UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nik Adžič

RAZVOJ MOBILNE APLIKACIJE CARBONTRACKER

DIPLOMSKA NALOGA

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE

RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Mojca Ciglarič

Ljubljana, 2012

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: 00048/2012
Datum: 13.04.2012

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

Kandidat: **NIK ADŽIČ**

Naslov: **RAZVOJ MOBILNE APLIKACIJE CARBONTRACKER**
MOBILE APPLICATION CARBONTRACKER DEVELOPMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Danes se vedno bolj zavedamo onesnaževanja ozračja z ogljikovim dioksidom. Zasnujte in izvedite mobilno aplikacijo, ki bo pomagala dvigati ozaveščenost uporabnikov o različnih načinih prevoza in o njihovih različnih ogljičnih odtisih. Preučite in izberite primerne tehnologije, opišite arhitekturo aplikacije in pri njeni zasnovi upoštevajte zmogljivosti današnjih povprečnih mobilnih naprav. Aplikacijo izdelajte in preizkusite, opišite njeno delovanje in način uporabe. Kritično ovrednotite svoje delo in predlagajte morebitne možnosti za nadaljnje izboljšave.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani

Nik Adžič, z vpisno številko 63090036,

sem avtor diplomskega dela z naslovom:

RAZVOJ MOBILNE APLIKACIJE CARBONTRACKER

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne _____ 2012 Podpis avtorja:

Zahvala

Iskreno se zahvaljujem mentorici doc. dr. Mojci Ciglarič za spodbude, strokovno pomoč in usmerjanje pri izdelavi diplomske naloge.

Zahvaljujem se Saši, Anžetu, Marku in Andreju za izjemen trud in sodelovanje pri projektu CarbonTracker.

Zahvaljujem se tudi družini za spodbude med samim študijem in med časom pisanja diplome.

Zahvala gre tudi prijateljici Andreji, ki je poskrbela za lektoriranje mojega diplomskega dela.

Kazalo vsebine

Seznam kratic

Kratica	Angleški izraz	Slovenski izraz
SQL	Structured Query Language	Strukturirani poizvedovalni jezik
IDE	Integrated development environment	Integrirano razvojno okolje
XML	Extensible Markup Language	Razširjeni označevalni jezik
SDK	Software Development Kit	Programski razvojni paket
ADT	Android Development Tools	Androidova razvojna orodja
CSS	Cascading Style Sheet	Kaskadne slogovne predloge
HTML	HyperText Markup Language	Hipertekstni označevalni jezik
PHP	Hypertext Preprocessor	Hipertekstni predprocesor
JDT	Java Development Tools	Javanska razvojna orodja
MVC	Model-View-Controller	Model-pogled-nadzor
Java VM	Java Virtual Machine	virtualni stroj Java
GPS	Global Positioning System	Sistem globalnega določanja položaja
TLS	Transport Layer Security	Transportni nivo varnosti
SSL	Secure Sockets Layer	Nivo varnih vtičnic
GPRS	General Packet Radio Service	Splošna paketna radijska storitev
API	Application Programming Interface	Vmesnik za programiranje aplikacij
SGML	Standard Generalized Markup Language	Standardni posplošeni označevalni jezik
XSLT	EXtensible Stylesheet Language Transformations	Transformacije razširljivega slogovnega jezika

Povzetek

Diplomsko delo je razvoj mobilne aplikacije CarbonTracker in strežniškega dela, ki lahko teče ločeno od mobilne aplikacije in ga lahko uporabljamo neodvisno od operacijskega sistema mobilne aplikacije.

Mobilna aplikacija je bila razvita v programskem jeziku Java za operacijski sistem Android, ki je trenutno vodilni med operacijskimi sistemi za mobilne naprave. Razvita je bila v integriranem razvojnem okolju (kratica IDE) Eclipse z Androidovim programskim razvojnim paketom (kratica SDK) in Androidovimi razvojnimi orodji (kratica ADT). Opisali smo gradnike in pakete mobilne aplikacije, pri tem pa uporabili različne tehnologije, kot so: Google Maps, Google Transit, GPS in XPath. Predstavili in opisali smo tudi našo podatkovno bazo XML na mobilni napravi aplikacije. Pri strežniškem delu smo na nazoren način prikazali podatkovni model podatkovne baze MySQL in spletni del, kjer smo uporabili spletne tehnologije PHP, HTML in CSS.

V naslednjem delu pa smo opisali delovanje in razvoj obeh načinov aplikacije, TRACK in DIRECTIONS, pomen datoteke AndroidManifest.xml ter postavitev, ki je arhitektura za uporabniški vmesnik znotraj aktivnosti.

Aplikacijo smo razvili na našo željo, saj smo želeli pridobiti nekaj novih izkušenj s področja razvoja mobilnih aplikacij na platformi Android ter podatkovnih in spletnih tehnologij. Razvoj aplikacije je potekal nekaj mesecev, razvijali pa smo jo dobro načrtovano in sistematično. Z njo smo se udeležili dvotedenske intenzivne izmenjave EPSIA v Amsterdamu ter jo predstavili konzorciju Green IT, ki se zavzema za zmanjšanje onesnaževanja okolja in povezuje to z informacijskimi tehnologijami.

Ključne besede: mobilna aplikacija, CarbonTracker, strežnik, podatkovna baza XML, Android, Xpath, MySQL.

Abstract

The following Bachelor's thesis discusses the CarbonTracker mobile application development as well as the part of its server which may run without connection to the mobile application and may be used independently of it.

The mobile application was developed in the Java programming language for the currently leading operating system for mobile applications-Android. It was designed in the integrated development environment (IDE) Eclipse with the Android Software Development Kit (SDK) and the Android Development Tools (ADT). With the help of several different technologies (Google Maps, Google Transit, GPS and XPath), both the application elements and the mobile application packages are described as well as our own XML database on the application's mobile device.

The part with the server provides an illustrative example of both the data model of the MySQL database and of the network part, the latter being constructed with the use of PHP, HTML and CSS web technologies.

The next part includes a description of the operation and the development of both ways of the application, TRACK and DIRECTIONS, the meaning of the AndroidManifest.xml data and the layout serving as the design of the active user interface.

The application was developed as a result of our need to gain experience both in the field of mobile device development on the Android platform as well as in the field of database and web technologies. The disciplined and systematic approach of the application development was applied and the realisation of the project under this particular approach lasted a few months. The application enabled the two-week student exchange with the intensive training called EPSIA in Amsterdam and was presented to the committee of the Green IT, which fights environmental pollution and connects it with information technologies.

Keywords: mobile application, CarbonTracker, server, XML database, Android, XPath, MySQL.

1 Uvod

V današnjem času so pametni telefoni z nameščenim operacijskim sistemom Android v veliko primerih že vsakdanjik. S tem je povezana uporaba mobilnih aplikacij, ki jih uporabljajo uporabniki pametnih telefonov. Skoraj vsako podjetje ima svojo mobilno aplikacijo, ki mu omogoča bodisi predstavitev podjetja bodisi druge storitve. To omogoča podjetju večji zaslužek, širjenje, opravljanje določenih storitev v podjetju kar prek pametnih telefonov ipd.

Težavo pa predstavlja onesnaževanje okolja. Onesnaženost je v večini primerov posledica dejavnosti ljudi, ni pa nujno. Ko danes govorimo o onesnaževanju, gre v večini primerov za posledice prekomernih izpustov kemičnih in bioloških dejavnikov ter delcev iz industrijske proizvodnje, kmetijstva, prometa in gospodinjstva.

Mi smo se osredotočili na onesnaževanje, ki ga povzroča promet, saj se nam je zdelo, da lahko ravno pri tem prispevamo velik del. V prometu je v zadnjem desetletju poleg hrupa glavna težava povečana koncentracija ogljikovih dioksidov zaradi povečanega obsega osebnega in tovornega avtomobilskega prometa. Ogljikovi dioksidi, kot benzen, ki nastaja, ko bencinski motorji niso ogreti na delovno temperaturo, povzročajo raka in so največja težava ob glavnih prometnih žilah in v mestnih središčih.

Ravno z razvojem naše aplikacije CarbonTracker pa smo poizkušali opozoriti ljudi na zavedanje o onesnaževanju, ki ga povzroča promet, in posledično zmanjšati onesnaženost okolja, natančneje ozračja.

Pri razvoju sem veliko časa posvetil določenim aktivnostim, ustvarjanju XML podatkovne baze, sinhronizaciji podatkov med podatkovnima bazama in spletnemu delu aplikacije.

2 Opis problema in izbrane tehnologije

2.1 Opis problema

Z razvojem sodobne družbe, kamor sodi predvsem mobilnost, se je zelo povečalo onesnaževanje okolja, predvsem ozračja. Prevozna sredstva, ki se uporabljajo v cestnem prometu in za pogon uporabljajo tekoča goriva, bi lahko uvrstili med najresnejše onesnaževalce okolja. Z modernizacijo družbe v Sloveniji smo dosegli to, da npr. imamo več kot na vsakega drugega prebivalca en osebni avtomobil. Ravno to pa omogoča pogostejše vožnje z osebnim avtomobilom, kar posledično pomeni večje onesnaževanje okolja, predvsem ozračja.

Če povzamemo uvod in strnemo naše besede, se naš problem nanaša na ne zavedanje ljudi o onesnaževanju okolja, natančneje ozračja. Z našo mobilno aplikacijo smo želeli doseči to, da bi ob njeni uporabi uporabnike opozorila in jim dala podatke, kako močno onesnažujejo okolje, predvsem ozračje ob nepravilni uporabi prevoznega sredstva. To je problem katerega smo se lotili v našem delu.

Naš prvotni cilj je bil implementacija rešitve, ki je namenjena širši množici uporabnikov pametnih telefonov z operacijskim sistemom Android. Drugi, prav tako pomemben, pa je bil osvetliti celotno teoretično ozadje naše aplikacije.

Kot dokaz nam je uspela implementacija naše rešitve, ki pa predstavlja osnovo za nadaljnje delo in razvoj dodatnih funkcionalnosti.

2.2 Izbrane tehnologije

Med študijem na Fakulteti za računalništvo in informatiko Univerze v Ljubljani smo veliko časa namenili programskemu jeziku Java, ki je trenutno eden vodilnih na svetu. Mobilnih aplikacij med študijem nismo razvijali, zato je to bila za nas nova izkušnja. Za Android aplikacijo smo se odločili zato, ker je trenutno ta operacijski sistem najbolj razširjen in znan med mobilnimi napravami in ravno v njem se kaže velik potencial.

Za razvoj mobilne aplikacije smo uporabili razvojno orodje Eclipse. V Eclipsu lahko razvijamo vse mogoče vrste aplikacij, od namiznih do spletnih in tudi mobilnih. Eclipse smo izbrali prav zaradi tega ker je eno vodilnih razvojnih orodij za Javo, prav tako pa spodbujajo njegovo uporabo na naši fakulteti. Našo mobilno aplikacijo smo razvijali v programskem jeziku Java, ki je temeljen za razvoj mobilnih aplikacij za operacijski sistem Android. Zraven smo še namestili programski razvojni paket (kratica SDK) za operacijski sistem Android, ki pa vključuje nabor celovitih orodij za razvijanje. Prav tako smo še namestili Eclipsov dodatek razvojnih orodij za Android (kratica ADT), ki omogoča zelo hiter razvoj mobilnih aplikacij. Za nadzorovanje izvedb projekta smo izbrali Subclipse. Aplikacijo pa smo zgradili po vzorcu MVC, ki omogoča ločitev uporabniškega vmesnika od poslovne logike.

Aplikacijo smo želeli narediti čim bolj privlačno in zanimivo, zato smo izbrali čim več tehnologij, saj nam vsaka tehnologija posebej ponuja nekaj novega. Izbrali smo sistem globalnega določanja položaja (kratica GPS), ki nam omogoča zaznavanje našega trenutnega položaja, prav tako pa tudi Google Maps, ki nam ponuja različne storitve glede zemljevidov

in v zvezi z njimi potovanj. Ena takih storitev je Google Transit, ki nam omogoča načrtovanje poti z različnimi tipi prevoznih sredstev.

Od podatkovnih tehnologij smo izbrali tehnologije XML, XPath in MySQL. XML smo izbrali, ker omogoča preprosto ustvarjanje značk in hranjenje podatkov znotraj njih, v njem pa smo po podatkih poizvedovali z XPathom, ki je preprost za uporabo. Kot strežniško odprtokodno programsko opremo za delo s podatkovnimi bazami pa smo izbrali tehnologijo MySQL. Za MySQL smo se odločili, ker je zastonj in smo ga poznali že od prej. Za izdelavo podatkovnega modela smo izbrali spet odprtokodno orodje MySQL Workbench, ki nam omogoča definiranje in ustvarjanje shem podatkovnih baz, dokumentiranje obstoječih podatkovnih baz in celo opravljanje zapletenih migracij na sistemu MySQL.

Za postavitev spletne aplikacije, ki teče na istem strežniku kot podatkovna baza MySQL, smo kot glavno platformo izbrali PHP. Ta teče na strežniku, torej ni nameščen pri odjemalcu. PHP smo izbrali zaradi tega, ker je preprost in ima dobro podprto knjižnico za delo s podatkovno bazo MySQL. Podatke smo iz podatkovne baze MySQL pridobili s pomočjo knjižnice v PHP-ju za delo z MySQL. Te podatke smo analizirali in prikazali na spletni strani. Za lepši prikaz podatkov smo izbrali kaskadne slogovne predloge CSS v povezavi z označevalnim jezikom HTML, ki pa sodita med najbolj pogosto uporabljene spletne tehnologije.

2.2.1 Razvojno okolje Eclipse

Eclipse je programsko razvojno okolje, ki vključuje integrirano razvojno okolje (kratica IDE) in razširljiv sistem dodatkov. Večinoma je napisano v Javi in ga lahko uporabljamo za razvoj aplikacij, napisanih v Javi, in tudi drugih programskih jezikih, kot so Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy in Scheme. V njem lahko razvijamo pakete za programsko opremo Mathematica.

Razvojno okolje Eclipse vsebuje javanska razvojna orodja (kratica JDT) za Javo, ki smo jih izbrali za razvijanje našega projekta, obstajajo pa še tudi druga, kot sta Eclipse CDT za C/C++ in Eclipse PDT za PHP. Eclipsov programski razvojni paket SDK (ki vključuje razvojna orodja za Javo) je mišljen za Java razvijalce, medtem ko razvijalci lahko naložijo dodatke za druge programske jezike.

V našem primeru smo naložili Eclipse IDE za Java razvijalce, Androidov programski razvojni paket (kratica SDK), Eclipsov dodatek razvojnih orodij za Android (kratica ADT) in na koncu povezali Androidov programski razvojni paket (kratica SDK) z Eclipsom.

