Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Erol Merdanović

Mobilna aplikacija za Erasmus študente na Univerzi v Ljubljani

diplomsko delo
univerzitetni študijski program

Mentor: doc. dr. Rok Rupnik

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: 01941 / 2013
Datum: 4.9.2013

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

Kandidat: **EROL MERDANOVIĆ**

Naslov: **MOBILNA APLIKACIJA ZA ERASMUS ŠTUDENTE NA UNIVERZI V LJUBLJANI**
MOBILE APPLICATION FOR ERASMUS STUDENTS AT UNIVERSITY OF LJUBLJANA

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Tuji študentje Erasmus programa pridejo začasno študirat na eno od fakultet Univerze v Ljubljani in potrebujejo dostop do različnih informacij: informacije vezane na univerzo in informacije vezane na mesto Ljubljana. Razvijte mobilno aplikacijo, ki bo Erasmus študentom na Univerzi v Ljubljani omogočila dostop do najpomembnejših podatkov naslednjih entitet: fakultete univerze, večje knjižnice, tuja veleposlaništva ter študentski domovi. Mobilno aplikacijo razvijte z uporabo ogrodja PhoneGap, kar bo omogočilo uporabo aplikacije na platformah Android, iOS in Windows Phone. Razvijte tudi mehanizem za osveževanje podatkov v lokalni podatkovni bazi aplikacije.

Mentor: Dekan:

doc. dr. Rok Rupnik prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Erol Merdanović, z vpisno številko **63070206**, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za Erasmus študente na Univerzi v Ljubljani

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- 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 4. decembra 2013 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Roku Rupniku za pomoč in nasvete v času nastajanja diplomskega dela.

Zahvaljujem se tudi družini in punci za potrpežljivost in podporo v času študija.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Preučitev zahtev in scenarijev uporabe **3**

2.1 Vsebinski del 4

2.2 Tehnični del 5

3 Pregled obstoječih rešitev **7**

3.1 Zaledni del 7

3.1.1 Domorodna aplikacija 8

3.1.2 Hibridna aplikacija 10

3.2 Čelni del 13

3.2.1 jQuery Mobile 13

3.2.2 Zepto 14

3.2.3 Sencha Touch 15

3.3 Mape 15

3.3.1 Google Maps 15

3.3.2 Bing Maps 16

3.3.3 OpenStreetMap in Leaflet 17

3.4 Shranjevanje podatkov 18

3.4.1 Web Storage 18

3.4.2 Web SQL Database 19
3.4.3 Indexed Database 19
3.4.4 SQLite 20
3.5 Ugotovitve 21

4 Uporabljene tehnologije **23**
4.1 Razvojno orodje 23
4.2 HTML5 in CSS3 23
4.3 JavaScript 24
4.4 PhoneGap 24
4.5 SQLite 24
4.6 jQuery in jQuery Mobile 25
4.7 JSON 25
4.8 OpenStreetMap 26
4.9 Leaflet 26

5 Spletna administracija **27**
5.1 Preučitev virov in formatov podatkov 27
5.2 Razvoj administracije 28
5.3 Razvoj komunikacijskega vmesnika 29
5.3.1 Dostop do podatkov 30

6 Razvoj mobilne aplikacije **33**
6.1 Priprava razvojnega okolja 33
6.2 Razvoj mobilne aplikacije 35
6.2.1 Osnovna struktura 35
6.2.2 Poslovna logika 36
6.2.3 Prezentacijska logika 47
6.3 Testiranje mobilne aplikacije 53

7 Sklepne ugotovitve **55**

Povzetek

Prišli smo v obdobje, ko mobilni telefoni postajajo del našega življenja. Nove tehnologije nam omogočajo reševanje problemov, ki so nam bili pred nešteto leti nerešljivi. Dandanes imamo ogromno število mobilnih aplikacij. Nekatere so dobre, spet druge slabe. Razlika med njimi je, da dobre aplikacije uporabljajo zadnje tehnologije in pristope ter najbolje rešujejo specifičen problem. Vprašanje je: katere tehnologije in pristope je potrebno uporabiti za reševanje specifičnega problema?

Naš problem oz. cilj diplomske naloge je razvoj mobilne aplikacije za Erasmus študente, ki so na izmenjavi v Ljubljani. Naloga le-te je omogočiti dostop do osnovnih informacij, ki jih študentje potrebujejo tekom študija. To obsega informacije kot npr. lokacija in predstavitev Univerze v Ljubljani, lokacije in osnovni podatki vseh fakultet, knjižnic, veleposlaništev, študentskih domov in drugih javnih ustanov ter uporabni nasveti za lažje bivanje v Ljubljani. Razvoj vključuje določitev specifikacije mobilne aplikacije, preučitev rešitev za omejitve, ki se pojavljajo pri različnih scenarij uporabe ter na koncu tudi sama implementacija na realnih podatkih. Z uporabo ogrodja *PhoneGap* smo pokrili platforme *Android*, *iOS* in *Windows Phone*. Za uporabniški vmesnik smo uporabili *HTML5* in *jQuery Mobile*. Podatke smo shranili v lokalno SQLite podatkovno bazo ter razvili mehanizem za osveževanje podatkov. Za vizualizacijo lokacij na mapi nam služijo odprtokodne *OpenStreetMap* mape skupaj s knjižnico *Leaflet*, ki poskrbi za prikazovanje in urejanje map na mobilni napravi.

