik za Java programe HTMLUnit (glej 2.4.2). S pomočjo HTMLUnita

spletna aplikacija prebrska vse podstrani regat in izlušči podatke o regatah. Primer 3.1 prikazuje odpiranje strani s prvimi 125 regatami.

Primer 3.1: Odpiranje strani s pomočjo HTMLUnita.

```
HtmlPage page = webClient.getPage("http://members.sailing.org/
   regattas.php?max=125&begin=1");
```

Na tem mestu se podatki o regatah še ne preberejo, prebere se le identifikacijska številka regate (ID). Za vsako regato se nato ustvarijo naloge (*tasks*), ki tečejo vzporedno. Naloge na podlagi ID-ja določijo spletni naslov (URL) podstrani regate in jo odprejo na tak način kot v primeru 3.1. Tam so podatki o regati vedno izpisani v enaki HTML obliki, zato jih aplikacija lahko poišče z vnaprej izdelanim XPath izrazom. Ti podatki se nato shranijo v Datastore kot je prikazano v primeru 3.2. Objekt *tdClassList* v primeru 3.2 predstavlja seznam vseh HTML celic s podatki o regati. Iz njih izluščimo podatke o regati, jih shranimo v objekt tipa *Entity*, ki ga nato z ukazom *put* shranimo v podatkovno bazo Datastore.

Primer 3.2: Branje in shranjevanje podatkov o regatah

```
String discipline = tdClassList.get(0).getTextContent();
String type = tdClassList.get(1).getTextContent();
...

Entity regattaClass = new Entity ("class", regatta.getKey());
regattaClass.setProperty("Discipline", discipline);
regattaClass.setProperty("Type", type));
...

datastore.put(regattaClass);
```

Na tej točki smo Datastore napolnili s podatki tipa *class* in *regattas*.

Koordinate

Vsaki regati je treba dodati še koordinate, ki se uporabijo pri prikazu lokacije na zemljevidu in pri zahtevi za vremensko napoved. Za pretvorbo naslova v

koordinate aplikacija uporablja Google Geocoding API tako kot je prikazano v primeru 2.2. Iz prejetega JSON dokumenta se preberejo koordinate in se shranijo poleg ostalih osnovnih podatkov o regatah kot tip *regatta*.

Prenočišče

Za podatke o prenočišču aplikacija uporablja podatke s spletne strani *www.booking.com*. Koda za pridobivanje podatkov o prenočišču se sproži s HTTP zahtevo /putextras, ki zažene *putExtras.class*. Za vsako regato v podatkovni bazi ta razred ustvari nalogo in ji poda infomacije, ki so potrebne za iskanje prenočišč (primer 3.3).

Primer 3.3: Zahteva za novo nalogo na Google App Engine

```
Queue queue = QueueFactory.getDefaultQueue();
queue.add(withUrl("/updatebooking")
    .param("venue", venue)
    .param("country", country)
    .param("regattaKey", regattaKey)
    .param("startDay", startDateForInput[0])
    .param("startMonthYear", startDateForInput[1])
    .param("endDay", endDateForInput[0])
    .param("endMonthYear", endDateForInput[1]));
```

Metode HTMLUnita v razredu *updateBooking.class* odprejo spletno stran, izpolnijo obrazce v katere vpišejo mesto, državo, datum začetka in konca regate ter izberejo ponudnike prenočišč s tremi zvezdicami. Stran s končnimi rezultati iskanja se odpre z navideznim klikom na povezavo, kot je prikazano v primeru 3.4.

Primer 3.4: Iskanje elementa na spletni strani in klik nanj

```
HtmlAnchor link = (HtmlAnchor) page3.getByXPath("//*[@id='
   filter_class']/div[2]/a[3]").get(0);
HtmlPage page = link.click();
```

Aplikacija na tej strani poišče prvih 20 rezultatov, izračuna povprečno ceno ter jo poleg spletnega naslova shrani v podatkovno bazo kot tip *booking*.