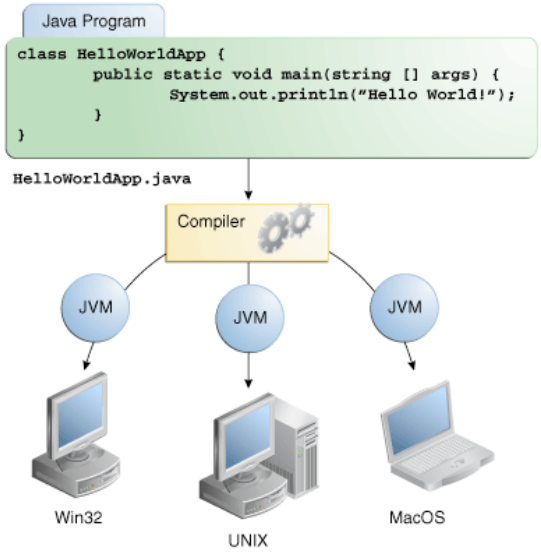
2.2.1.1 Java

Java tehnologija pomeni programski jezik in platformo.

2.2.1.1.1 Programski jezik Java

Java je tako imenovani visoki programski jezik, ki je označen z vsemi naslednjimi besedami: preprost, objektno orientiran, razdeljen, večniten, dinamičen, arhitekturno nevtralen, prenosen, visoko zmogljivosten, robusten in varen. V programskem jeziku Java je vsa izvorna koda zapisana v prazno tekstovno datoteko, ki se konča s končnico .java. Ta izvorna datoteka

se sestavi v .class datoteko z Java prevajalnikom, ki je sestavljena iz bajtov kode, jezika virtualnega stroja Java (Java VM).



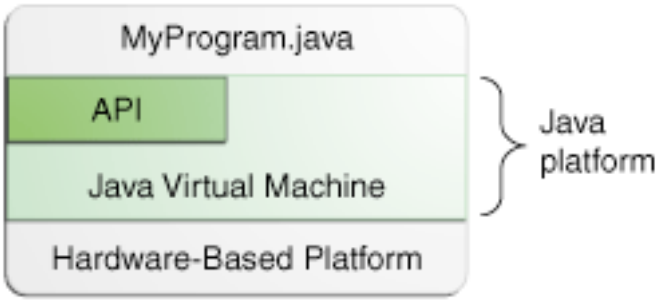
Slika 1: Virtualni stroj Java na več platformah

Kot vidimo, lahko aplikacija zaradi virtualnega stroja Java teče na več platformah.

2.2.1.1.2 Platforma Java

Platforma je strojno ali programsko okolje, v katerem program teče. Platforma Java se razlikuje od drugih platform, kot so Linux, Windows, Mac OS X, po tem, da je samo programska platforma, ki teče na vrhu drugih strojno baziranih platform. Platforma Java vsebuje dve komponenti: virtualni stroj Java in javanski vmesnik za programiranje aplikacij (kratica API).

API je velika zbirka pripravljenih programskih komponent, ki zagotavljajo veliko uporabnih zmogljivosti.



Slika 2: API in Virtualni stroj Java osamita program od strojne podlage

2.2.1.2 Android SDK

Androidov programski razvojni paket (kratica SDK) vključuje celovit nabor orodij za razvijanje: razhroščevalnik, knjižnice, emulator, dokumentacijo, primere kode in različne vodiče. Uradno podprto integrirano razvojno okolje (kratica IDE) je Eclipse, ki uporablja še zraven dodatek razvojnih orodij za Android (kratica ADT).

Androidov programski razvojni paket (kratica SDK) podpira starejše verzije Android platform, in če se razvijalci želijo usmeriti na razvoj aplikacij za starejše naprave, jim je to omogočeno.

Android aplikacije so zapakirane v .apk formatu in shranjene v mapi /data/app na operacijskem sistemu Android. Apk paketi vsebujejo .dex datoteke (zbrane kodne datoteke se imenujejo Dalvik executables), vir datotek itd.

2.2.1.3 ADT vtičnik (ang. plugin)

Dodatek razvojnih orodij za Android (kratica ADT) je dodatek za programsko razvojno okolje Eclipse, ki je zasnovano tako, da ponuja zmogljivo, integrirano okolje, v katerem lahko razvijamo Android aplikacije.

Dodatek razvojnih orodij za Android (kratica ADT) razširja zmožnosti Eclipsa in dopušča hitro vzpostavitev novih projektov, ustvarjanje aplikacij z uporabniškimi vmesniki, dodajanje paketov, ki temeljijo na Androidovem API-ju, odpravljanje napak aplikacij z uporabo programskega razvojnega paketa Android in uvoz .apk datotek.

Razvijanje v Eclipsu z dodatkom razvojnih orodij za Android (kratica ADT) je močno priporočeno, prav tako pa je to najhitrejši način za začetek razvijanja aplikacij. Z vodeno projektno nastavitvijo dodatek razvojnih orodij za Android (kratica ADT) zagotavlja, integracijo orodij, običajnih XML urejevalnikov in odpravljanje napak, ter neverjeten pospešeni razvoj aplikacij za operacijski sistem Android.

2.2.1.4 Subclipse

Za izdelavo našega projekta smo izbrali Subclipse, ki je Eclipsov ekipni nadzorni dodatek za zagotavljanje podpore subverzije znotraj integriranega razvojnega okolja (kratica IDE) Eclipse.

Za Subclipse smo se odločili zato, ker smo projekt razvijali v programskem razvojnem okolju Eclipse. Subclipse je dodatek k Eclipsu (angleško Add-on), uporaba Subclipsa pa je preprosta in učinkovita. Naš projekt smo razvijali s Subclipsom z namenom, da ne bi ogrozili delovanje našega programa, torej zaradi varnosti.

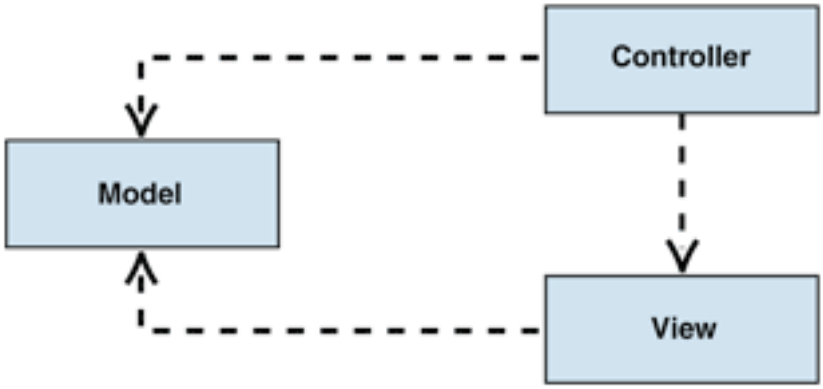
2.2.1.5 MVC

Našo aplikacijo smo razvili po tako imenovanem vzorcu model-pogled-nadzor (Model-View-Controller – MVC), ki razdeli aplikacijo na tri osnovne dele:

1. Model (angleško Model) usmerja vedenje in podatkovni del aplikacijske domene, se odziva na zahtevke in daje informacije o svojem stanju, pogosto pa daje tudi te zahtevke pogled. Odziva se tudi na navodila nadzornika, da spremeni svoje stanje.

2. Pogled (angleško View) usmerja prikazovanje informacij.

3. Nadzornik (angleško Controller) interpretira vnose uporabnika z miško ali tipkovnico in obvešča model in/ali pogled za uvedbo ustreznih sprememb.



Slika 3: MVC diagram

Kot vidimo, vzorec MVC ločuje uporabniški vmesnik od poslovne logike. To omogoča dober pregled nad celotno kodo in dobro strukturiranost projekta.

Vzorec MVC je v današnjih časih vse bolj popularen, prav tako pa določena razvojna okolja omogočajo ustvarjanje tako imenovanih MVC tipov projektov.

2.2.2 Spletne tehnologije

2.2.2.1 HTML

Hypertext Markup Language (v nadaljevanju HTML) je zaporedje ukazov, ki povedo, kako naj se predstavitvena stran prikaže. Ukazi se imenujejo tudi značke (angleško tags). Značke se vedno nahajajo med znakoma < in >, kot npr. <p>. HTML značke so največkrat uporabljene v parih kot npr. <h3> in </h3>, kljub temu pa jih je nekaj, ki niso v parih, tak primer je značka <img>. Prva značka se imenuje začetna značka, druga pa končna značka. Med ti dve znački lahko programerji ali grafični oblikovalci dodajo besedilo, značke, komentarje ipd.

HTML pri elementih in atributih dovoljuje velike ali male črke, na katere torej ni občutljiv (angleško case sensitive). Prav tako HTML dovoljuje slikam in objektom, da so vgrajeni in uporabljeni za ustvarjanje interaktivnih obrazcev. Vsebuje lahko tudi skriptne jezike, kot je

programski jezik JavaScript, prav tako pa se lahko sklicuje na CSS, ki se uporablja za stilsko predstavitev besedila, slik.

Primer preproste HTML strani:

```
<html>
   <head>
      <title>To naslov naše strani</title>
   </head>
   <body>
      <p>To je en odstavek</p>
   </body>
</html>
```

2.2.2.2 CSS

Cascading Style Sheets (v nadaljevanju CSS) so predloge, predstavljene v obliki preprostega slogovnega jezika, ki skrbi za predstavitev spletnih strani. Z njimi lahko definiramo stil HTML-ja oziroma XHTML-ja, lahko pa tudi drugih, npr. XML elementov v smislu pravil, kako se naj ti prikažejo na strani. Določamo lahko barve, velikosti, odmike, poravnave, obrobe, pozicije in vrsto drugih lastnosti, prav tako pa lahko nadziramo aktivnosti, kot je npr. prekritje povezave z miško.

Bistvo uporabe CSS je poleg definiranja pravil predvsem ločitev vsebine strani, ki jo podaja označevalni jezik skupaj z vsebino od njene predstavitve. S tem omogočimo lažje urejanje in dodajanje stilov ter poskrbimo za večjo preglednost dokumentov, ki temeljijo na HTML sintaksi. Prav tako omogočimo strani uporabo istih predlog, kar posledično pripelje do zmanjšanje dolžine kode.

Primer vključitve preproste CSS kode znotraj HTML strani:

```
<head>
   <style type="text/css">
      hr {color:SpringGreen;}
      p {margin-left:15px;}
      body {background-image:url("slike/nik.gif");}
   </style>
</head>
```

2.2.2.3 PHP

Hypertext Predprocessor (v nadaljevanju PHP) je jezik, ki je prerasel svoje ime. Izvirno je bil zamišljen kot niz makrov, ki bi piscem kode pomagali pri vzdrževanju osebnih domačih strani, ime pa se je razvilo iz njegovega namena. Kakor so se povečevale zmožnosti PHP-ja, tako se je večala tudi njegova priljubljenost.

Danes je že razvito veliko ogrodij, ki omogočajo lažji in hitrejši razvoj spletnih aplikacij s PHP-jem. Primer takih ogrodij za razvijanje s PHP-jem: CakePHP, Zend, Codelgniter, Yii itd.

PHP je strežniški skriptni jezik, pogosto zapisan v povezavi s HTML-jem. V nasprotju z običajno HTML stranjo, strežnik PHP skripta ne pošlje neposredno odjemalcu, ampak ga razčleni sam pogon PHP. Elementi HTML so v skriptu izpuščeni, koda PHP pa je prevedena in izvedena. S kodo PHP v skriptu lahko poizvedujemo po zbirkah podatkov, izdelujemo slike, beremo in zapisujemo datoteke ter komuniciramo z oddaljenimi strežniki.

PHP je lahko nameščen tudi kot aplikacija, ki se izvaja iz ukazne vrstice, zaradi česar je odlično orodje za pisanje skript na strežniku.

Primeri preproste PHP kode znotraj HTML strani:

```
<html>
   <head>
      <title>PHP znotraj HTML</title>
   </head>
   <body>
      <?php echo '<p>Pozdravljen svet!</p>'; ?>
   </body>
</html>
```

2.2.3 Podatkovne tehnologije

2.2.3.1 MYSQL

MySQL je sistem, za upravljanje s podatkovnimi bazami oziroma je odprtokodna implementacija relacijske podatkovne baze, ki za delo s podatki uporablja jezik SQL. V našem primeru smo MySQL izbrali pri strežniškem delu, kamor se shranjujejo podatki, ki se pridobijo pri izvajanju naše mobilne aplikacije in se kasneje uporabijo za prikaz podatkov na spletni strani, kjer si uporabnik lahko ogleda analizo sledenja svojemu premikanju.

V našem primeru smo ga izbrali v kombinaciji s programskim jezikom Java in skriptnim jezikom PHP. Pri programskem jeziku Java smo si pomagali z javanskim gonilnikom, ki omogoča manipulacijo z MySQL.

MySQL deluje na principu odjemalec-strežnik, pri čemer lahko strežnik namestimo kot sistem, porazdeljen na več strežnikih. Sistem MySQL je napisan v programskih jezikih C in C++, deluje pa lahko na več različnih operacijskih sistemih. Uporablja zelo hitre diskovne tabele MyISAM s stiskanjem indeksov.

2.2.3.2 XML

Extensible Markup Language (v nadaljevanju XML), ki je bil razvit leta 1998, je množica pravil za kodiranje dokumentov v obliko, ki je razumljiva računalniku. Specifikacijo, ki je okleščena različica SGML, so razvili pri W3C. XML je bil razvit, da bi stvari v internetu postale bolj enostavne, splošnejše in uporabnejše.

XML je označevalni jezik, ki je ustvarjen za oblikovanje besedil in določanje strukture dokumentov. Za dosego tega uporabljamo ukaze, ki so napisani kar v besedilu samem. XML si predstavljamo kot posebno zaporedje znakov, ki ga je lahko preprosto ločiti od ostalega besedila. Ima močno Unicode podporo za jezike povsod po svetu.

Zelo lepa lastnost XML-ja je ta, da lahko v njem ustvarjamo lastne oznake. To namreč omogoča, da so podatki med različnimi aplikacijami in organizacijami dobro definirani in da se jih med njimi da prenašati, pravilno interpretirati in preveriti. Za prikazovanje XML podatkov je priporočeno izbrati transformacije razširljivega slogovnega jezika (kratica XSLT), ki so mu bližje kot CSS.

Lahko ga izberemo kot spletno tehnologijo, kjer ustvarimo XML dokument s svojimi značkami in ga prikažemo na spletu. Prav tako pa ga lahko izberemo kot podatkovno bazo, kjer si tudi definiramo svoje značke, med njih pa vstavimo podatke, ki jih pozneje poizvedujemo s pomočjo imena značk ali drugih parametrov.

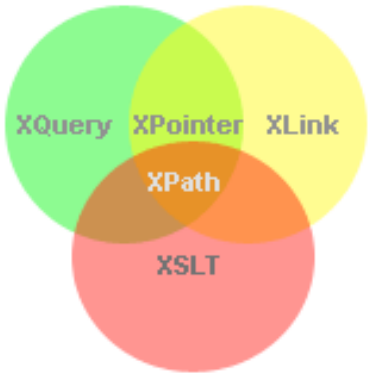
Od leta 2009 je bilo razvitih na stotine takšnih jezikov, med njimi so najbolj znani: XHTML RSS, Atom, SOAP, VoiceXML, Twitter Markup Language, Office Open XML, OpenDocument.