Ključne besede: PhoneGap, jQuery Mobile, SQLite, OpenStreetMap

Abstract

We have arrived to a period when mobile phones are becoming part of our lives. New technologies enable us to solve problems, which seemed unsolvable few years ago. Nowadays we have a lot of mobile applications. Some are good, some are bad. Difference between them is that good applications use latest technologies and approaches and best solve specific problem. Question is: which technologies and approaches have to be used to solve specific problem?

Our problem or goal of our dissertation is to develop mobile application for Erasmus students on exchange in Ljubljana. Task of the application is to enable access to basic information needed while studying. Information consists of location and presentation of University of Ljubljana, locations and basic information about faculties, libraries, embassies, student dormitories and other public institutions, and useful tips for easier living in Ljubljana. Development includes defining mobile application specification, examination of solutions for limitations at different scenarios of usage and on the end also implementation on actual data. With framework *PhoneGap* we covered *Android*, *iOS* and *Windows Phone* platforms. For user interface we used *HTML5* and *jQuery Mobile*. We stored data in local SQLite database and developed mechanism for updating data. For visualisation of locations on map we used opensource *OpenStreetMap* maps together with *Leaflet* library, which takes care of displaying and enhancing map on mobile device.

Keywords: PhoneGap, jQuery Mobile, SQLite, OpenStreetMap

Poglavje 1

Uvod

Z hitro rastočim trgom mobilnih naprav posledično raste tudi trg mobilnih aplikacij. Kljub temu, da se je v zadnjih letih razvilo ogromno aplikacij, je še vedno veliko prostora za nove. Za razvoj je smiselno uporabiti najboljše rešitve, ki trenutno obstajajo na tržišču. Pojem najboljše je lahko varljiv, saj moramo upoštevati vse zahteve aplikacije. Vendar če se držimo teh pravil, bo končni rezultat mobilna aplikacija, ki služi svojemu namenu.

Namen naše aplikacije je ponuditi študentom na Erasmus izmenjavi v Ljubljani enostaven in hiter dostop do relevantnih informacij med bivanjem v Ljubljani. Namen diplomske naloge je preučiti katere vse informacije je potrebno vključiti v aplikacijo, kako jih shraniti in prikazati ter na koncu aplikacijo tudi razviti. Diplomsko delo opisuje vse korake, ki smo jih izvedli.

Prvo smo preučili vse zahteve in scenarije uporabe mobilne aplikacije, kar je opisano v drugem poglavju. Cilj je bil ugotovitev, katere vse osnovne funkcionalnosti mora aplikacija vsebovati. Bolj podrobno, katere vse probleme lahko aplikacija reši oz. katere informacije mora vsebovati. Pri tem smo se osredotočili tako na vsebinski kot tehnični del. Rezultat je bil seznam vseh vsebinskih zahtev ter seznam vseh tehničnih omejitev.

V naslednjem oz. tretjem poglavju predstavimo pristope k izpolnjevanju zahtev. Probleme pri razvoju mobilne aplikacije smo razdelili v 4 dele: zaledni (*backend*), čelni (*frontend*), mape in shranjevanje podatkov. Za vsak del

smo pregledali par možnih rešitev. Za vsako rešitev smo preučili, ali ustreza našim željam in zahtevam in na kratko opisali pluse in minuse uporabe. Na koncu smo strnili naše ugotovitve in katere rešitve smo uporabili.

V četrtem poglavju so predstavljene rešitve oz. tehnologije, ki smo jih uporabili. Pomembno je, da smo kot prvo seznanjeni z zmožnostmi in omejitvami vsake tehnologije. Kot drugo, znati jo moramo ustrezno uporabiti pri razvoju.

V petem poglavju je predstavljen razvoj spletne administracije, ki služi za agregacijo podatkov. Omogoča dodajanje in urejanje podatkov kot npr. dodajanje nove lokacije ali urejajanje obstoječe. Hkrati smo razvili vmesnik, preko katerega mobilna aplikacija črpa podatke.

V šestem poglavju je opisan naš razvoj mobilne aplikacije. Prvo smo pripravili vso potrebno orodje ter knjižnice, ki so potrebne za posamezno platformo. Razvoj smo razdelili v več delov. Strmeli smo k pregledni strukturi mobilne aplikacije ter jasno določiti namen vsakega gradnika. Po uspešno končanem razvoju smo aplikacijo tudi ročno testirali na različnih platformah in mobilnih napravah ter odpravili najdene napake.

Uspešnost diplomskega dela merimo v dveh sklopih. Prvi sklop je izdelana mobilna aplikacija, ki vsebuje vse osnovne informacije glede na zahteve, ki smo jih določili. Drugi sklop pa predstavlja odziv in sprejetje med študenti. Ta sklop je malo težje merljiv, ker je potreben čas. Hkrati je odvisen od veliko dejavnikov kot npr. oglaševanje oz. promoviranje same aplikacije med študenti ali domneva, da študentje raje uporabljajo druge vire informacij (študentske pisarne, študentske organizacije, spletni forumi, ...).

Poglavje 2

Preučitev zahtev in scenarijev uporabe

Koraki, ki smo jih izvedli pri določanju specifikacije aplikacije, smo razdelili v 2 sklopa: vsebinski in tehnični.

Vsebinski del je zavzemal pregled *Facebook* skupin ter forumov na temo Erasmus izmenjave v Ljubljani. Fokusirali smo se na najbolj pogosta vprašanja oz. težave, ki se pojavljajo in so rešljive z mobilno aplikacijo. Postavili smo tudi par vprašanj, s katerimi smo ugotovili kaj si študentje želijo oz. potrebujejo v mobilni aplikaciji. Hkrati smo pregledali podobne aplikacije na spletu, vendar na žalost jih ni bilo veliko oz. niso bile dobro zasnovane ali implementirane.