Periodično osveževanje podatkov

Osveževanje podatkov na spletnem strežniku se zgodi enkrat na dan. Aplikacija uporablja funkcijo App Engina imenovano *cron* za periodično klicanje funkcij zalednega sistema. *Cron* omogoča razvijalcu, da nastavi urnik za izvajanje kode. Urnik se nastavi v XML dokumentu *cron.xml*. Primer 3.5 predstavlja cron dokument v naši aplikaciji.

Primer 3.5: Dokument cron.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/putregattas1</url>
    <description>Update regattas 1/4</description>
    <schedule>every day 19:30</schedule>
    <timezone>Europe/Ljubljana</timezone>
  </cron>

  <cron>
      <url>/putregattas2</url>
      ...
  </cron>
  ...

  <cron>
    <url>/putextras</url>
    <description>Update bookings</description>
    <schedule>every day 20:00</schedule>
```

```
    <timezone>Europe/Ljubljana</timezone>
  </cron>
</cronentries>
```

V primeru 3.5 je prikazan nastavljen urnik za enga od štirih delov kode, ki osvežujejo podatke o regatah iz strani mednarodne jadralne zveze in koordinate regate - to so zahteve za URL */putregattas#*. Ti štirje deli se razlikujejo le po tem katere regate osvežujejo. Vsak od njih osveži 125 regat. Razlog za delitev na štiri dele je, da App Engine brezplačno dovoljuje le 30 sekund za obdelavo HTTP zahteve, kar ni dovolj za osvežitev celotne podatkovne baze. Poleg periodičnega osveževanja podatkov o regatah se periodično osvežujejo tudi podatki o prenočiščih - HTTP zahteva z URL-jem */putextras* (zadnji del XML dokumenta v primeru 3.5).

Prenos podatkov na mobilno napravo

Podatki so na tej točki zbrani in shranjeni v podatkovni bazi v oblaku. Sedaj jih je treba prenesti na mobilno napravo, kjer bodo prikazani uporabniku. Aplikacija za prenos uporablja tehnologijo Google Cloud Endpoints. Cloud Endpoints razvijalcu močno olajšajo delo pri prenosu podatkov iz Google App Enginea na odjemalce, v primeru moje aplikacije na Android napravo.

Razvijalcu se ob uporabi Cloud Endpoints ni treba ukvarjati s prenosom po omrežju. ADT vtičnik za Eclipse sam ustvari razred, ki služi kot končna točka povezave (*endpoint*) na strani strežnika. Razvijalec v ta razred napiše metode v katerih se želeni podatki za prenos zapakirajo in vrnejo kot objekt. Metode je treba označiti (*annotate*) po predpisanih pravilih. Samodejno ustvarjen razred *RegattaEndpoint* in metoda, ki pošlje vse regate iz podatkovne baze je prikazan v primeru 3.6.

Primer 3.6: Razred RegattaEndpoint

```
@Api(name = "regattaendpoint")
public class RegattaEndpoint {
    @ApiMethod (httpMethod="GET")
    public List<Regatta> downloadRegattas() {
        List<Regatta> results = new ArrayList<Regatta>();

        // polnjenje seznama z objekti tipa Regatta

        return results;
    }
}
```

ADT nato na podlagi oznak (*@Api, @ApiMethod*) sam zgenerira knjižnice, ki bodo služile kot končna točka na strani odjemalca. ADT samodejno vključi te knjižnice v Eclipsov projekt, kjer razvijamo odjemalce oziroma osprednji del aplikacije. S klicem ene od metod v teh knjižnicah se podatki prenesejo na napravo in preoblikujejo nazaj v objekt s katerim manipuliramo naprej (primer 3.7).

Primer 3.7: Klic za prejem podatkov na napravo