Primer preproste XML datoteke:

```
<?xml version="1.0"?>
<note>
   <to>To</to>
   <from>Nik</from>
   <heading>Reminder</heading>
   <body>Do it!</body>
</note>
```

2.2.3.3 XPath

Za pridobivanje podatkov iz naše drevesne strukture XML smo izbrali jezik za iskanje informacij znotraj XML dokumentov imenovan XPath. XPath smo spoznali že prej na fakulteti pri predmetu spletno programiranje in se mi je zdel najbolj primeren za naš primer.



Slika 4: Zbrane tehnologije, ki tvorijo XPath

XPath je pravzaprav sintaksa za definiranje delov XML dokumenta. 16. novembra 1999 je postal W3C priporočilo (angleško W3C Recommendation). Namenjen je za uporabo z transformacijami razširljivega slogovnega jezika (kratica XSLT), XPointerjem in ostalo razpoznavno XML programsko opremo.

Preprost primer poizvedovanja z XPathom:

```
/trgovina/kruh[1]
```

Zgornji primer nam izbere prvi element značke kruh, ki je otrok od značke trgovina.

2.2.4 Googlove tehnologije

2.2.4.1 Google Maps

Google Maps, slovensko Google zemljevid, je prosto dostopen strežnik z geografskimi podatki in zemljevidi. Omogoča nam visoko resolucijsko antensko ali satelitsko sliko za večino mestnih področij po vsem svetu. Vsebuje zemljevide držav sveta, zemljevide večjih mest in satelitske slike celega sveta. Vključuje številne storitve zemljevidov, kot so Google Maps spletna stran, Google Ride Finder, Google Transit in mape, vgrajene v tretjeosebne spletne strani preko Google Maps API-ja. Ponuja mape ulic, načrtovanje poti za potovanje peš, z avtom, kolesom, javnim prevozom in mestni poslovni lokator za veliko držav po svetu.

V Sloveniji ulični pogled ni možen. Omogoča nam tudi prikaz navodil za premik iz točke A v točko B.

2.2.4.2 Google Transit

V decembru 2005 se je v Googlovih laboratorijih začel razvijati Google Transit, ki je 20 % projekt Chrisa Harrelsona in Avichala Garga. Z njim lahko načrtujemo poti z različnimi tipi prevoznih sredstev. Z začetka je imel samo podporo za Portland in Oregon, zdaj pa že vključuje na stotine mest v Združenih državah Amerike, Kanadi, Evropi, Aziji, Afriki, Avstraliji, Indiji in Novi Zelandiji. Storitev izračuna pot, čas in stroške in lahko primerja potovanje enega prevoznega sredstva z drugim. Oktobra 2007 je uradno izšel Google Transit, razvit v Googlovih laboratorijih in od takrat naprej je integriran v Google Maps.

Pokritost Google Transita je javno dostopna in je razširjen po vsem svetu, včasih celo po celotnih državah, kot so Kitajska, Japonska in Švica.

V naši aplikaciji smo ga izbrali za načrtovanje poti peš, z osebnim avtomobilom in z avtobusom.

2.2.5 GPS

Sistem globalnega določanja položaja (kratica GPS) je navigacija in orodje za natančno določanje položaja. Razvil ga je Department of Defense leta 1973. GPS je bil prvotno zasnovan za pomoč vojakom in vojaškim vozilom, letalom in ladjam, bolj natančno določanju njihove lokacije po vsem svetu. Danes se je uporaba GPS-a razširila tako v komercialne namene kot tudi znanstvene. V komercialnih namenih se GPS uporablja kot navigacija in kot orodje za določanje položaja v letalih, ladjah, avtomobilih in še v nekaterih drugih rekreacijskih dejavnostih, kot so pohodništvo, ribolov in še nekatere druge. V znanstvenih namenih igra GPS pomembno vlogo pri vedi o Zemlji. Meteorologi ga uporabljajo za napovedovanje vremena in različne študije o podnebju. Geologi pa ga lahko uporabljajo za različne študije o potresih.

2.2.6 Mobilni operacijski sistem Android

Android je operacijski sistem za mobilne naprave, kot so pametni telefoni in tablični računalniki, ki temeljijo na operacijskem sistemu Linux. Razvija ga konzorcij Open Handset Alliance, katerega vodi Google, člani pa so še druga podjetja.

Google je podprl začetno razvijanje programske opreme podjetja Android Inc., katerega je tudi kupil leta 2005. Leta 2007 je bila izdana prva verzija Android distribucije in tudi takrat je bil ustanovljen konzorcij Open Handset Alliance. Google izdaja Android kodo kot odprtokodno, znano kot Android Open Source Project (kratica AOSP), s čimer podpirajo razvoj operacijskega sistema s strani zunanjih razvijalcev.

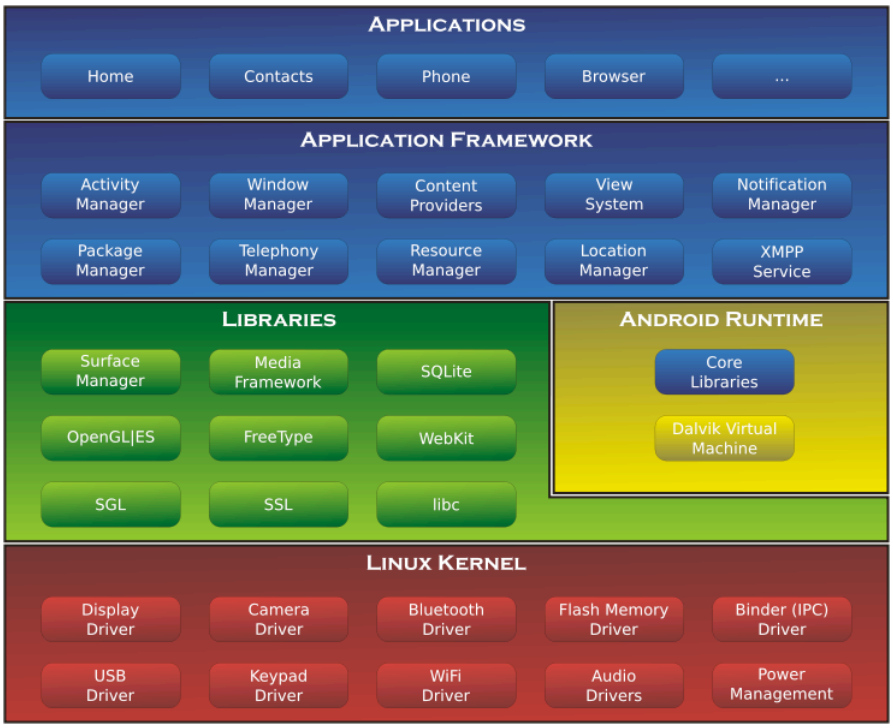
Operacijski sistem Android ima odprto tržišče Trgovino Play (angleško Google Play) za prodajanje in distribucijo aplikacij. Trgovina Play omogoča nadzor nad prodajanjem proizvodov, ki jih lahko uporabnik, ki jih je razvil, prosto objavi na odprtem tržišču. Uporabniki Androida si vsak mesec prenesejo preko Trgovine Play več kot 1.5 milijona aplikacij in iger.

Android je postal svetovno vodilna platforma za pametne telefone ob koncu leta 2010. V prvem četrtletju 2012 je Android pokrival 59 % tržnega deleža pametnih telefonov po svetu. Na sredini leta 2012 pa je bilo 400 milijonov naprav aktiviranih z operacijskim sistemom Android.

2.2.6.1 Android arhitektura

Android vsebuje jedro, ki bazira na Linux jedru z vmesno programsko opremo, knjižnicami in programskim vmesnikom, napisanim v programskem jeziku C, in aplikacijsko programsko opremo, ki teče na aplikacijskem ogrodju in vsebuje knjižnice.

Android uporablja virtualni stroj Dalvik (angleško Dalvik virtual machine) z izboljšano zmogljivostjo časa delovanja programa za zagon Dalvik dex-code (Dalvik Executable), ki je pogosto prevedena iz Java bajt kode.



Slika 5: Arhitektura operacijskega sistema Android

Kot lahko vidimo na zgornji sliki, so na najvišjem nivoju operacijskega sistema Android aplikacije. Te nam omogočajo pregled kontaktnih številk, brskanje v internetu, telefoniranje itd.

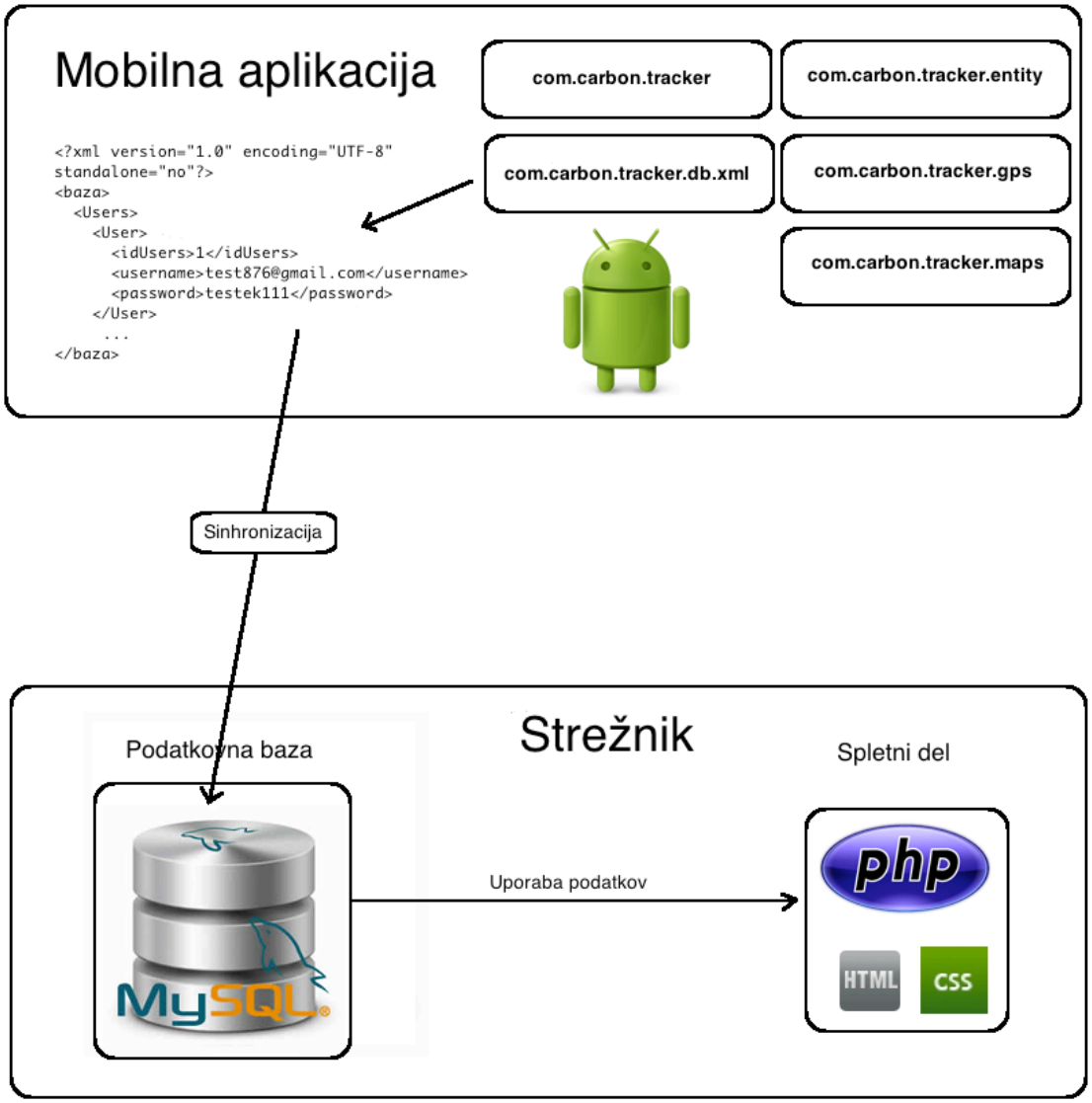
En nivo nižje se nahaja aplikacijsko ogrodje, ki vsebuje različne upravljavce. Ti nudijo aplikacijam različne opcije. Prav tako še na tem nivoju najdemo pogled sistema (angleško View System), ki nam omogoča gradnjo grafičnega vmesnika, in ponudnike vsebine (angleško Content Providers), ki nam delijo podatke med aplikacijami.

En nivo nižje najdemo dve vrsti knjižnic (angleško Libraries), ene so napisane v C/C++ (zelen prostor), druge pa v Javi (rumen prostor – Android Runtime).

Najnižje pa je Linux jedro (angleško Linux kernel), kjer imamo gonilnike, ki omogočajo delovanje strojnih naprav.

3 Arhitektura aplikacije

Arhitektura celotne aplikacije je sestavljena iz dveh večjih delov: mobilne aplikacije in strežnika. Strežnik pa je sestavljen iz dveh manjših delov, in sicer iz MySQL podatkovne baze in spletnega dela, kjer uporabnikom prikazujemo podatke na spletni strani. V naslednjih poglavjih so podrobneje predstavljeni vsi ti deli.



Slika 6: Arhitektura naše aplikacije

3.1 Paketi in deli aplikacije

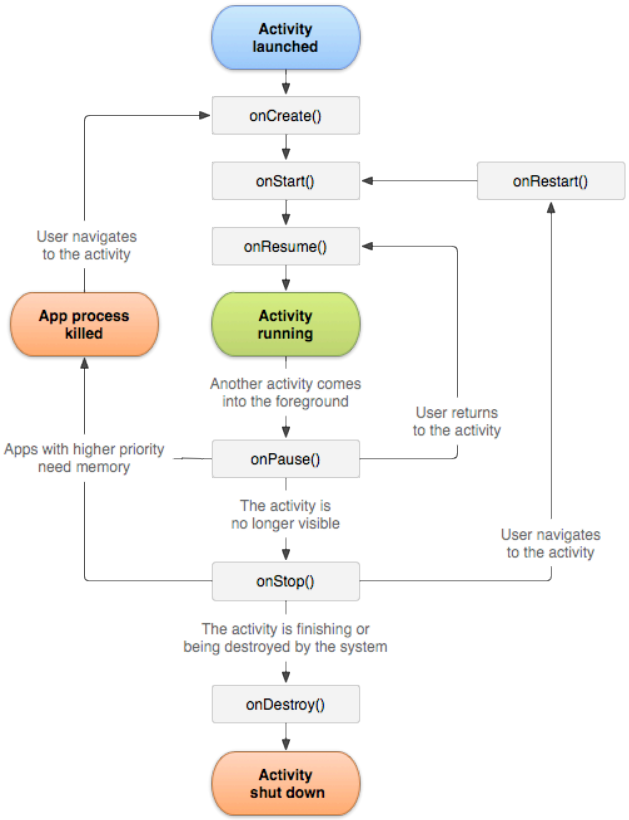
Našo mobilno aplikacijo smo strukturirano razdelili v pet večjih paketov, in sicer po funkcionalnostih. Ta strukturirana razdelitev nam omogoča boljšo preglednost kode, in tudi to, da se hitreje znajdemo med programsko kodo.