Pri tehničnem delu smo pregledali katere vse operacijske sisteme moramo pokriti, da zadostimo čim večjemu številu študentov. Pri tem smo preučili, kako realizirati vsebinske zahteve in scenarije ter jih preslikati v mobilno aplikacijo z uporabo obstoječih rešitev.

Z vsemi informacijami smo prišli do naslednjih zaključkov.

2.1 Vsebinski del

Mobilna aplikacija ponuja študentom hiter in enostaven dostop do naslednjih informacij:

Fakultete in akademije Univerze v Ljubljani

Vse fakultete in akademije Univerze v Ljubljani smo predstavili z njihovo lokacijo, delovnim časom študentskega referata, telefonsko številko in elektronskim naslov ter povezavo do spletne strani za potrebe pridobitve dodatnih informacij.

Veleposlaništva

Mobilna aplikacija vsebuje vsa veleposlaništva držav iz katerih prihajajo študentje Erasmus izmenjave v Ljubljano za potrebe opravljanja državnih dolžnosti kot npr. volitve, udeležba v predstavitvah držav ali kaj podobnega. Informacije vključujejo lokacijo, delovni čas, telefonsko številko in spletni naslov vseh veleposlaništev.

Knjižnice

Za razjasnitev ali dograditev znanja morajo študentje poseči po dodatnem učnem gradivu, zato je važno, da imajo dostop do informacij kje se nahajajo knjižnice, kdaj so odprte ter do kontaktov v primeru vprašanj. Vse to mobilna aplikacija vsebuje.

Študentski domovi

Ker vsakemu Erasmus študentu pripada bivanje v študentskih domovih, aplikacija vsebuje tudi vse lokacije študentskih domov, ki sprejemajo študente na izmenjavi. Poleg osnovne lokacije doma, lahko študent pridobi informacije o številu postelj, telefonsko številko, elektronski naslov in naslov spletne strani (če le-ta obstaja).

Ostale javne ustanove

Poleg prej omenjenih ustanov, mobilna aplikacija vsebuje informacije tudi o ostalih nujno potrebnih ustanovah kot so Ljubljanski potniški

promet, Davčni urad, Urad za tujce, Ljubljanski klinični center itd. Poleg osnovnih informacij o delovnem času, telefonski številki, elektronskem naslovu in spletni strani, je predstavljen tudi namen vsake ustanove npr. ureditev stalnega prebivališča, ureditev Urbane za uporabo Ljubljanskega potniškega prometa itd.

Predstavitev Ljubljane

Glede na manjšo prepoznavnost Ljubljane oz. Slovenije v tujini, se študentje na Erasmus izmenjavi sprašujejo o glavnem mestu Slovenije. Mobilna aplikacija jim ponuja nekaj osnovnih informacij o zgodovini Ljubljane, velikosti ter viziji za prihodnost. Pri tem je vključeno tudi več povezav do spletnih strani z dodatnimi informacijami.

Predstavitev univerze

Univerza v Ljubljani je ena izmed 500 najboljih univerz na svetu [7]. To je eden izmed glavnih razlogov zakaj si zasluži, da je predstavljena v mobilni aplikaciji. Študentje pridobijo osnovne informacije kot so opis in vizijo, kontaktne informacije ter dostop do informativne spletne strani.

Uporabni nasveti

Vsakdo, ki pride v novo okolje, želi izvedeti uporabne nasvete domačinov. Ravno zato smo v mobilno aplikacijo dodali tudi uporabne nasvete kot npr. kje kupiti rabljeno kolo, kje naročati hrano preko spleta, kje poiskati prenočišče do ureditve začasnega prebivališča itd.

Vsebinski del bi lahko vključeval še veliko ostalih informacij, vendar smo prišli do zaključka, da so nekatere informacije na voljo v drugih aplikacijah ali da je pogostost potrebe po specifičnih informacijah zanemarljiva.

2.2 Tehnični del

Glede na statistiko [8], smo pokrili 3 glavne operacijske sisteme: *Android, iOS* in *Windows Phone*. Tako smo dosegli, da pokrijemo večino študentov.

Zavedati se moramo, da ne moremo pokriti vseh platform. Razlog je, da so nekatere slabo razširjene ali so v zatonu. Nekatere niso tehnološko dovršene in ne omogočajo izvedbo vsebinskih scenarij oz. zahtev.

Zaradi omejitve prenosa podatkov ali nezmožnosti povezave na splet, smo podatke shranjevati lokalno v aplikacijo. Podatki obsegajo celotni vsebinski del. Zaradi velike količine podatkov in potrebe po poizvedovanju nad njimi, smo uporabili podatkovno bazo. Po potrebi nove ali posodobljene podatke prenesemo s strežnika. V naslednjem koraku jih shranimo v lokalno podatkovno bazo. Zato smo razvili mehanizem, ki preveri, če so na voljo novi podatki, jih prenese in lokalno shrani.

V mobilni aplikaciji velik delež podatkov predstavljajo lokacije. Potrebno jih je ustrezno vizualizirati. Najboljši način je prikaz na mapi. Problem, ki je bil omenjen v prejšnjem odstavku je, da imamo omejen dostop do spleta. Zato smo uporabili tehnologijo, ki omogoča lokalno shranjevanje map ter prikaz lokacij. S tem smo dosegli, da uporabniku hitro in nemoteno predstavimo lokacije brez potrebe, da vzpostavlja internetno povezavo.