```
ArrayList<Regatta> regattas = (ArrayList<Regatta>)
   regattaEndpoint.regattaEndpoint().downloadRegattas().execute
   ().getItems();
```

3.5.2 Osprednji sistem - Android

Osprednji sistem oziroma sistem, ki prikazuje podatke uporabniku, predstavlja aplikacija *eRegatta* za operacijski sistem Android. Sestavljena je iz treh t.i. fragmentov (*Fragments*), vsak predstavlja en pogled:

- *SearchFragment* - vsebuje obrazec za iskanje regat.
- *ListRegattasFragment* - prikaže seznam regat, ki so bile najdene.

- *RegattaViewFragment* - prikaže podatke, ki se nanašajo na izbrano regato.

Fragmenti so deli logike ali uporabniškega vmesnika aplikacije, ki tečejo znotraj aktivnosti (*activity*). Prvič so bili predstavljeni v Androidu verzije 3.0. Pred uvedbo fragmentov je praviloma vsak pogled predstavljala ena aktivnost, te pa ne morejo teči vzporedno. Sedaj lahko delce aplikacije (fragmente) poljubno prikazujemo enega poleg drugega. To je predvsem uporabno, ko želimo izkoristiti večje ekrane prenosnih naprav, ki lahko zato prikažejo večji uporabniški vmesnik. Razvijalec lahko določi, da se na večjem ekranu prikaže več fragmentov kot na manjšem, kar aplikacijo naredi bolj priročno.

SearchFragment - Iskanje regat

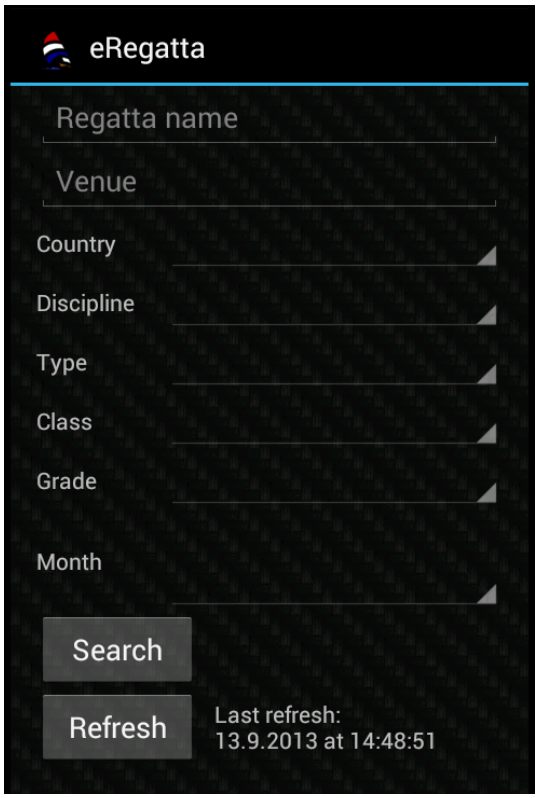
SearchFragment je prvi del aplikacije in se odpre, ko uporabnik zažene aplikacijo (slika 3.4). Vsebuje obrazec za iskanje regat in gumb *Refresh*.

Refresh gumb sproži osveževanje podatkovne baze s podatki iz zalednega sistema. Iz zalednega sistema prejmemo objekte treh vrst:

- *Regatta* - podatki tipa *regatta* v podatkovni bazi zalednega sistema.
- *RegattaClass* - podatki tipa *class* v podatkovni bazi zalednega sistema.
- *Booking* - podatki tipa *booking* v podatkovni bazi zalednega sistema.

V ločeni niti se najprej za vsako vrsto objektov zaženejo metode, ki s tehnologijo Google Cloud Endpoints pridobijo podatke iz zalednega sistema (primer 3.7). Ti se nato zapišejo v podatkovno bazo (slika 3.3 - tabele *regattas*, *classes* in *bookings*). Poleg njih se v podatkovno bazo zapiše tudi trenutni čas, da uporabnik ve kdaj je nazadnje osvežil podatke.

Za vsa branja in pisanja v podatkovno bazo je zadolžen razred, ki razširja razred *SQLiteOpenHelper*. V ta razred razvijalec napiše metode, ki predstavljajo en opravek s podatkovno bazo. Primer 3.8 prikazuje metodo, ki v



Slika 3.4: Prvi zaslon aplikacije eRegatta.

podatkovni bazi osveži podatke o regatah. Metoda kot vhod dobi seznam objektov vrste *Regatta* (prejeti v primeru 3.7) in za vsakega v podatkovno bazo zapiše vsebino vseh lastnosti.

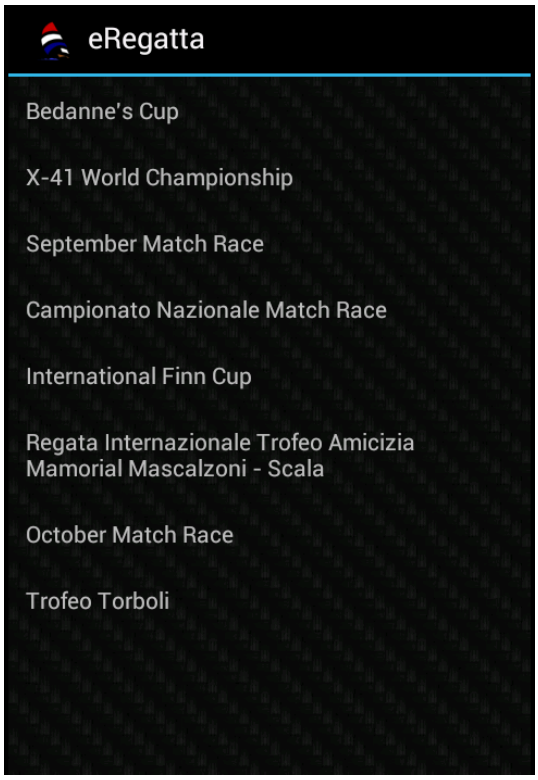
Primer 3.8: Metoda *refreshRegattas*