3.1.1 Paket com.carbon.tracker

V prvem paketu hranimo vse naše aktivnosti, zraven pa še javansko datoteko Constants.java. Ta datoteka Constants.java hrani vse naše konstante, ki jih uporabljamo v našem projektu. Tak način programiranja, da hranimo konstante v posebni datoteki, je uporaben zato, ker lahko to datoteko uporabimo tudi v drugih projektih in jo samo modificiramo, torej dodamo ali odstranimo kakšno konstanto.

Vsaka aktivnost je samostojna, neodvisna od drugih, torej je nekaj, kar lahko uporabnik dela. Skoraj vse aktivnosti komunicirajo z uporabnikom, zato razred Activity skrbi za ustvarjanje okna, v katerega lahko postavimo uporabniški vmesnik z metodo setContentView(View).

Če aktivnosti uporabljamo z Context.startActivity(), morajo imeti vsi razredi aktivnosti ustrezno deklaracijo <activity> v datoteki AndroidManifest.xml.



Slika 7: Življenjski cikel aktivnosti

3.1.2 Paket com.carbon.tracker.db.xml

V tem paketu imamo programsko datoteko XML.java, v kateri so metode za manipulacijo z našo lokalno XML bazo. Pri tem si pomagamo z XPathom in pa s knjižnicami za delo z XML datotekami v Javi. V spodnjem primeru lahko vidimo metodo, katera vrne objekt tipa User z uporabniškim imenom uporabnika iz XML podatkovne baze glede na podani ID, pri tem pa si pomagamo z XPathom:

```
public User selectUser(int id) throws ParserConfigurationException, SAXException,
IOException, XPathExpressionException {

   initDoc();

   XPathFactory factory = XPathFactory.newInstance();
   XPath xpath = factory.newXPath();
   XPathExpression username = xpath.compile("//Users/User[idUsers='"
   +id+"']/username/text()");
   Object resultUsername = username.evaluate(doc, XPathConstants.NODESET);
   NodeList nodesUsername = (NodeList) resultUsername;
   User user = new User();
   user.setUsername(nodesUsername.item(0).getNodeValue());
   return user;
}
```

3.1.3 Paket com.carbon.tracker.entity

Paket vsebuje več datotek; vsaka entiteta, kot sta npr. Pot ali Uporabnik, pa ima svojo javansko datoteko. Ta datoteka pa vsebuje zasebne spremenljivke in tako pridobitelje (angleško getterje) in nastavljavce (angleško setterje), torej metode, ki vračajo in nastavljajo vrednosti teh zasebnih spremenljivk.

Prednost takega načina programiranja je v tem, da do zasebnih spremenljivk nihče ne more dostopati in jih spreminjati od zunaj.

Glavna prednost objektno orientiranega programiranja je ponovna uporaba programske kode. Spodaj lahko vidimo primer iz naše kode.

Zasebna spremenljivka:

```
private Date startTime;
```

Primer metode tipa getter:

```
public Date getStartTime() {
   return startTime;
}
```

Primer metode tipa setter:

```
public void setStartTime() {
   this.startTime = new Date();
}
```

3.1.4 Paket com.carbon.tracker.gps

Ta paket vsebuje javanski programski datoteki GPSCallback.java in GPSManager.java. GPSManager ima implementirane določene poslušalce, pridobitelje in nastavljavce, uporablja pa že prej omenjeni vmesnik (angleško interface). GPSManager vsebuje metodo startListening(), ki nam v kombinaciji z drugimi metodami poizkuša pridobiti lokacijo, kjer se trenutno nahajamo. Dodana je še metoda stopListening(), ki omogoča zaustavitev GPS klica ob zaustavitvi aplikacije. Brez te metode bi ob zaprti aplikaciji GPS še vedno bil v teku.

3.1.5 Paket com.carbon.tracker.maps

Naš zadnji paket, ki pa, kot že ime pove, vsebuje programske datoteke z metodami za delo z mapo. Te metode nam omogočajo: preračunavanje podatkov na mapi, deklariranje razredov, kjer hranimo vse podatke o aktivnosti, razred za risanje po mapi, rokovanje s KML podatki in razčlenjevanje KML podatkov.

3.2 Podatkovna baza XML na napravi mobilne aplikacije

Pri načrtovanju naše aplikacije smo se odločili, da bomo uporabljali lokalno XML bazo za shranjevanje podatkov, kateri se ob koncu sledenja premikanju sinhronizirajo s podatkovno bazo MySQL na strežniku.

S tem smo pridobili to, da nam med izvajanjem aplikacije ni potrebno neprestano biti povezani na internet, saj naša aplikacija shrani podatke na strežnik ob koncu izvajanja mobilne aplikacije, kjer se podatki uporabijo za prikaz analize podatkov na spletni strani.

Za zapis podatkov v XML datoteko smo si pomagali z XPathom. Ta nam omogoča enostavno in pregledno dodajanje elementov in njihovih otrokov v XML datoteko.

Del naše lokalne XML baze:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<baza>
   <Users>
      <User>
         <idUsers>1</idUsers>
         <username>test56789@gmail.com</username>
         <password>test</password>
      </User>
   </Users>
   <carbonData>
      <carbonDataInstance>
         <idCarbonData>2</idCarbonData>
         <emitterName>carbon</emitterName>
         <gramsCarbon>2.75</gramsCarbon>
         <Road_idRoad>6</Road_idRoad>
         <Users_idUsers>1</Users_idUsers>
      </carbonDataInstance>
   </carbonData>
   <Road>
      <RoadInstance>
         <idRoad>3</idRoad>
```

```
        <Users_idUsers_r>1</Users_idUsers_r>
        <length>4.67</length>
        <TravelMode_idTravelMode>5</TravelMode_idTravelMode>
        <travelTime>00:05</travelTime>
        <DateTime>2012-08-17 21:32:54</DateTime>
      </RoadInstance>
   </Road>
   <TravelMode>
      <TravelModeInstance>
        <idTravelMode>20</idTravelMode>
        <mode>Car</mode>
      </TravelModeInstance>
   </TravelMode>
   <MapPoints>
      <MapPoint>
        <idMapPoints>16</idMapPoints>
        <speed>5.265936740</speed>
        <Road_idRoad>3</Road_idRoad>
        <latitude>52.137859</latitude>
        <longitude>4.961944</longitude>
        <TravelMode_idTravelMode_m>20</TravelMode_idTravelMode_m>
      </MapPoint>
   </MapPoints>
</baza>
```

3.3 Strežnik

V celotni naši aplikaciji se nam je najbolj smiselno zdelo uporabiti dve podatkovni bazi. XML podatkovna baza, ki je na napravi, kjer teče mobilna aplikacija, katere podatki se ob koncu sledenja premikanju naše aplikacije sinhronizirajo s podatkovno bazo MySQL na strežniku.

3.3.1 Podatkovni model

Naš podatkovni model baze na strežniku je sestavljen iz šestih entitet in je relativno preprost. Za potrebe naše aplikacije nismo potrebovali bolj zapletenega modela, saj nam je že ta bil več kot dovolj. Kot lahko vidimo na sliki modela, uporabljamo različne vrste ključev, ki nam omogočajo različne omejitve, posledično pa boljšo strukturiranost podatkov.

V entiteti users so shranjena uporabniška imena, ki so v bistvu naslovi elektronskih pošt in gesla uporabnikov, s katerimi se lahko registrirani uporabniki prijavijo na spletno stran http://www.carbontracker.si, kjer si lahko ogledajo analizo preteklih sledenj premikanju mobilne aplikacije, prav tako pa se tudi s tem uporabniškim imenom in geslom prijavijo v mobilno aplikacijo.