Končne tehnične zahteve obsegajo:

- podpora platform *Android*, *iOS* in *Windows Phone*;
- prikaz lokacij na lokalni mapi;
- shranjevanje podatkov v lokalno podatkovno bazo, poizvedovanje nad njimi in po potrebi posodobitev le-teh.

Poglavje 3

Pregled obstoječih rešitev

Da smo zadovoljili vse potrebe in zahteve, ki smo jih definirali v prejšnjem poglavju, smo aplikacijo razdeliti v več plasti. Pri vsaki plasti smo pregledali najbolj popularne oz. najbolj ustrezne rešitve ter preučili njihove prednosti in slabosti. Pri vsaki plasti smo poleg omejitev upoštevali tudi možnost komunikacije med plasti oz. njihova kompatibilnost. Ker smo bili časovno in finančno omejeni, sta bila čas in strošek razvoja bistvenega pomena. Med zahtevami ni zahteve za odprtokodnost. Kljub temu smo se odločili, da poskušamo uporabiti tehnologije in rešitve, ki so odprtokodne.

3.1 Zaledni del

Za razvoj zalednega dela aplikacije smo izbirali med tipi aplikacij: domorodne, hibridne in spletne aplikacije. Vsak tip ima prednosti in slabosti, ki smo jih predstavili v tabeli. Naloga zalednega dela je, da zagotovi dostop do zahtevanih komponent mobilne naprave in izvaja čelno kodo. To si lahko predstavljamo kot shranjevanje in branje podatkov, uporaba kamere, pošiljanje elektronske pošte itd.

Poleg tega mora biti delovanje aplikacije hitro oz. dovolj hitro, da je uporabniška izkušnja zadovoljiva. Glede na to, da naše zahteve niso vključevali veliko procesorske moči, so vsi tipi aplikacij ustrezni.

	Dostop do komponent naprave	Hitrost	Potrebna internetna povezava	Stroški razvoja in vzdrževanja
Domorodna aplikacija	Polni	Hitro delovanje	Ne	Veliki
Hibridna aplikacija	Omejen	Hitro do srednje hitro delovanje	Ne	Majhni
Spletna aplikacija	Zelo omejen	Počasno delovanje	Da	Majhni

Tabela 3.1: Prednosti in slabosti različnih tipov aplikacij

Potrebno je omeniti, da smo primerjali razvoj iste aplikacije oz. aplikacije z istimi funkcionalnostmi. V nekaterih robnih primerih so lahko stroški razvoja in vzdrževanja ter hitrost delovanja drugačni.

Zaradi omejitve, da mora biti aplikacija dostopna in uporabna tudi brez internetne povezave, smo se fokusirali samo na razvoj domorodnih in hibridnih aplikacij.

3.1.1 Domorodna aplikacija

Domorodna aplikacija je mobilna aplikacija, ki je sprogramirana v specifičnem programskem jeziku, kor npr *Java* [11] za *Android*, *Objective C* [12] za *iOS*, *C#* [5] za *Windows Phone* itd. Domorodne aplikacije ponujajo veliko zmogljivosti in večjo hitrost v izvajanju kode, hkrati pa zagotavljajo veliko raven zanesljivosti. Pri razvoju uporabljamo orodja in tehnologije, ki so primarno namenjene za razvoj aplikacij za izbrano platformo.

Domorodne mobilne aplikacije imajo dostop do različnih komponent mobilne naprave kot npr. kamera, kontakti, shranjevanje datotek itd. Tako uporabimo vse zmogljivosti mobilne naprave. V primerjavi z aplikacijami, ki se nahajajo na spletu, lahko uporabniki domorodne aplikacije uporabljajo brez internetne povezave.

Zaradi večje stabilnosti in hitrosti izvajanja, se domorodne aplikacije oz. njihov razvoj večinoma uporablja pri razvoju mobilnih iger. Ravno pri igrah je pomembna uporabniška izkušnja, saj ne želimo, da se igra izvaja počasno

ter nenehno sesuva oz. zapira.

Prednosti:
Aplikacija razvita kot domorodna aplikacija ponuja najboljše in najhitrejše delovanje, ker se izvaja v primarnem programskem jeziku platforme. Izrablja vse komponente, kor npr. večnitnost in izvajanje na grafičnem čipu mobilne naprave. Oboje nam ni na voljo ali pa je v okrnjeni obliki pri hibridnih ali spletnih aplikacijah. Vse platforme ponujajo knjižnice (*Android SDK, Windows Phone SDK, iOS SDK*, ...), ki jih avtorji sami posodabljajo ter skrbijo za njihovo stabilnost in varnost.

Z razvojem domorodne aplikacije uporabimo vse gradnike platforme kot npr. gumbi, seznami, slike itd. Posledično je razvoj hitrejši za specifično platformo. Z uporabo gradnikov, ki so znani uporabnikom, se zmanjša nevarnost slabe uporabniške izkušnje. Domorodne aplikacije omogočajo uporabo vseh komponent platforme in mobilne naprave. Vsekakor vse, kar je v meji dovoljenja s strani avtorja platforme.