```
public void refreshRegattas(ArrayList<Regatta> regattas)
{
    SQLiteDatabase db = this.getWritableDatabase();
    Iterator<Regatta> regattaIterator = regattas.iterator();
    while (regattaIterator.hasNext()
    {
        Regatta regatta = (Regatta) regattaIterator.next();
        ContentValues values = new ContentValues();
        values.put(COLUMN_ID, regatta.getId());
        values.put(COLUMN_NAME, regatta.getName());
        ...
        db.insert(TABLE_REGATTAS, null, values);
    }
    db.close();
}
```

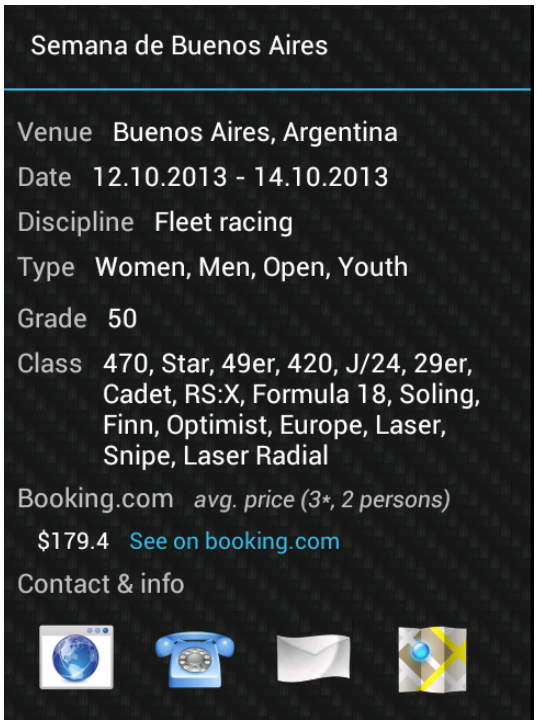
Obrazec je enak tistemu na ISAF-ovi spletni strani. Uporabnik lahko išče po imenu, kraju, državi, jadralni disciplini, tipu, jadralnem razredu, nivoju in mesecu regate. Ob pritisku na gumb *Search* se izvede metoda, ki sproži poizvedbo po SQLite podatkovni bazi. Seznam regat, ki ustrezajo poizvedbi se prikaže v fragmentu *ListFragment*.

ListRegattasFragment - prikaz rezultatov iskanja

Ta fragment je zadolžen za prikaz seznama regat, ki ga vrne poizvedba v *SearchFragment*. Slika 3.5 prikazuje uporabniški vmesnik tega fragmenta. Gre za preprost del aplikacije, ki v zanki na zaslon izriše seznam imen regat in vsakemu doda poslušalca za klik (*OnClickListener*), ki ob kliku na regato sproži prikaz podatkov o regati v naslednjem fragmentu - *RegattaViewFragment*.



Slika 3.5: Fragment *ListRegattaFragment* prikazuje rezultate iskanja



Slika 3.6: Grafični vmesnik fragmenta *RegattaViewFragment*, ki prikazuje podatke o regati

RegattaViewFragment - prikaz podatkov o regati

Ta fragment je za uporabnika aplikacije najbolj pomemben, saj prikazuje vse informacije o regatah, ki se zbirajo in predelujejo v zalednem in tudi ospredjem sistemu. Sliki 3.6 in 3.7 prikazujeta uporabniški vmesnik fragmenta, ki vsebuje:

- podatke o regati
- povprečno ceno prenočišča in spletno povezavo do relevantnih prenočišč
- gumbe za uporabo podatkov v drugih aplikacijah
- vremensko napoved

RegattaViewFragment iz prejšnjega fragmenta (*RegattaListFragment*) dobi identifikacijsko številko regate. Z njo poišče vse relevantne podatke iz tabel

regattas, *classes* in *booking*.