Naslednja pomembna entiteta je road, v kateri so shranjeni osnovni podatki o enem sledenju premikanja. Ti podatki pa so: dolžina poti, čas potovanja, datum in ura začetka potovanja.

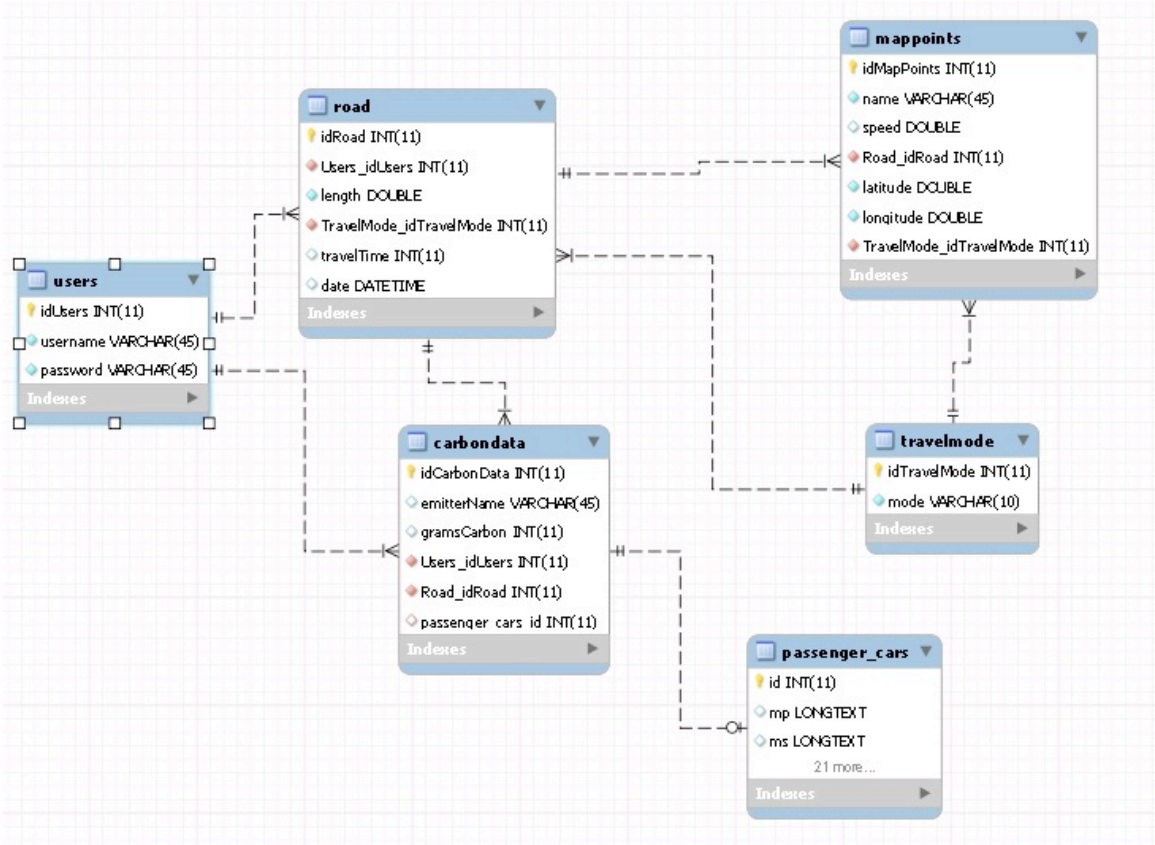
Entiteta travelmode je najbolj preprosta entiteta, ki pa vsebuje samo en atribut mode, ki vsebuje vrste načina potovanja. Za zdaj implementacija naše aplikacije vsebuje le tri vrste načina potovanja: z osebnim avtomobilom, z avtobusom in peš.

Naslednja naša entiteta je passenger_cars, kjer so shranjeni različni podatki o osebnih avtomobilih: znamka osebnega avtomobila, model, prostornina motorja itd.

Naš najpomembnejši podatek v tej entiteti, je povprečna poraba goriva specifičnega avtomobila.

Naslednja taka entiteta je carbondata, ki pa je povezana z entiteto users in passenger_cars. Ta vsebuje dva osnovna podatka o emisijskem viru: ime vira in število gramov onesnaževanega vira na razdalji enega kilometra.

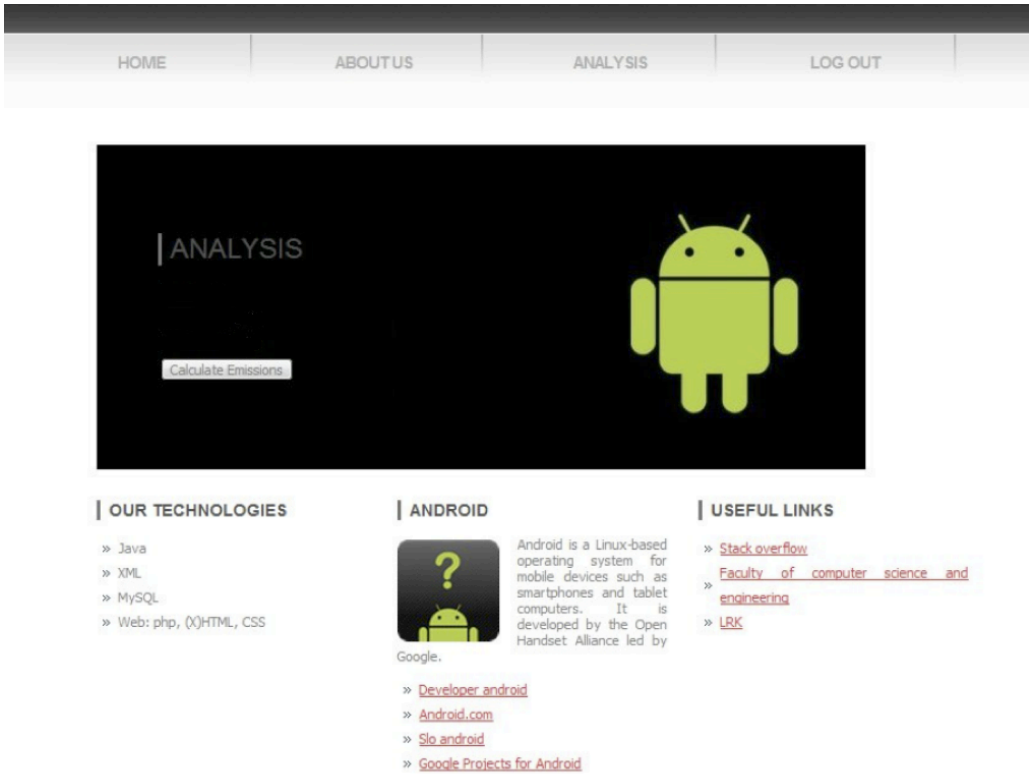
Zadnja naša entiteta na našem podatkovnem modelu je mappoints, kjer so shranjeni podatki o točkah na mapi med premikanjem: ime točke na mapi, naša trenutna hitrost v tej točki, geografska širina in dolžina točke.



Slika 8: Primer podatkovnega modela na strežniškem delu aplikacije

3.3.2 Spletni del

Ta del celotne naše aplikacije teče na strežniku, kjer teče tudi naš MySQL strežnik. To pomeni, da spletna aplikacija pobira podatke iz lokalne podatkovne baze MySQL in ne potrebuje dostopa do oddaljene podatkovne baze.

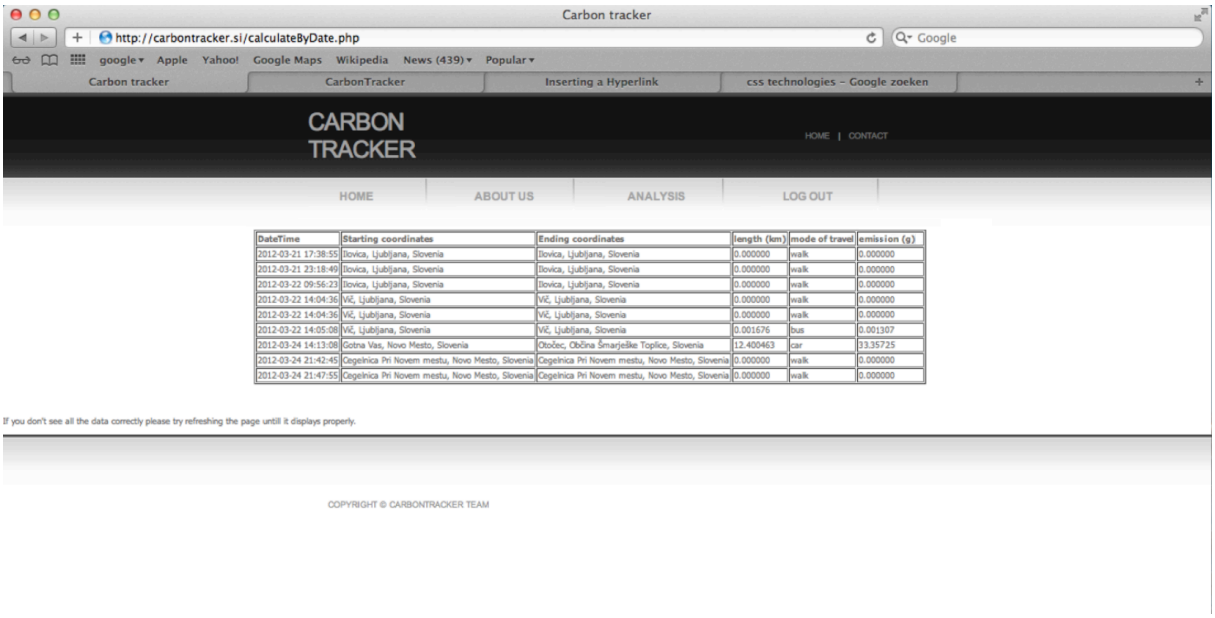


Slika 9: Spletna stran z gumbom za izračun podatkov

Namen spletnega dela je, da si uporabniki lahko ogledajo vsa svoja pretekla sledenja premikanju in kako močno so v teh premikanjih onesnažili okolje. Pri tem lahko vidijo datum in čas sledenja, začetni koordinati, kjer so premikanje začeli, končni koordinati oziroma mesto cilja, dolžino poti, način premikanja in pa koliko gramov ogljikovega dioksida so oddali v okolje s svojim prevoznim sredstvom.

Ob prvem prihodu na stran ima gost strani v meniju izbiro LOGIN, s čimer se lahko prijavi v spletno stran. Naslov elektronske pošte (uporabniško ime) in geslo uporabnika sta shranjena v podatkovni bazi MySQL, v podatkovno bazo pa sta se shranila, potem ko smo zagnali aplikacijo in nam je bila ponujena izbira registracije novega uporabnika, kasneje pa sinhronizacija podatkov s podatkovno bazo MySQL.

Po uspešni prijavi v spletno stran se v meniju uporabniku pojavi izbira ANALYSIS in če jo izbere, ga preusmeri na stran, kjer ima možnost izbire Calculate emissions (slovensko Izračunaj emisijo), gumb, ki posodobi podatke iz podatkovne baze. Akcija tega gumba ni nič drugega kot branje podatkov s pomočjo skriptnega jezika PHP iz podatkovne baze MySQL in prikaz teh podatkov v obliki dinamične tabele s kombinacijo skriptnega jezika PHP in označevalnega jezika HTML.



Slika 10: Primer prikaza podatkov o naših sledenjih

Kot lahko vidimo, smo za našo spletno stran uporabili tudi CSS za boljšo preglednost in ureditev strani. Naš namen ni bil ustvariti posebej oblikovano stran z izjemnim videzom, temveč le pregledno in dobro strukturirano stran.

4 Razvoj in delovanje aplikacije

V tem poglavju bomo podrobneje predstavili razvoj in delovanje naše aplikacije CarbonTracker.

V okviru univerzitetnega študijskega programa računalništvo in informatika se nismo srečali z razvojem mobilnih aplikacij. Na začetku smo imeli največ težav s pridobitvijo ideje in kako zadevo sploh načrtovati.

Pomislili smo na to, da se danes zelo onesnažuje okolje in da bi lahko na nek način poizkušali prispevati k zmanjšanju onesnaženosti okolja, bolj natančno ozračja. Tako smo se odločili za našo aplikacijo CarbonTracker, ki je sestavljena iz mobilne aplikacije in strežniškega dela, ki vsebuje podatkovno bazo in spletni del, kjer si kot prijavljeni uporabnik lahko ogledamo našo analizo sledenja premikanju.

V primeru izbire načina TRACK naša aplikacija komunicira s strežnikom s pomočjo internetne povezave (WIFI, GPRS), kar pomeni nek strošek za uporabnika naše aplikacije. Ta najprej shrani podatke v lokalno XML datoteko oziroma bolje rečeno podatkovno bazo, potem pa se podatki iz podatkovne baze XML sinhronizirajo z našim strežnikom in shranijo v podatkovno bazo MySQL. Prednost tega načina shranjevanja je ta, da vsakič, ko aplikacija shrani podatke v lokalno XML podatkovno bazo, prepiše že obstoječe podatke o premikanju in s tem preprečimo prekomerno shranjevanje podatkov na mobilni napravi (angleško overhead). Prav tako pa ni treba ves čas uporabljati internetne povezave (WIFI, GPRS).

Ko so podatki o sledenju shranjeni v podatkovni bazi MySQL, se je mogoče prijaviti v spletno stran s svojim naslovom elektronske pošte (uporabniško ime) in geslom, ter si ogledati analizo svojih sledenj premikanju. Ker je cel strežniški del dejansko neodvisen od operacijskega sistema mobilne naprave, bi ga lahko neodvisno uporabili od naše mobilne aplikacije.

V drugem načinu izbire DIRECTIONS tudi naša aplikacija komunicira z Googlovimi orodji in uporablja internetno povezavo (WIFI, GPRS). V tem primeru podatkov ne shranjujemo v podatkovno bazo, saj ni nobene potrebe po tem.

Ciljna množica uporabnikov naše aplikacije so uporabniki, na katere želimo vplivati z uporabo naše aplikacije, da bi izbrali ustrezno prevozno sredstvo za določeno razdaljo in posledično manj onesnažili okolje, podrobneje ozračje. Uporabniki pa so tudi tisti, ki si želijo s pomočjo te mobilne aplikacije shraniti podatke o sledenju svojemu premikanju in si ogledati različne analize teh podatkov.

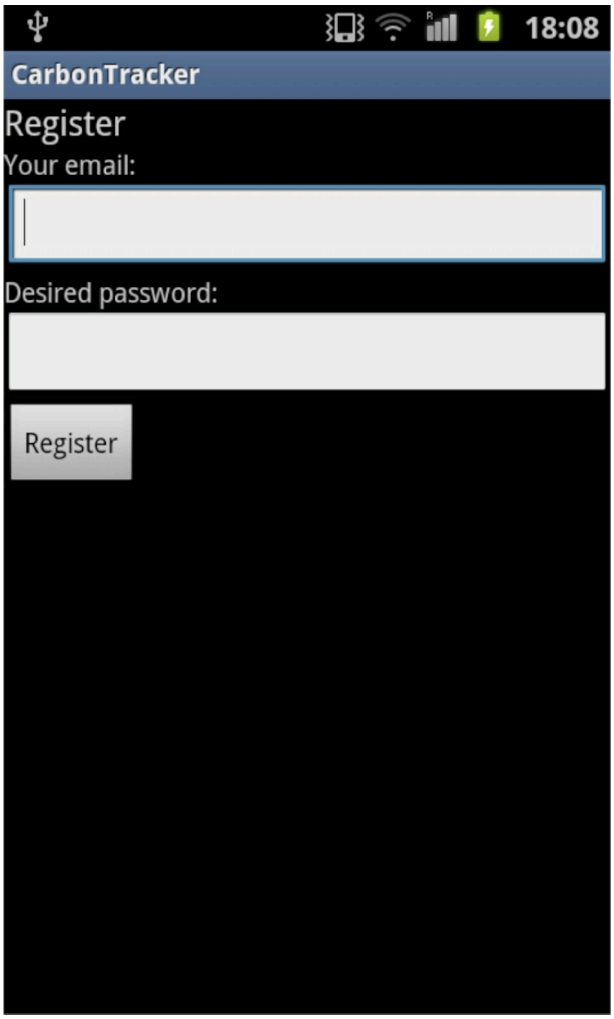
Za zagon aplikacije je potrebno imeti operacijski sistem Android, najmanj verzije 2.3.3 in pa tudi internetno povezavo na mobilni napravi (WIFI, GPRS), če pa ne želimo med delovanjem aplikacije uporabljati internetne povezave (WIFI, GPRS) v primeru sledenja premikanju, pa lahko namesto te uporabimo GPS.

Ob prvem zagonu aplikacije se nam pojavi okno, ki od nas zahteva registracijo. Kot smo že omenili, potrebujemo za uspešno delovanje aplikacije internetno povezavo (WIFI, GPRS), saj se po uspešni registraciji podatki o registriranem uporabniku shranijo v lokalno podatkovno

XML bazo na napravi, kjer teče mobilna aplikacija, ob koncu izvajanja aplikacije pa se podatki iz te podatkovne baze sinhronizirajo z bazo na strežniku. Ob vpisovanju podatkov se izvede tudi zelo preprosto preverjanje naslova elektronske pošte in gesla:

```
if(!(username.length() > 0 && password.length() > 0 )){
   Toast.makeText(LoginActivity.this, "Blank fields", Toast.LENGTH_SHORT).show();
}

if(!(username.contains("@")&& username.contains("."))){
   Toast.makeText(LoginActivity.this,      "You      have      to      enter      email",
   Toast.LENGTH_SHORT).show();
   return;
}
```

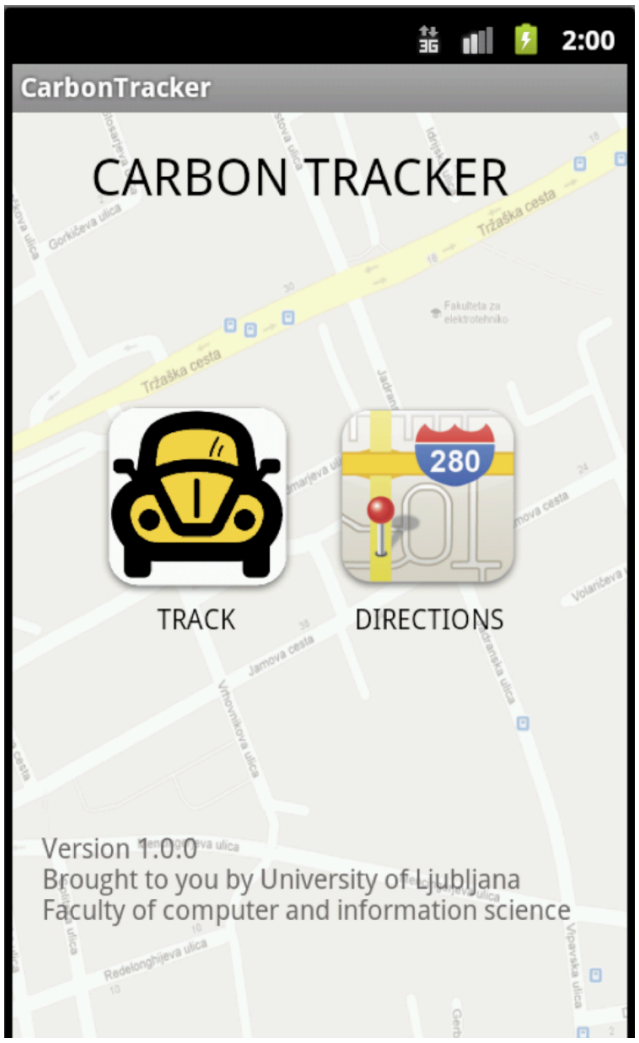


Slika 11: Primer registracije novega uporabnika

V nasprotnem primeru, če smo že registrirani uporabnik, imamo na naši mobilni napravi že ustvarjeno XML datoteko, kjer imamo shranjen naslov elektronske pošte (uporabniško ime) in geslo.

Zaradi varnostnih razlogov v prvem koraku aplikacija preveri, če že obstaja XML datoteka, in v primeru, da obstaja, pridobi iz nje naš naslov elektronske pošte (uporabniško ime) in geslo, ki se uporabita za preverjanje identifikacije uporabnika na našem strežniku. Če se naslov elektronske pošte (uporabniško ime) in geslo ujemata s podatki na strežniku, smo že obstoječi uporabnik in aplikacija avtomatično preskoči korak registracije novega uporabnika.

Ko smo registrirani uporabnik in imamo ustvarjeno XML datoteko, nas aplikacija preusmeri na naslednji korak, kjer lahko izberemo način delovanja naše aplikacije. Prvi način je TRACK, ki pomeni sledenje, drugi pa je DIRECTIONS, ki pomeni poti.



Slika 12: Izbira načina delovanja TRACK in DIRECTIONS

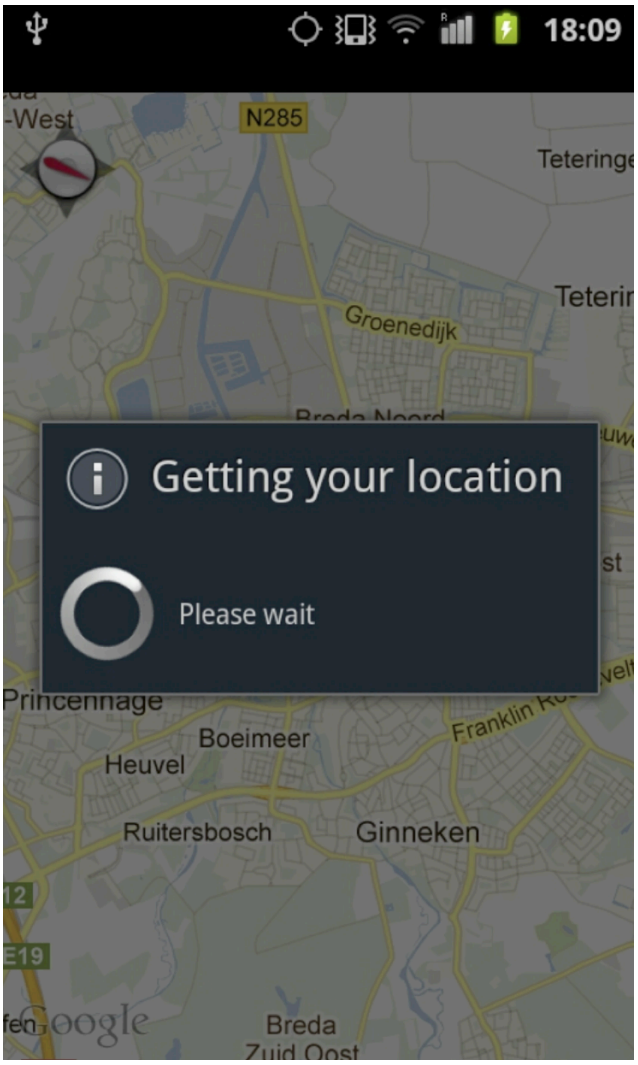
Načina sta si precej različna, zaradi tega lahko vsakega posebej uporabljamo za različne namene.

4.1 Izbira načina TRACK

V primeru načina TRACK (slovensko sledenje) bo aplikacija delovala tako, da bo sledila našemu premikanju in bo shranjevala točke na poti v lokalno XML bazo na napravi mobilne aplikacije. Ta način je uporaben v primeru, ko se bomo premikali in želimo, da naša aplikacija sledi našemu premikanju, si zabeleži našo pot, čas potovanja in pa, da izvemo, koliko gramov

ogljikovega dioksida smo oddali v okolje, torej kako močno smo onesnažili ozračje. Na koncu nam še vrne primerjavo, kako bi lahko bolj ali pa manj onesnažili ozračje, če bi izbrali drugo prevozno sredstvo.

V naslednjem koraku, ko izberemo katerokoli od teh izbir, bodisi TRACK bodisi DIRECTIONS, nam aplikacija ob uporabi internetne povezave (WIFI, GPRS) ali GPS-a poizkuša pridobiti našo trenutno lokacijo, kjer se nahajamo.

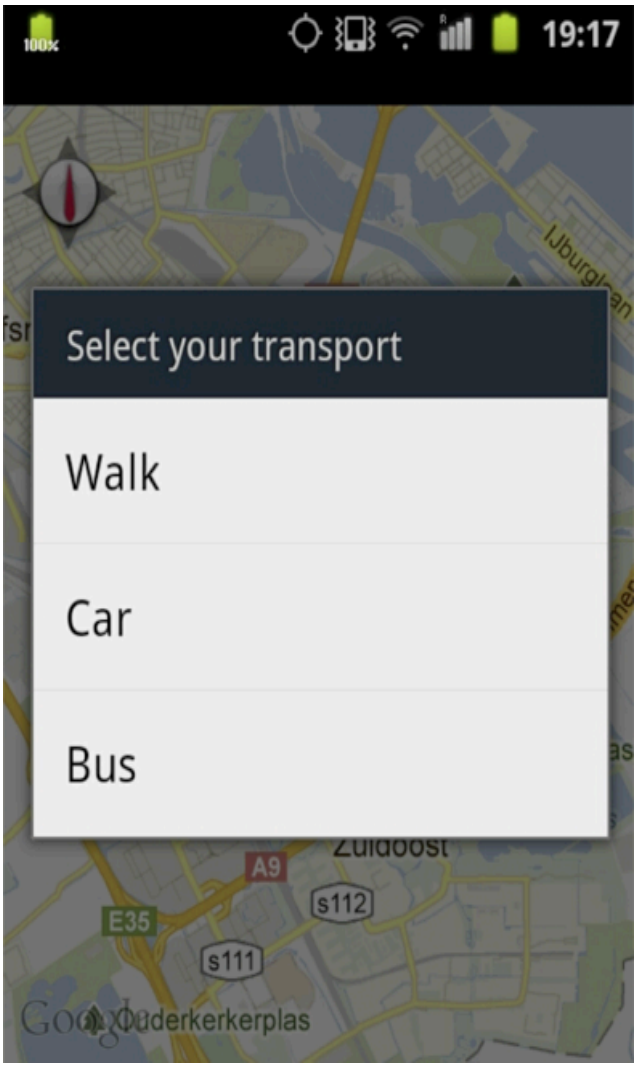


Slika 13: Pridobivanje naše trenutne lokacije

V tem koraku izvajanja aplikacije smo uporabili dialog napredka (angleško Progress Dialog in upravljavca lokacije (angleško Location Manager).

Prvi, ki razširja opozorilni dialog (angleško Alert Dialog), prikazuje stanje napredka in dodatno lahko vsebuje tekstovno sporočilo ali pogled. Upravljavec lokacije razširja objekt, torej nam ta razred omogoča dostop do sistemskih lokacijskih storitev. Te storitve omogočajo aplikaciji pridobitev redne posodobitve geografske lokacije ali osredotočenje aplikacije na točno določeno lokacijo takrat, ko preide aplikacija v bližino te geografske lokacije.

Ko aplikacija uspešno najde našo lokacijo, se fokusira na njo, zatem pa se pojavi naslednji dialog.



Slika 14: Izbira načina prevoza

Dialog nam ponudi tri možne izbire prevoznih sredstev: hoditi peš (Walk), peljati se z osebnim avtomobilom (Car) ali avtobusom (Bus). Dialog je v večini primerov majhno okno, ki se pokaže v ospredju trenutne aktivnosti (angleško Activity). Natančneje, uporabili smo opozorilni dialog, ki je najbolj pogost in omogoča vsebovanje izbirnih elementov in drugih, kot so npr. gumbi. V nadaljevanju je podan primer kode ustvarjanja našega dialoga za izbiro načina prevoznega sredstva.

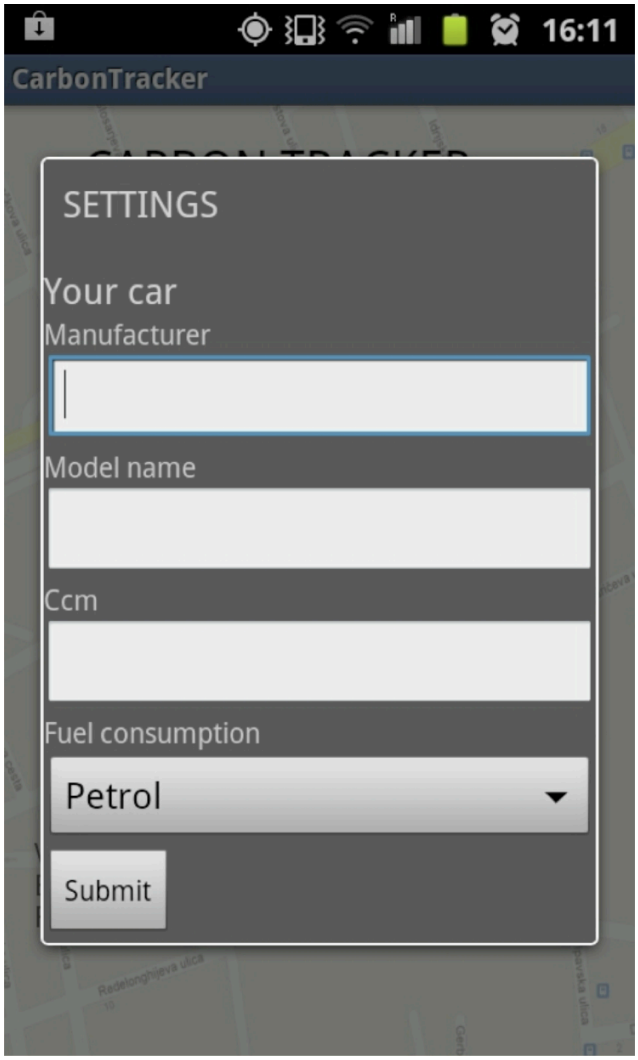
Primer kode ustvarjanja našega dialoga:

```
private void transportDialog() {
   final CharSequence[] items = {"Walk", "Car", "Bus"};
   AlertDialog.Builder builder = new AlertDialog.Builder(this);
   builder.setCancelable(true);
   builder.setOnCancelListener(new OnCancelListener() {

      public void onCancel(DialogInterface dialog) {
         if(roads.isEmpty()){
            Road road = new Road();
            road.setMode(1);
            roads.add(road);
         }else{
            roads.getFirst().setMode(1);
            roads.add(roads.getFirst());
         }
      }
   });
   builder.setTitle(R.string.means_transport);
   builder.setItems(items, new DialogInterface.OnClickListener() {

      public void onClick(DialogInterface dialog, int which) {
      //ravnaj se po int which.. ce je 0 = Walk, 1 = Car, 2 = Bus
         if(roads.isEmpty()){
            Road road = new Road();
            road.setMode(which);
            roads.add(road);
         }else{
            roads.getFirst().setMode(which);
            roads.add(roads.getFirst());
         }
      }
   });
   AlertDialog alert=builder.create();
   alert.show();
}
```

Kot vidimo, naša metoda vsebuje tudi dve podmetodi. Prva se pokliče, ko se prekine dialog, druga pa, ko se uporabnik dotakne katerega izmed elementov znotraj dialoga. Ti dve metodi nam omogočata manipulacijo z našim dialogom. Če izberemo osebni avtomobil (angleško

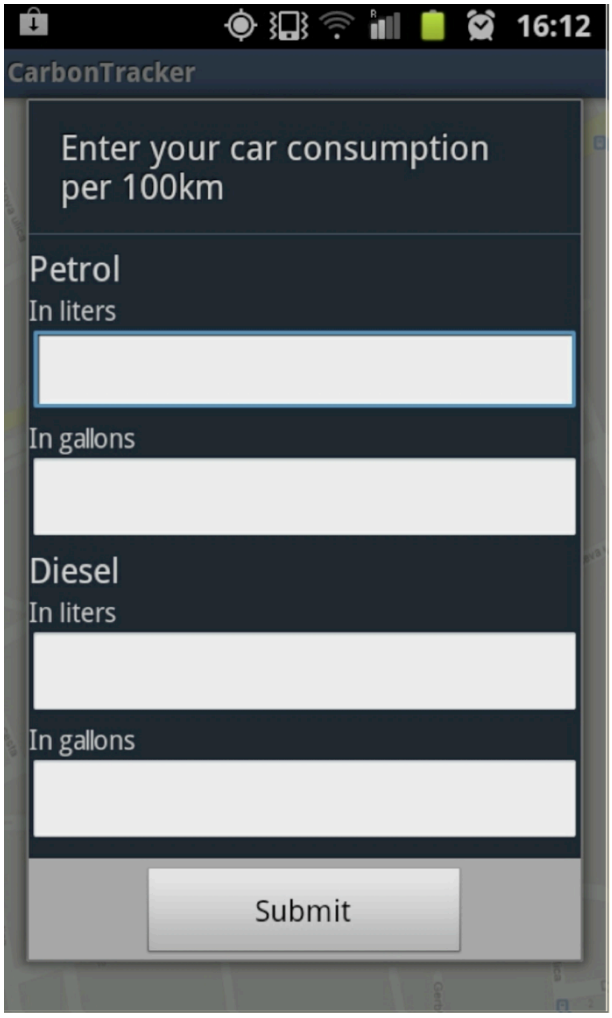


Slika 15: Vnos podatkov o želenem osebnem avtomobilu

Car), se nam pojavi naslednji dialog, ki je prikazan na zgornji sliki. Ta nam omogoča, da vpišemo znamko osebnega avtomobila, njegov model, njegovo prostornino in izberemo vrsto goriva. Po potrditvi se izvede naslednji stavek v programskem jeziku SQL:

```
String sql = "SELECT * FROM passenger_cars WHERE man LIKE '"+manufacturer+"' AND Cn
LIKE '"+modelName+"' AND `Ec (cm3)` BETWEEN "+ccm1+" AND "+ccm2+" AND Ft='"+fuel+"'
AND Va='"+mkNum+"'";
```

Ta iz naše podatkovne baze na strežniku poizkuša pridobiti podatek, koliko gramov ogljikovega dioksida sprošča v ozračje podani osebni avtomobil. Če pa osebni avtomobil s temi podatki ne obstaja v naši podatkovni bazi, se nam odpre dialog, ki nam ponudi, da sami vpišemo porabo goriva za naš avtomobil.

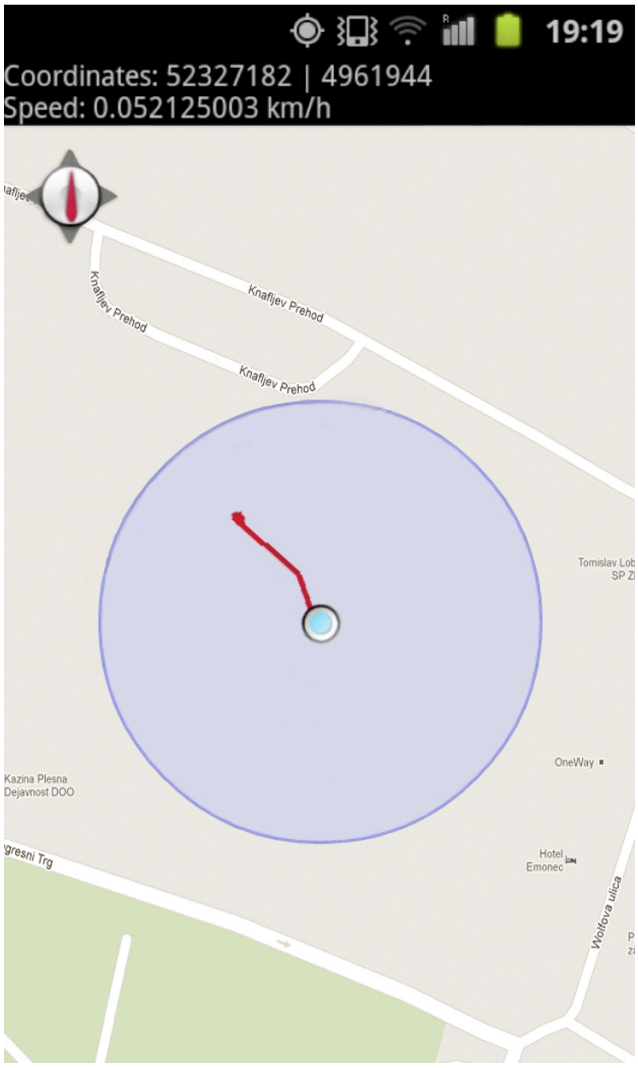


Slika 16: Vnos porabe goriva našega avtomobila

Pri vpisovanju imamo na izbiro dve vrsti goriva: bencin in dizel. Pri vsakem od teh goriv pa lahko še izberemo enoto za gorivo. Za to verzijo smo se odločili zato, ker v različnih državah uporabljajo različne enote za gorivo, npr. v Združenem kraljestvu, v Združenih državah Amerike in še kje drugje. V celotni aplikaciji, kot vidimo, je jezik angleščina, kar nam omogoča globalno uporabo aplikacije po svetu.

Če izberemo katero drugo prevozno sredstvo, se iz baze prebere podatek o tem, koliko ogljikovega dioksida v povprečju oddaja izbrano prevozno sredstvo.

Po uspešno izbranih parametrih začne aplikacija slediti našemu premikanju. Izrisovanje poti našega premikanja poteka v treh različnih barvah. V primeru izbire načina hoje se nam naša pot izrisuje v rdeči barvi, v primeru izbire avtobusa v zeleni barvi, in še zadnja opcija premikanje z osebnim avtomobilom v rumeni barvi. Med izrisovanjem poti se te točke na poti sproti shranjujejo v našo lokalno XML bazo, katere podatki se pozneje sinhronizirajo s strežnikom. Za uspešno izrisovanje poti smo si pomagali s komponento Canvans. Canvas je



Slika 17: Primer izpisa poti v načinu Walk

komponenta v Javi, lahko tudi v drugih programskih jezikih, ki nam prikazuje prazno pravokotno ploskev na zaslonu in omogoča aplikaciji risanje po tej ploskvi, ter lovljenje vhodnih dogodkov uporabnika.

Spodaj je podan primer naše metode, ki izrisuje našo pot, katera je sestavljena iz krajših črt:

```
public void drawPath(MapView mv, Canvas canvas) {

   ArrayList<MapPoint> mPoints = mRoad.getmRoute();

   int x1 = -1, y1 = -1, x2 = -1, y2 = -1;
   Paint paint = new Paint();

   paint.setStyle(Paint.Style.STROKE);
   paint.setStrokeWidth(4);

   for (int i = 0; i < mPoints.size(); i++) {
      Point point = new Point();
      Mappoint mp = mPoints.get(i);
      GeoPoint g = new GeoPoint(mp.getLatitudeE6(), mp.getLongitudeE6());
      paint.setColor(getColor(mPoints.get(i).mTravelMode));
      mv.getProjection().toPixels(g, point);
      x2 = point.x;
      y2 = point.y;

      if (i > 0)
         canvas.drawLine(x1, y1, x2, y2, paint);

      x1 = x2;
      y1 = y2;
   }
}
```

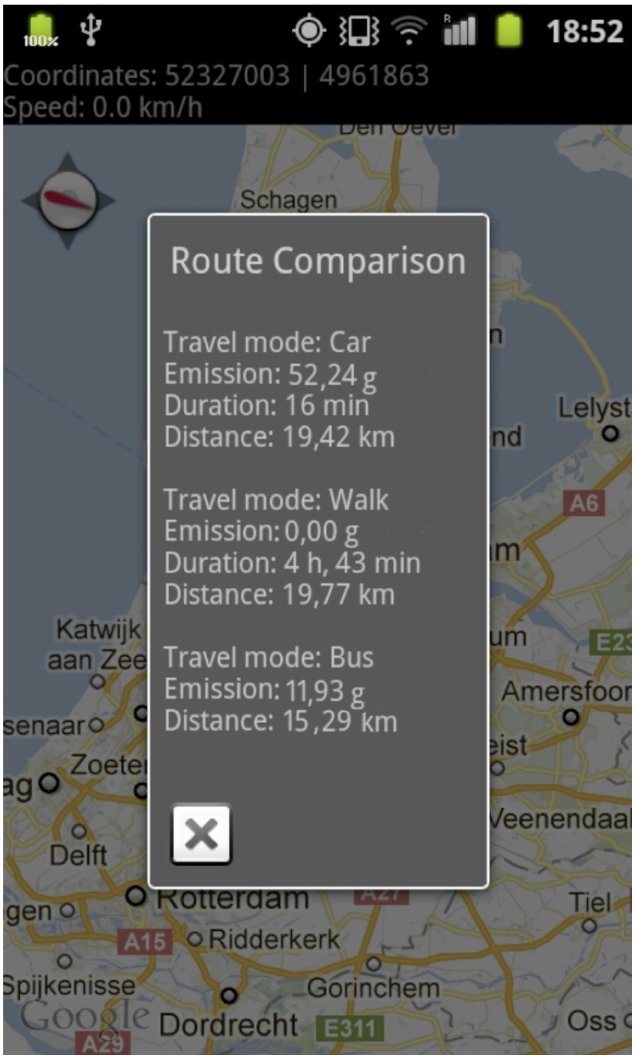
Naša metoda se sprehodi skozi seznam točk in nastavi projekcijo na geografsko širino in dolžino ter nariše linijo med točkama $T_1(x_1,y_1)$ in $T_2(x_2,y_2)$. Ko izriše linijo med tema točkama, nastavi vrednost koordinat točke T_1 na vrednost koordinat točke T_2, saj smo se premaknili na novo mesto, kjer sta vrednosti koordinat točke $T_{1(}x_1,y_{1)}$ enaki kot pri točki T_2.

Po tem nastavljanju novih koordinat točk sledi spet ponovno izvajanje zanke, kjer se postopek ponovi, pridobimo pa le nove koordinate, kamor smo se premaknili.

Ko želimo končati z našim sledenjem, imamo v aplikaciji na izbiro možnost end track (slovensko končaj sledenje). Po zaslonskem kliku na ta gumb se nam pojavi dialog primerjave poti (angleško Route Comparison), ki ni nič drugega kot preprosti dialog s prikazom podatkov o poteh.

Na naslednji sliki lahko vidimo, da nam ta dialog vrne vse tri vrste načine sledenja premikanju. Pri prvem načinu Car nam vrne emisijsko onesnaževanje, trajanje našega sledenja premikanju in pa tudi razdaljo, ki smo jo prebili. Iste vrste podatkov nam aplikacija vrne pri drugem načinu Walk, pri tretjem pa nismo vključili podatek o trajanju našega potovanja, saj je zelo težko predvideti, kako dolgo bi trajalo našo potovanje z avtobusom, v primeru, da ni bil izbran kot prevozno sredstvo.

Pri potovanju z avtobusom so namreč mogoča različna prestopanja, različno dolgo čakanje na naslednji avtobus in podobne zadeve, zato tega podatka o trajanju nismo vključili v našo aplikacijo.



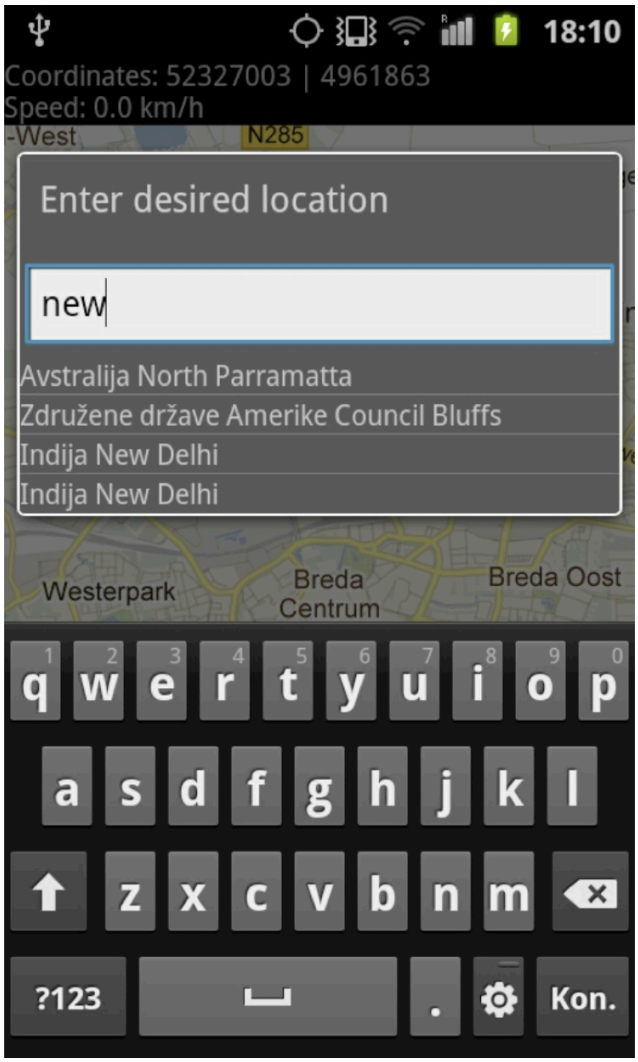
Slika 18: Primer primerjave poti

Preden se izvede in prikaže dialog, ki pokaže primerjavo poti, se podatki o našem sledenju premikanju sinhronizirajo s podatkovno bazo na strežniku. To lahko traja nekaj časa, zato je treba počakati.

4.2 Izbira načina DIRECTIONS

Ta izbira je precej drugačna od načina izbire TRACK, saj ta način ne zahteva od nas premikanja, da bi nam aplikacija sledila. Po prvem kliku na zaslonski gumb DIRECTIONS se nam pojavi isti dialog kot pri izbiri načina TRACK, torej dialog, ki s pomočjo internetne povezave (WIFI, GPRS) ali GPS-a najde našo lokacijo.

Nato vnesemo na isti način podatke o našem osebnem avtomobilu, kot pri izbiri načina TRACK, če pa ga ni v naši podatkovni bazi, nam omogoči mobilna aplikacija ročni vnos podatkov o porabi. Tukaj gre samo za izris poti od naše trenutne lokacije do želene lokacije. V prvem koraku mobilna aplikacija pridobi podatke o naši lokaciji.



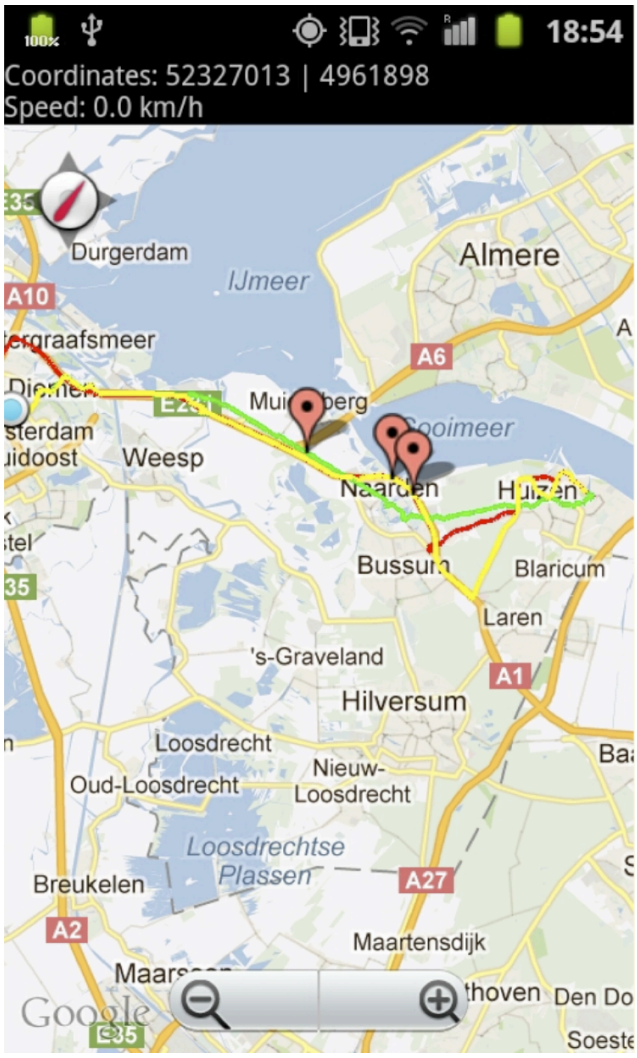
Slika 19: Primer vnosa želene lokacije

Naslednji dialog, ki je prikazan na prejšnji strani, nam s pomočjo internetne povezave (WIFI, GPRS) in samodokončanja (angleško Autocomplete) ponudi našo želeno lokacijo, do kam želimo izrisati poti. Ta dialog je sestavljen iz dveh gradnikov, iz spreminjanja besedila (angleško Edit Text) in pogleda seznama (angleško List View).

Ko izberemo želeno lokacijo, moramo počakati nekaj časa, da se izrišejo naše tri vrste poti do želene lokacije.

Spodnja slika nam prikazuje pot od naše lokacije, to je moder krogec, do želene lokacije. Google Transit nam s pomočjo Canvasa izriše vse tri vrste poti. Pri eni smo kot prevozno sredstvo uporabili avtobus, pri drugi osebni avtomobil, pri tretji pa način premikanja peš.

Kot lahko vidimo na sliki, je vsaka vrsta poti različne barve, saj so pri vsakem prevoznem

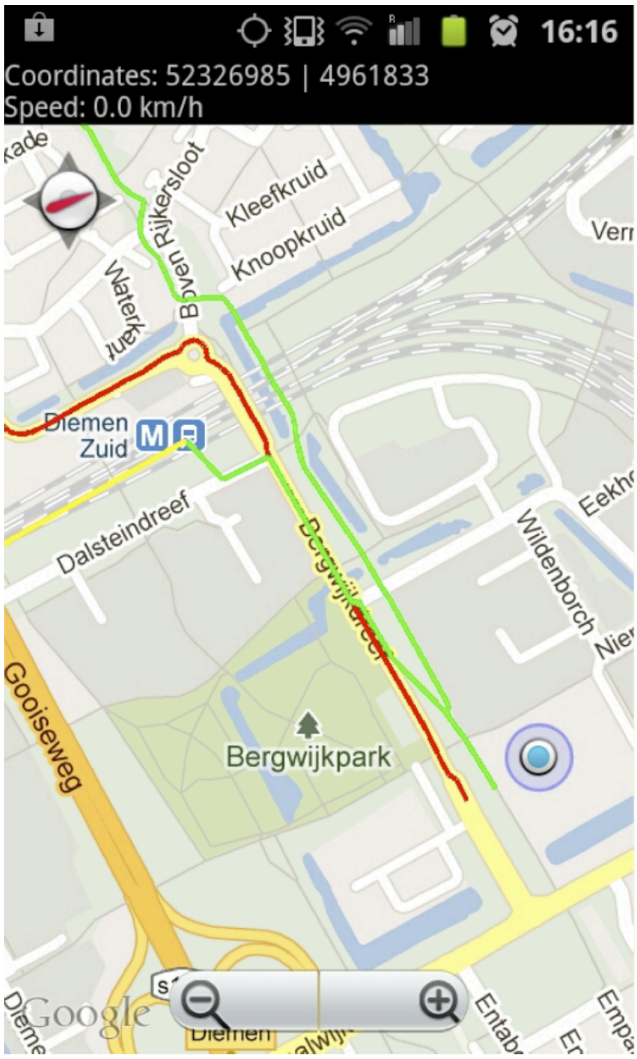


Slika 20: Primer izrisa poti v vseh treh načinih: Walk, Car in Bus

sredstvu možne različne poti, ki jih uberemo. Npr., določeni avtobusi lahko vozijo samo po določeni cesti, po kateri ni dovoljena vožnja z osebnim avtomobilom. Na naslednji sliki lahko

vidimo podrobnejši pogled zemljevida, kjer lahko vidimo podani primer, ko je pot od avtobusa drugačna od poti osebnega avtomobila.

Po končanem izrisovanju poti imamo možnost izbire, da končamo z ogledom izrisanih poti. Če izberemo to funkcijo, se nam prikaže enak dialog o primerjavi poti kot pri načinu TRACK, vendar z drugačnimi podatki.



Slika 21: Podrobnejši pogled poti v vseh treh načinih: Walk, Car in Bus

Naša druga izbira načina delovanja aplikacije DIRECTIONS ni glavni del, ampak smo se jo domislili in razvili sproti, potem ko smo končali razvoj izbire načina TRACK. Smiselno se nam je zdelo imeti tudi način, ki ne zahteva premikanja uporabnika aplikacije, saj si uporabniki včasih želijo vnaprej informativno izračunati, kako močno bodo onesnažili okolje, koliko časa bo trajala njihova vožnja in kakšna je razdalja med dvema lokacijama na zemljevidu.

4.3 Datoteka Android Manifest

Vsaka Android aplikacija mora imeti XML datoteko AdnroidManifest.xml v svojem korenskem direktorju. Ta predstavlja bistvene informacije o aplikaciji za uporabo operacijskega sistema Android. Te informacije mora imeti sistem, preden zažene katerokoli aplikacijsko kodo. Med drugim Android Manifest naredi naslednje:

- poimenuje Java paket za aplikacijo. To ime služi kot edinstveni identifikator za uporabo,
- določa, kateri procesi bodo gostili komponente aplikacije,
- določa, katera dovoljenja mora imeti aplikacija, da lahko dostopa do zaščitenih delov API-ja, in interakcijo z drugimi aplikacijami,
- določa, katera dovoljenja morajo imeti druge aplikacije, da lahko vplivajo na komponente aplikacije,
- navaja knjižnice, s katerimi mora biti aplikacija povezana,
- navaja minimalno raven Androidovega API-ja, ki ga aplikacija potrebuje,
- opisuje komponente aplikacij, prav tako tudi poimenuje razrede, ki implementirajo vsako komponento,
- navaja Instrumentation razrede, ki zagotavljajo profiliranje in druge informacije, ko aplikacija teče.

4.3.1 Naša Android Manifest datoteka

```
<uses-sdk android:minSdkVersion="7" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE" >
</uses-permission>
```

Kot lahko vidimo, smo naši aplikaciji nastavili različne pravice, ki ji omogočajo naslednje:

- dovoljenje aplikaciji za odpiranje omrežnih vtičnikov,
- dovoljenje aplikaciji za dostop do finih (npr. GPS) lokacij,
- dovoljenje aplikaciji za dostop do grobih (npr. WIFI) lokacij,
- dovoljenje aplikaciji za dostop do informacij o omrežjih,
- dovoljenje aplikaciji za spremembo stanja omrežne povezave,
- dovoljenje aplikaciji bralnega dostopa do stanja telefona.