Slabosti:
Ker je potrebno razviti aplikacijo za vsako platformo posebej, je čas razvoja daljši. Zaradi manjše razširjenosti oz. uporabe programskih jezikov za specifično platformo, je tudi težje poiskati ljudi, ki imajo ustrezno znanje za razvoj mobilne aplikacije. Obenem je zaradi različnih mobilnih aplikacij potrebno zagotavljati različno vzdrževanje. Vse to poveča stroške.

Dodatna slabost je v posodabljanju mobilne aplikacije. Enkrat, ko uporabnik naloži aplikacijo, ga je potrebno obvestiti, da je na voljo nova verzija. Obstajati mora mehanizem, ki omogoča prenos nove verzije ter zamenjavo stare verzije na mobilni napravi. Problem se tudi pojavi, ko uporabnik zavestno ne posodobi verzije, kar lahko povzroči kompatibilne nevarnosti pri komunikaciji s strežnikom ali drugimi napravi. Da o samih nevarnostih v primeru slabo spisane kode ali varnostnih lukenj ne govorimo. V nasprotju, takšnih problemov pri spletnih aplikacijah ni.

Na žalost razen *Android* operacijskega sistema, platforme niso odprtokodne. To smatramo kot slabost, saj nismo imeli vpogleda v kodo operacijskega sistema, da preverimo, kako so določene funkcionalnosti implementirane.

3.1.2 Hibridna aplikacija

Hibridna mobilna aplikacija je aplikacija, ki je sestavljena iz različnih tehnologij. V večini primerov obsega 2 plasti. Zgornja plast skrbi za uporabniški vmesnik in je v večini primerov razvita s tehnologijo *HTML5, JavaScript* in *CSS3*. Hkrati komunicira s spodnjo plastjo, ki je implementira v programskem jeziku, ki je primeren za platformo. Spodnja plast omogoča dostop do večine komponent mobilne naprave.

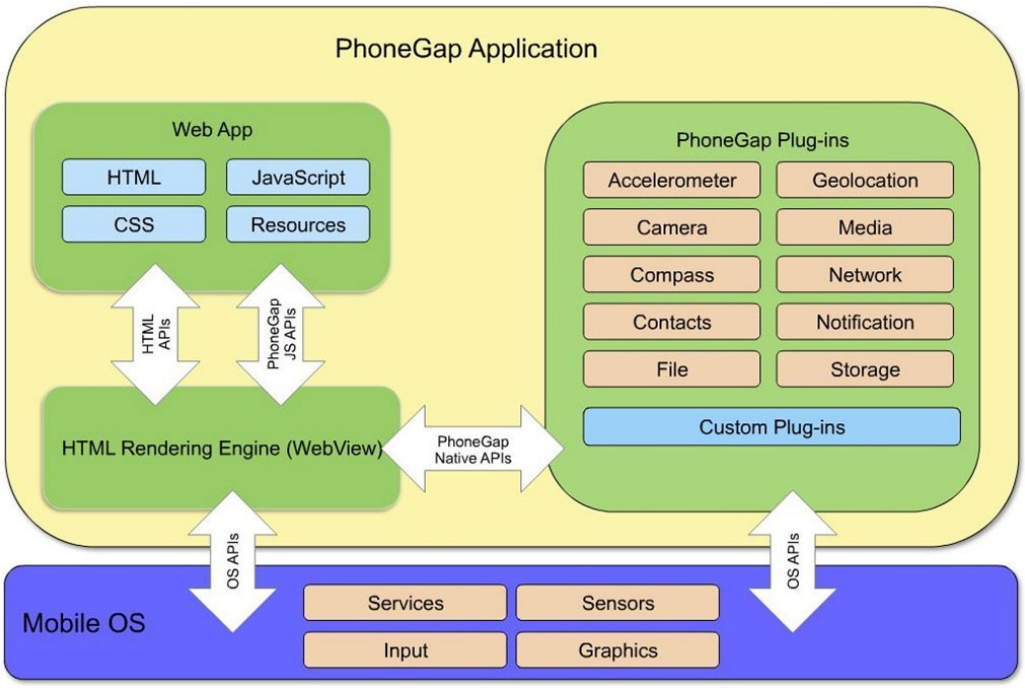
Prednosti so, da lahko eno kodo z minimalnimi spremembami uporabimo za razvoj mobilne aplikacije za več platform. Tako skrajšamo čas in stroške razvoja. Slabost je v večini primerov počasnost in omejenost dostop do komponent mobilne naprave oz. operacijskega sistema.

Preučili smo ogrodje *PhoneGap*, ki je v primerjavi z drugimi ogrodji najbolj razširjeno ter preverjeno. Hkrati je dobro sprejeto v programerski skupnosti, zato je na voljo ogromno dodatkov, ki poenostavijo in pohitrijo razvoj. Prednosti in slabosti hibridnih aplikacij smo predstavili kot prednosti in slabosti *PhoneGap* ogrodja.

PhoneGap

PhoneGap [1], [9] je odprtokodno ogrodje, ki se uporablja za razvoj mobilnih aplikacij z uporabo tehnologij *JavaScript, HTML5* in *CSS3*. *PhoneGap* je na voljo za *Android, iOS, BlackBerry, WebOS, Windows Phone, Symbian* in *Bada* [10]. Potrebno je omeniti, da pri nekaterih platformah niso podprte vse komponente mobilne naprave.

PhoneGap omogoča, da aplikacijo razvijemo s spletnimi tehnologijami. *PhoneGap* nato kodo izvaja znotraj brskalnika, ki se vzpostavi znotraj naše aplikacije. Pri tem smo neodvisni od platforme, saj PhoneGap poskrbi za



Slika 3.1: Shema strukture PhoneGap ogrodja.