Podatki o regati so prikazani najprej. To so podatki iz tabel *regattas* in *classes*, ki so bili pridobljeni iz strani mednarodne jadralne zveze.

Povprečna cena prenočišča je prikazana za tem. Prikazuje koliko stane povprečna soba za dve osebi v hotelu s tremi zvezdicami v času regate. Za več informacij uporabnik pritisne na tekst *See on booking.com* in v brskalniku se odpre spletna stran booking.com s seznamom prostih prenočišč v bližini regate.

Gumbi za aplikacije zaženejo druge aplikacije in jim v zahtevi za zagon (*Intent*) podajo nekatere podatke:

- naslov spletne strani regate - Zažene se brskalnik in odpre spletno stran.
- telefonska številka organizatorja - Odpre se aplikacija za klicanje, kjer je že vtipkana telefonska številka, uporabnik le še pokliče.
- e-poštni naslov organizatorja - Odpre se novo e-poštno sporočilo v privzeti aplikaciji za e-pošto. Samodejno se izpolni prejemnik.
- koordinate regate - Odpre se aplikacija Zemljevidi (*Google Maps*) in na zemljevidu se označi mesto kjer se odvija regata.

Slika 3.7: Drugi del grafičnega vmesnika fragmenta *RegattaViewFragment*, ki prikazuje vremensko napoved

Vremenska napoved je zadnji del fragmenta. Če se v podatkovni bazi že nahajajo podatki o vremenski napovedi se prikažejo na zaslon. Vremenska napoved (slika 3.7) je sestavljena iz sledečih elementov, ki so relevantni za jadralce:

- ikona, ki prikazuje splošno vremensko stanje
- najnižja temperatura (°C)
- najvišja temperatura (°C)
- hitrost vetra v vozlih (kt)
- grafični prikaz smeri vetra

Poleg podatkov je tu še gumb *Refresh weather*, ki osveži podatke o vremenu. Podatki o vremenu na napravo pridejo preko spletnega API-ja Forecast.io (glej 2.6). Ta uporabniku vrne JSON dokument z vremensko napovedjo za kraj s koordinatami, ki jih aplikacija poda v URL naslov. Pošiljanje

zahteve za vremensko napoved, prejem vremenskih podatkov in njihovo pisanje v podatkovno bazo se zgodijo v ločeni niti. Primer 3.9 prikazuje sestavo URL naslova za zahtevo, njeno izvrševanje in vračanje JSON dokumenta v katerem so zapisani vsi podatki o vremenski napovedi.

Primer 3.9: Prejemanje vremenske napovedi