```
<application
       android:persistent="true"
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:debuggable="true"
>
       <uses-library android:name="com.google.android.maps" />
```

```
        <activity
                android:name=".CarbonTrackerActivity"
                android:label="@string/app_name"
                android:screenOrientation="portrait">
                <intent-filter>
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
        </activity>
        …
</application>
```

Naša aplikacija vsebuje tudi različne aktivnosti, ki morajo biti ugnezdene znotraj značke <application>, saj če niso, jih sistem ne vidi in posledično niso zagnane. Aktivnosti nam predstavljajo dele vizualnega uporabniškega vmesnika aplikacije.

4.4 Postavitev (angleško Layout)

Kot pomemben del naše aplikacije, bi še lahko izpostavili postavitev, ki je arhitektura za uporabniški vmesnik v aktivnosti. Postavitev definira strukturo postavitve in ima elemente, ki se pojavijo neposredno uporabniku. Postavitev lahko definiramo na dva načina: deklariranje elementov uporabniških vmesnikov znotraj XML-ja in pa sprožanje postavitve elementov med časom delovanja.

V našem primeru smo uporabili prvo izbiro, da smo deklarirali elemente uporabniških vmesnikov znotraj XML datoteke. Prednosti te izbire so, da nam omogoča nazorno ločitev grafičnega prikaza naše aplikacije od obnašanja programske kode. Opisi teh uporabniških vmesnikov so v zunanji aplikacijski kodi, kar pomeni, da jih lahko modificiramo ali prilagodimo brez tega, da bi nam bilo treba modificirati izvorno kodo in jo ponovno sestaviti. Npr., znotraj XML datoteke lahko naredimo postavitev za različne usmeritve zaslona, različne velikosti zaslonov naprav in različne jezike. Deklariranje postavitve v XML datoteki nam dodatno omogoča lažjo vizualno strukturo uporabniških vmesnikov, posledično pa lažje odpravljanje napak.

Primer kode naše postavitve, ko ustavimo sledenje premikanju, npr. datoteka stopped.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        <TextView android:id="@+id/text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/stoppedText" />
        <Button android:id="@+id/buttonBack"
               android:layout_width="170px"
               android:layout_height="wrap_content"
               android:layout_marginLeft="150px"
               android:layout_marginTop="100px"
               android:text="@string/back" />
</LinearLayout>
```

Po uspešnem deklariranju XML datoteke jo shranimo s končnico .xml v mapo res/layout, da bo pravilno sestavljena.

Primer kode, kako naložimo postavitev v Javi, če jo shranimo pod imenom login.xml:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.login);
}
```

Vsak pogled in objekt skupine pogledov podpirata različne lastnosti. Vidimo, da tudi naš pogled vsebuje lastnosti, kot so širina, višina, orientacija, ID, besedilo, levo mejo in zgornjo mejo.

Vsak objekt pogleda ima lahko celoštevilski ID, ki enolično določa pogled znotraj drevesa. Simbol @ na začetku niza označuje, da XML razčlenjevalnik lahko razčlenjuje in razširi preostali del ID niza in ga prepozna kot ID vir. Simbol plus (+) pomeni ime sredstva, ki mora biti narejeno in dodano v naše vire, torej v R.java datoteko.

Primer kode v datoteki R.java:

```
public static final class id {
   public static final int buttonBack=0x7f090026;
   // others …
}
```

Lastnost LinearLayout je pogled skupine, ki poravna vse otroke v eni smeri, navpično ali vodoravno, v našem primeru navpično.

4.4.1 Uporabljeni gradniki v aplikaciji

- Button (slovensko gumb) je zaslonski gradnik, ki ob dotiku uporabnika sproži akcijo.
- TextView (slovensko pogled besedila) prikazuje besedilo uporabniku in opcijsko omogoča spreminjanje tega besedila.
- ImageView (slovensko pogled slike) prikazuje poljubne podobe, kot so ikone in podobne.
- EditText (slovensko spreminjanje besedila) je samo dodatek TextViewu, ki omogoča spreminjanje besedila.
- MapView (slovensko pogled mape) je gradnik od Googla, ki nam omogoča, da naredimo svojo aktivnost za pogled mape.
- ListView (slovensko pogled seznama) nam omogoča pogled skupine, ki prikazuje seznam pomičnih predmetov.

5 Ocena izvedbe in nadaljnje delo

V diplomskem delu smo predstavili teoretični del, prav tako pa tudi praktični del naše aplikacije. V tem poglavju smo želeli predstaviti globljo resnico in izbire nadaljnjega dela za našo aplikacijo CarbonTracker.

Aplikacija se mi osebno zdi zelo dober koncept, če pa bi hoteli, da bi postala prodajni produkt, bi bilo treba vložiti v njo še nekaj dela. Zelo všeč mi je bilo to, da smo se projekta lotili pravočasno, dobro načrtali našo arhitekturo in da nam je uspela tudi implementacija. S tem smo si tudi odprli vrata v svet razvoja mobilnih aplikacij, pridobili tudi nekaj novega znanja s področja podatkovnih tehnologij, spletnih tehnologij, prav tako pa podrobneje spoznali integrirano razvojno okolje Eclipse, Androidove razvojne pakete in Googlovo tehnologijo za delo z GPS-om in mapami. V dobro razvoja aplikacije štejem tudi to, da nam je aplikacijo uspelo razviti brez denarnih sredstev, saj smo uporabili vso brezplačno programsko opremo. Posledično je njena prednost pred podobnimi aplikacijami ravno to, da je brezplačna. S tem smo želeli doseči, da bi čim več uporabnikov uporabljalo našo aplikacijo, prav tako pa bi bila večja verjetnost, da bi se izmed njih našel nekdo, ki bi nadaljeval naše delo in jo poizkušal izboljšati.

Kljub temu, da aplikacija deluje, bi jo bilo možno izboljšati z dodatnimi funkcionalnostmi. Poskusni uporabniki so nas opozorili na možne izboljšave naše aplikacije.

Prva taka je kriptiranje podatkov pri sinhronizaciji podatkov lokalne podatkovne baze XML na napravi aplikacije s podatkovno bazo MySQL na strežniku. Če podatkov ne kriptiramo, se lahko zgodi, da se najde srečni najditelj, ki se seznani z njimi in jih v najslabšem primeru uporabi sebi v korist in nam v škodo. Zato bi bilo treba v naši aplikaciji uporabiti kriptografski protokol TLS ali vsaj njegovega predhodnika SSL. To sta kriptografska protokola, ki zagotavljata varno komunikacijo v internetu. Ta dva uporabljata asimetrično kriptografijo za izmenjavo ključev, simetrično kriptografijo za zaupanje in pristne kode za integriteto sporočila.

Možno bi bilo še izboljšati hitrost razčlenjevalnika XML baze na napravi aplikacije. To smo opazili med testiranjem aplikacije. Do neke mere nam je razčlenjevalnik uspelo pospešiti, vendar verjamem, da bi ga bilo mogoče še bolj pospešiti ter optimizirati.

Eden od takih delov mobilne aplikacije, ki še ni tako dobro in dovršeno izpopolnjen, je uporabniški vmesnik. Temu smo sicer namenili nekaj časa, oblikovali grafične elemente v čim boljšem stilu z oblikovalskim orodjem Photoshop, vendar bi bilo po mojem mnenju grafične elemente mogoče še boljše oblikovati. S tem bi dosegli še večjo prijaznost uporabniku, posledično pa po vsej verjetnosti pogostejšo uporabo aplikacije.

Našo aplikacijo bi lahko izboljšali s področja podatkovne analize in drugega pomembnega področja prilagoditve uporabniku.

Na področju podatkovne analize bi se lahko usmerili na področje, ki se ukvarja z obdelavo in analiziranjem podatkov. To je poslovna inteligenca.

Ena od takih izboljšav, ki bi nam pomagala pri analiziranju podobnosti med uporabniki, je hierarhično razvrščanje (angleško Hierarchical clustering). Ta nam omogoča, da uporabnike s podobnimi lastnostmi razvrstimo v isto skupino. Prav tako pa lahko vidimo tudi podobnosti

med različnimi skupinami. Za določanje podobnosti med uporabniki s hierarhičnim razvrščanjem pa lahko uporabimo različne razdalje: Evklidsko razdaljo, Manhattansko razdaljo, Maksimalno razdaljo, Kosinusno in še druge. Vsaka od teh razdalj je uporabna v točno določenem specifičnem primeru.

Prav tako bi za iskanje podobnosti med uporabniki lahko uporabili podobno zadevo kot hierarhično razvrščanje, imenovano večrazsežno normiranje (angleško multi dimensional scaling - MDS). To nam omogoča predstavitev podobnosti med uporabniki na večrazsežnem grafu.

S tema dvema iskanjema podobnosti bi torej lahko v iste skupine razvrstili uporabnike, ki spustijo v ozračje približno enako količino ogljikovega dioksida, tiste, ki uporabljajo enaka prevozna sredstva, in druge uporabnike, ki so si podobni po drugih lastnostih.

Na področju prilagoditve uporabniku pa bi lahko dodali dodatne funkcionalnosti na mapo. Ena takih koristnih bi bila za naše uporabnike, ki uporabljajo mestni promet, da bi naša aplikacija na mapi, kjer so izrisane poti premikanja ali le poti, vsebovala pokritost postaj mestnega prometa. S tem bi namreč uporabniku omogočili, da bi takoj videl, kje je najbližja avtobusna postaja.

Druga taka izboljšava na mapi bi bila, da bi nam aplikacija na zaslonu izpisala prihod avtobusa na najbližji postaji.

Menim, da bi vse zgoraj omenjene izboljšave bilo mogoče implementirati. Nobena se mi ne zdi nemogoča, vse kar je potrebno narediti, je vložiti nekaj dela.

6 Sklepne ugotovitve

V okviru diplomskega dela smo razvili aplikacijo z naslovom CarbonTracker, ki vključuje mobilno aplikacijo za mobilne naprave z operacijski sistemom Android in strežniški del aplikacije, ki omogoča shranjevanje pridobljenih podatkov iz mobilne naprave, ter prikazovanje teh podatkov na spletni strani. V diplomskem delu pa smo prišli tudi do nekaterih ugotovitev, ki jih bomo predstavili v naslednjih odstavkih.

Naš namen je bil opomniti uporabnike naše aplikacije o zavedanju onesnaževanja okolja, podrobneje ozračja in kako lahko prispevajo k zmanjšanju onesnaževanja v primeru izbire ustreznega prevoznega sredstva. Ugotovili smo, da je to mogoče doseči s pomočjo informacijskih tehnologij, podrobneje z mobilno aplikacijo, ki meri, kako močno onesnažujemo okolje oziroma ozračje. To smo ugotovili tako, da smo aplikacijo praktično razvili, jo ponudili zainteresiranim uporabnikom in jih po nekaj časa uporabe povprašali o zadovoljstvu. Rezultat je bil v večino primerih pozitiven, dobili pa smo tudi nekaj nasvetov za izboljšavo aplikacije, ki jih bomo poizkušali implementirati v bližnji prihodnosti.

Največ težav nam je povzročalo usposobitev tehnologije GPS in shranjevanje podatkov v podatkovno bazo XML na napravi mobilne aplikacije. Ravno ta del razvijanja se mi je zdel najbolj zanimiv, saj je ravno ta eden izmed glavnih delov aplikacije. Težav pri usposabljanju tehnologije Google Maps za izrisovanje poti na zemljevidu nismo imeli, pojavile pa so se pri prej omenjenem shranjevanju pridobljenih podatkov in usposobitvi tehnologije GPS. Rešiti nam jih je uspelo po krajšem raziskovanju tehnologij.

Med samim razvojem mobilnega dela naše aplikacije smo tudi ugotovili, da je razvoj mobilnih aplikacij za operacijski sistem Android dobro podprt. Zelo dobro je pripravljena dokumentacija, ki jo lahko najdemo na spletni strani http://developer.android.com. Ravno ta podprtost omogoča širši množici programerjev razvoj mobilnih aplikacij za operacijski sistem Android, kar pa zaradi privlačnosti, zanimivosti in uporabnosti teh aplikacij vpliva na pogostejšo uporabo pametnih telefonov z operacijskim sistemom Android.

Če povzamemo naše ugotovitve, lahko ugotovimo, da smo se naučili dosti novega, kar nam bo še prišlo prav v bližnji prihodnosti, ko se bomo srečali s podobnimi težavami, kot smo se pri razvoju naše aplikacije.

Literatura

[1] (2012) <activity> | Android Developers. Dostopno na: http://developer.android.com/guide/topics/manifest/activity-element.html

[2] (2012) Layouts | Android Developers. Dostopno na: http://developer.android.com/guide/topics/ui/declaring-layout.html

[3] (2012) Model-View-Controller. Dostopno na: http://msdn.microsoft.com/en-us/library/ff649643.aspx

[4] (2012) Java Platform SE 6. Dostopno na: http://docs.oracle.com/javase/6/docs/api/

[5] (2012) Install Android SDK, Eclipse, and Emulator (AVDs) | Developer How-To | Android | Konreu. Dostopno na: http://android.konreu.com/developer-how-to/install-android-sdk-eclipse-and-emulator-avds/

[6] (2012) Android, the world's most popular mobile platform | Android Developers. Dostopno na: http://developer.android.com/about/index.html

[7] (2012) XPath Syntax. Dostopno na: http://www.w3schools.com/xpath/xpath_syntax.asp

[8] Matt Zandrsta: Sams Teach Yourself PHP in 24 Hours, Third Edition, Sams, December 2003

[9] Robert W. Sebesta: Programming the World Wide Web, Sixth Edition, Addison Wesley, Marec 2010

Kazalo slik