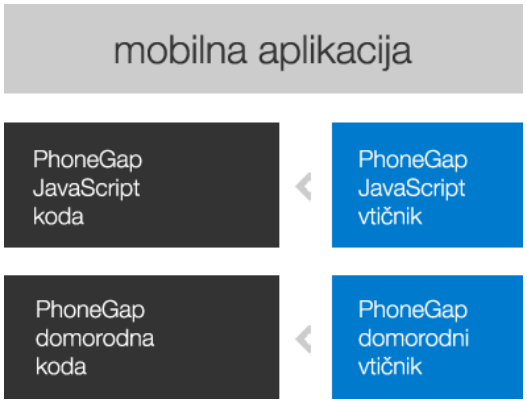
kompatibilnost. Ponuja komunikacijo s samo mobilno napravo preko vmesnika. Na voljo nam je dostop do kontaktov, uporaba kamere, shranjevanje datotek itd. Uporabnik uporablja PhoneGap aplikacijo kot vsako drugo, ker se izvorna koda zapakira v domorodno aplikacijo.

Prednosti:

Prednost *PhoneGap* je možnost uporabe iste izvorne kode na različnih platformah. S tem dosežemo hitrejši razvoj, zmanjšamo potrebe vzdrževanja in posledično tudi stroške. *PhoneGap* omogoča, da neodvisno od platforme dostopamo do osnovnih komponent mobilne naprave kot npr. kamera, kontakti, datoteke itd. Vse to naredimo s tehnologijo *JavaScript*, ki preko *PhoneGap* plasti dostopa do naprave.

Kljub temu, da je aplikacija razvita s spletnimi tehnologijami, z uporabo *PhoneGap* ni potrebe po nenehni povezavi do spleta. Vse datoteke so zapakirane znotraj aplikacije. Brskalnik, ki se vzpostavi znotraj aplikacije dostopa

do datotek ter jih interpretira kot navadno spletno stran. Kljub temu, da smo v tem primeru omejeni z zmogljivostjo brskalnika platforme, večjih sprememb oz. popravkov za delovanje na različnih brskalnikih nismo izvedli.



Slika 3.2: Prikaz vključevanje vtičnika v PhoneGap ogrodje.

PhoneGap omogoča dostop do omejenega števila komponent mobilne naprave oz. operacijskega sistema. Če želimo dodatne komponente, lahko *PhoneGap* razširimo z vtičniki. Ti so sestavljeni iz programske kode, ki se izvaja kot domorodna koda, ter iz *JavaScript* kode, ki komunicira z njo.

Slabosti:
Domorodne aplikacije so v primerjavi s *PhoneGap* aplikacijami hitrejše. Razlog je, da se domorodna programska koda hitreje izvaja v primerjavi s spletnim programskim jezikom JavaScript. Hkrati hibridne aplikacije niso primerne za večjo količino podatkov in njihovo obdelavo ali za procesorsko zahtevne naloge kot npr. za igre. *PhoneGap* zaradi omejitve tehnologije JavaScript ne omogoča večnitno izvajanja kode, kot je to možno pri domorodnih aplikacij. Vse to se pozna v prikazovanju uporabniškega vmesnika, kjer ima uporabnik občutek, da aplikacija ni dovolj odzivna.

PhoneGap ne ponuja vseh komponent platforme, zato določeni gradniki niso na voljo. To povzroči, da aplikacija v nekaterih primerih ne izpolnjuje vseh uporabnikov zahtev. Zaradi omejitve na posamezne gradnike ne moremo ponuditi uporabniku enake uporabniške izkušnje kot pri domorodni aplikaciji.

Kljub temu, da možnost razvoja vtičnikov za *PhoneGap* smatramo kot prednost, ima tudi slabost. Slabost je v tem, da moramo razviti vsak dodatek posebej za vsako platformo. To povzroči daljši čas razvoja in večje stroške.

3.2 Čelni del

Pri čelnem delu smo želeli doseči čim boljšo uporabniško izkušnjo. To pomeni, da so vsi gradniki jasno definirani in je hitrost prikazovanja in spreminjanja uporabniškega vmesnika zadovoljiva. Pri domorodnih aplikacij je čelni del vključen v knjižnice vsake platforme. Zato ni potrebno delati primerjav med njimi, ampak smo se fokusirali na hibridne aplikacije.

Izbirali smo med tremi spletnimi rešitvami, ki se razlikujejo v hitrosti, funkcionalnostih in podprtosti na različnih platformah.

3.2.1 jQuery Mobile

jQuery Mobile [13], [3] je zelo popularna knjižnica, ki se uporablja za razvoj mobilnih aplikacij ali aplikacij, ki so prilagojene za mobilne naprave. Je dodatek še bolj znane knjižnice *jQuery*. Podpira ogromno različnih platform, med njimi tudi *Android*, *Windows Phone* in *iOS*.

Prednosti:
jQuery Mobile knjižnica je zasnovana na zelo popularni knjižnici *jQuery*, zato ima ogromno dodatkov. Hkrati je preverjena rešitev na ogromno velikih projektih ter podpira več platform. Vključuje tudi orodja, ki omogočajo hiter razvoj z *jQuery Mobile* gradniki in njihovo testiranje. Zaradi prepoznavnosti je ogromno ljudi, ki poznajo tehnologijo, zato iskanje nekoga za razvoj ni težavno. V našem primeru, ko smo sami razvijali, nam je bilo na voljo ogromno dokumentacije in primerov. Prav tako smo bili v primeru težav deležni pomoči ogromne skupnosti.