```
String url = FORECAST_URL+API_KEY+"/"+regatta.getLatitude().
   toString()+","+regatta.getLongitude().toString()+"?units=si";

HttpGet httpget = new HttpGet(url);
HttpResponse response = httpclient.execute(httpget);
String responseBody = EntityUtils.toString(response.getEntity())
```

JSON dokument aplikacija s pomočjo knjižnjice *Gson* [25] (Googlov JSON razčlenjevalnik) razčleni in sestavi objekte vrste *DailyWeather*. Ti se nato pošljejo metodi za shranjevanje v podatkovno bazo (tabela *dailyWeather*) in se uporabijo pri izrisovanju vremenske napovedi na zaslon. En primerek objekta *DailyWeather* predstavlja vremensko napoved za določen dan - en stolpec na sliki 3.7. Vremenske ikone v vsakem stolpcu se določijo glede na oznako ikone, ki jo prejmemo od Forecast.io, same ikone pa mora uporabnik spletnega API-ja sam zagotoviti. Mogočih je 10 ikon. Grafični prikaz smeri vetra je implementiran kot vrteča se bitna slika. Zavrti se za toliko stopinj kot je podatek o smeri vetra.

Poglavje 4

Zaključek

4.1 Sklepne ugotovitve

Cilj tega dela je bil jadralcem zagotoviti podatke, ki jih potrebujejo pri obisku regat in sicer na tak način, da bodo dostopni takrat, ko jih bo jadralec rabil.

Ugotovili smo, da podatki, ki jih jadralec potrebuje že obstajajo in so v digitalni obliki dostopni na svetovnem spletu. S pomočjo tehnik luščenja spletnih podatkov je mogoče te podatke samodejno pridobivati. To smo implementirali z uporabo spletnih API-jev in HTMLUnit knjižnice, ki teče na Google App Enginu in periodično pregleduje spletne vire ter shranjuje podatke v NoSQL podatkovno bazo v oblaku.

Ugotovili smo tudi, da je mobilna aplikacija najbolj primerna oblika prikaza teh podatkov, saj so podatki na jadralčevi mobilni napravi na voljo tudi med potovanjem na regate v tujini, ko je pogosto otežen dostop do internetne povezave. Odločili smo se za mobilni operacijski sistem Android, saj ima ta trenutek kar 79% tržni delež na trgu pametnih telefonov.

Podatke iz Google App Engina na mobilno napravo smo prenesli s tehnologijo Google Cloud Endpoints, ki je še ena od novih, perspektivnih tehnologij, ki smo jih spoznali. Na mobilno napravo smo nato preko spletnega API-ja Forecast.io prenesli še vremensko napoved in vse skupaj prikazali v poenotenem in poenostavljenem pogledu.

Cilj smo torej izpolnili. Možnosti za nadaljnji razvoj aplikacije pa je še kar nekaj.

4.2 Možnosti za nadaljnje delo

Aplikacija je prototip in možnosti za izboljšave je še veliko. Med izdelavo sem dobil še nekatere ideje:

- Prikaz vremenske napovedi za vsakih nekaj ur, ne le za cel dan. Za jadralce je pomembna podrobna vremenska napoved.
- Možnost dodajanja regat med priljubljene regate, za kasnejšo ponovno rabo.
- Možnost, da uporabnik doda svoje regate. Ta funkcija je nujno potrebna za resnično uporabnost aplikacije, saj regat, ki niso objavljene na strani ISAF, sedaj v aplikiciji ni.
- Prikaz podatkov še v spletni aplikaciji in aplikaciji za iOS. S tem bi zagotovili dostop do aplikacije za veliko večino potencialnih uporabnikov.
- Razviti algoritem za določanje naslova, kjer se regata odvija. Podatki o točnem naslovu, ki ga aplikacija prejme iz spletne strani ISAF, so namreč nekonsistentni, pogosto sploh ne gre za naslov. Veliko primerov bi bilo treba preoblikvati, da bi imel Google Geocoding API večji odstotek najdenih koordinat.
- Prikazovanje rezultatov regat, ko se regata konča. Za nekatere regate so rezultati dostopni na spletni strani ISAF-a.
- Lepši uporabniški vmesnik mobilne aplikacije

Komercialno trženje aplikacije ta trenutek ni mogoče. Če bi hoteli aplikacijo objaviti na Google App Storu bi morali:

- skleniti dogovor z mednarodno jadralno zvezo o uporabi podatkov o regatah, ki jih pridobivamo na njihovi strani. Če bi bil ta dogovor uspešen bi bilo bolj smiselno za mednarodno jadralno zvezo razviti API, ki bi ga aplikacija direktno uporabljala, kot da se podatki luščijo iz spletne strani.

- skleniti dogovor z Booking.com o uporabi podatkov na njihovi spletni strani. Predvidevam, da bi ta dogovor dosegli, saj aplikacija nudi direktno povezavo do hotelov na njihovi strani. Tudi tu bi bilo bolj smiselno razviti spletni API na strani vira.

- zakupiti več virov na Google App Enginu kot jih je na voljo brezplačno.

- zakupiti več zahtev za vremensko napoved.

Ciljna publika, ki bi aplikacijo uporabljala je relativno majhna, saj se z jadranjem ukvarja malo ljudi. Predvidevam da ima aplikacija nekaj deset tisoč potencialnih uporabnikov, če bi izpolnili nujne zahteve, ki so naštete zgoraj in aplikacijo izdali na App Storu. Potrebna bi bila tudi podrobna analiza trga, stroškov, trženja preden bi začeli razmišljati, da bi aplikacijo dali na trg. Ima pa aplikacija potencial za razširitev na druge športe, kar bi sigurno dalo dosti večjo vrednost aplikaciji, a bi bil tudi razvoj zaradi specifik vsakega športa dolgotrajen oziroma drag.

Literatura

[1] Maurizio Lenzerini: *Data integration: a theoretical perspective*, ACM, 2002.

[2] (2013) Wikipedia: Data integration. [Online].
http://en.wikipedia.org/wiki/Data_integration

[3] J. Gosling, B. Joy, G. Steele, G Bracha: Java(TM) Language Specification, The (3rd Edition), Addison-Wesley, 2005.

[4] (2013) Eclipse. [Online].
http://www.eclipse.org/

[5] (2013) Wikipedia: Platform as a Service. [Online].
http://en.wikipedia.org/wiki/Platform_as_a_service

[6] (2013) ComputeNext: Global IaaS/Paas Pie at $2B. [Online].
https://www.computenext.com/blog/tag/paas-market-share/

[7] Google Developers: App Engine. [Online].
https://developers.google.com/appengine/

[8] (2013) Wikipedia: NoSQL. [Online].
http://en.wikipedia.org/wiki/NoSQL

[9] (2013) Google Developers: Overview of Google Cloud Endpoints. [Online].
https://developers.google.com/appengine/docs/java/endpoints/

[10] (2013) MobiThinking: Global mobile statistics 2013. [Online]. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/b#mobilepageviews

[11] (2013) Wall Street Journal: Google's Android Seizes Smartphone Market. [Online]. http://online.wsj.com/article/SB10001424127887323838204578654520703852466.html

[12] (2013) App Brain: Free vs. paid Android apps. [Online]. http://www.appbrain.com/stats/free-and-paid-android-applications

[13] (2013) Flurry: The History of App Pricing, And Why Most Apps Are Free. [Online]. http://blog.flurry.com/bid/99013/The-History-of-App-Pricing-And-Why-Most-Apps-Are-Free

[14] (2013) R. Penman, T. Baldwin, D. Martinez: *Web Scraping Made Simple with SiteScraper*, 2009.

[15] (2013) Google Developers: Android Developer Tools. [Online]. http://developer.android.com/tools/help/adt.html

[16] (2013) Wikipedia: Web scraping. [Online]. http://en.wikipedia.org/wiki/Web_scraping

[17] (2013) Uradna HTMLUnit dokumentacija. [Online]. http://htmlunit.sourceforge.net/

[18] (2013) Wikipedia: HTMLUnit. [Online]. http://en.wikipedia.org/wiki/HtmlUnit

[19] (2013) W3C: XPath. [Online]. http://www.w3schools.com/xpath/

[20] (2013) Wikipedia: Geocoding. [Online]. http://en.wikipedia.org/wiki/Geocoding

[21] (2013) Google Developers: The Google Geocoding API. [Online].
https://developers.google.com/maps/documentation/geocoding/

[22] (2013) Uradna stran Forecast.io. [Online].
http://forecast.io

[23] (2013) Uradna spletna stran mednarodne jadralne zveze. [Online].
http://www.sailing.org

[24] (2013) Uradna spletna stran Booking.com. [Online].
http://www.booking.com

[25] (2013) Knjižnjica Gson. [Online].
https://code.google.com/p/google-gson/