Slabosti:
Slabost *jQuery Mobile* je njena hitrost. Kljub temu, da je z vsako verzijo hitrejša, ni najhitrejša med konkurenti. Razlog zakaj verjetno leži v podpori starih verzij platform oz. brskalnikov v katerih se izvaja.

Knjižnica *jQuery Mobile* ni namenjena za grajenje velikih aplikacij. Ni nobenih omejitev, vendar kmalu postane zmeda pri razvoju, saj nas *jQuery Mobile* ne prisili k uporabi strukture v aplikaciji. Mi smo to slabost reševali z uporabo vtičnikov, ki so na eni strani pohitrili razvoj in na drugi tudi prispevali k strukturiranosti kode.

3.2.2 Zepto

Zepto [14] je *JavaScript* knjižnica, ki je po sintaksi zelo podobna njej konkurečni knjižnici *jQuery* oz. *jQuery Mobile*. Zasnovana je na modularnem sistemu, tako da lahko enostavno vključimo samo tiste module, ki jih potrebujemo. Zaradi zasnove in funkcionalnosti je zelo primerna za razvoj tako hibridnih kot spletnih aplikacij.

Prednosti:
V primerjavi z ostalimi knjižnicami, je *Zepto* zelo minimalistična knjižnica, kar se odraža v njeni velikosti. To je lahko prednost, če imamo prostorske omejitve. Poleg tega je tudi v nekaterih primerih hitrejša, vendar je to čisto odvisno od scenarija in brskalnika, v katerem se izvaja. Ima dobro podporo za večino brskalnikov, med njimi je tudi brskalniki na mobilnih napravah.

Slabosti:
Za naše zahteve je bila največja omejitev, da ne deluje oz. ima kompatibilnostne težave z spletnim brskalnikom *Internet Explorer*. Če vemo, da je to privzeti brskalnik na platformi *Windows Phone*, potem ta knjižnica ni primerna. Ostalih slabosti nismo raziskovali.

3.2.3 Sencha Touch

Sencha Touch [15] je zasnovana na *Sencha HTML5* tehnologiji. Ima podporo za veliko število platform kot npr. *iOS, Android, Windows Phone, BlackBerry* itd. Poleg hitrosti in uporabe zadnjih standardov vključuje tudi veliko število gradnikov. Tako je razvoj hitrejši in končna aplikacija je bolj prilagojena za specifično platformo.

Prednosti:
Sencha Touch je strukturno zelo dobro zasnovana. Omogoča nam razvoj velikih aplikacij, saj nas prisili k bolj strukturiranem pisanju kode. Poleg tega ima funkcionalnosti, ki so razvite posebej za mobilne naprave. Vključuje veliko tem, ki jih lahko uporabimo kot osnovo za razvoj. Hkrati je odprtokodna rešitev.

Slabosti:
Slabost *Sencha Touch* smatramo v tem, da se preveč zanaša na *JavaScript* kot glavno orodje za grajenje vmesnika. Nekatere scenarije bi lahko bolj elegantno in hitreje rešili s tehnologijo *HTML5* ali *CSS3*. Poleg tega je zaradi velikosti in kompleksnosti zelo težko razhroščevati knjižnico. Skupnost je manjša v primerjavi z *jQuery Mobile*, zato je na voljo tudi manj dodatkov.

3.3 Mape

Zaradi zahteve, da lokacije vizualiziramo na mapi, smo pregledali 3 najbolj razširjene, preverjene in stabilne rešitve za mape.

3.3.1 Google Maps

Google Maps [16] je tehnologija razvita s strani spletnega velikana *Google*. Poleg uporabe na njihovi uradni strani in znotraj njihovih aplikacij, omogočajo uporabo map tudi v drugih aplikacijah preko vmesnika. Uporablja se tako

na spletnih straneh kot v mobilnih aplikacijah. *Google Maps* ima uradno knjižnico za ustvarjanje in upravljanje map.

Prednosti:
Dobra pokritost z veliko lokacijskimi informacijami. Omogočajo več tipov map in jih enostavno vključimo v naše rešitve preko vmesnika. Obenem ponuja veliko vtičnikov, s katerimi lahko manipuliramo mapo ter ji dodajamo elemente.

Slabosti:
Google Maps so na voljo samo za Android in iOS, če želimo razvijati domorodno aplikacijo. Če želimo mape vključiti v hibridne ali spletne aplikacije, potrebujemo spletno povezavo. Zadnje verzije Google Maps omogočajo tudi lokalno shranjevanje, vendar v zelo omejeni obliki. Zaradi avtorskih pravic jih nismo smeli shranjevati ter spreminjati.

3.3.2 Bing Maps

Bing Maps [17] je spletna storitev v lasti korporacije *Microsoft*. Omogoča veliko različnih pogledov in ponuja prikaz navigacije med različnimi destinacijami. Kot *Google Maps*, lahko *Bing Maps* vključimo v naše aplikacije preko vmesnika. V pomoč nam je knjižnica, preko katere vzpostavimo okolje za prikaz in urejanje map.

Prednosti:
Bing Maps pridobiva podatke iz različnih kvalitetnih virov, zato je natančnost map zelo velika. Preko vmesnika se enostavno vključijo v ostale aplikacije. Knjižnica za ustvarjanje in urejanje map omogoča enostavno delo in je dobro dokumentirana.

Slabosti:
Bing Maps so privzete mape v operacijskem sistemu *Windows Phone*. Sam

operacijski sistem omogoča lokalno shranjevanje map in njihovo uporabo brez spletne povezave. Vendar ne moremo dostopati do map iz drugih aplikacij kot tistih, ki so razvite s strani podjetja *Microsoft*. Če jih želimo vključiti v vse tipe aplikacij, moramo imeti spletno povezavo.

3.3.3 OpenStreetMap in Leaflet

OpenStreetMap [18] je projekt skupnosti, ki želi ustvariti odprtokodne mape. Glavni razlog za ustvaritev takšnega projekta je potreba po mapah, ki jih lahko brez omejitev spreminjamo in uporabimo v aplikacijah. Glavni vir podatkov so prostovoljci, ki dodajajo in urejajo mape.

Ker tehnologija *OpenStreetMap* ponuja prenos map v slikovni obliki, smo potrebovali tudi knjižnico za njihovo upravljanje. *Leaflet* je odprtokodna knjižnica, ki je primarno razvita za hibridne in spletne mobilne aplikacije in omogoča ustvarjanje in nadzor nad mapami kot npr. dodajanje elementov na mapo, zoomiranje mape, premikanje med različnimi lokacijami na mapi itd.

Prednosti:
Največja prednost tehnologije *OpenStreetMap* je odprtokodnost. Možnost dodajanja, urejanja in shranjevanja nam omogoča veliko možnosti vključevanja map v aplikacije. Z lokalnim shranjevanjem map ne kršimo nobenih avtorskih pravic, obenem pa jih lahko uporabimo tako z kot brez internetne povezave.

Slabosti:
Avtorji projekta *OpenStreetMap* nimajo za seboj nobene velike korporacije kot to imajo *Google Maps* in *Bing maps*. Posledica je, da ni finančnih sredstev za dodelavo map, vendar to rešujejo s pomočjo prostovoljcev. Ker lahko mape vsakdo ureja, obstaja nevarnost napačnih map, kar lahko povzroči nevšečnosti pri uporabi v navigaciji ali v poslovnem svetu.

3.4 Shranjevanje podatkov

Velika količina podatkov in potreba po poizvedovanju nad njimi, nas je prisilila k temeljiti preučitvi vseh podatkovnih baz. Za razvoj mobilnih aplikacij so na voljo 2 tipa podatkovnih baz:

- shranjevanje podatkov znotraj brskalnika;
- shranjevanje podatkov v plasti domorodne aplikacije.

Razlike se nahajajo v hitrosti, načinu shranjevanja podatkov in v načinu poizvedovanja nad podatki.

3.4.1 Web Storage

Web Storage [27] je podatkovna baza, ki deluje znotraj brskalnika. Zasnovana je na sistemu ključ-vrednost. Trenutna implementacija omogoča samo shranjevanje tekstovnih oblik podatkov, zato jih moramo v primeru drugačnega formata spremeniti v ustreznega. Samo upravljanje z bazo izvajamo s tehnologijo *JavaScript*.

Prednosti:
Zaradi zasnove je zelo hitra in enostavna za uporabo. Večina platform, med njimi *Android*, *iOS* in *Windows Phone*, podpirajo *Web Storage*. To pomeni, da to podatkovno bazo lahko brez skrbi zaradi kompatibilnosti uporabimo pri razvoju hibridnih ali spletnih aplikacij.

Slabosti:
Ne omogoča shranjevanja bolj kompleksnih struktur in tipov podatkov. Ključ-vrednost zasnova nam ne omogoča naprednih poizvedb ampak samo po ključu. Če želimo poizvedovati po vrednosti, moramo prebrati celotno bazo, kar je časovno zamudno.

3.4.2 Web SQL Database

Web SQL Database [28] je relacijska baza kor npr. *MySQL, PostgreSQL, MSSQL* itd. Kot je razvidno iz imena, je namenjena za delovanje oz. uporabo znotraj brskalnika. Omogoča strukturirano shranjevanje podatkov ter uporablja *SQL* za poizvedovanje nad podatki.

Prednosti:
Možnost shranjevanje podatkov v strukture ter napredno poizvedovanje nad vsebino podatkovne baze je velika prednost pred ostalimi podatkovnimi bazami, ki bazirajo v brskalniku. Kljub kompleksnosti in velikem nabor funkcionalnosti, ponuja hitro pisanje in branje podatkov.

Slabosti:
Web SQL Database se izvaja znotraj brskalnika, vendar ni podprta v vseh popularnih brskalnikih, zato tudi ni podprta v vseh platformah. Hkrati je opuščena tehnologija in ni primerna za uporabo.

3.4.3 Indexed Database

Indexed Database [29] je z zasnovo in funkcionalnostmi med *Web Storage* in *Web SQL Database* podatkovnimi bazami. Zasnovana je na ključ-vrednost zasnovi, vendar podpira dodatne ključe za hitrejše iskanje podatkov.

Prednosti:
Podatkovna baza ponuja enostaven vmesnik za upravljanje s podatki. Zaradi dodatnih ključev, omogoča hitro iskanje podatkov, zato nam ni potrebno naložiti vseh podatkov za iteracijsko iskanje med njimi.

Slabosti:
Indexed Database zaradi dodatnih ključev zavzema dodatni prostor. Zasnova ključ-vrednost ne omogoča shranjevanja kompleksnih struktur podat-

kov. Obenem ne ponuja naprednih poizvedb, ki so potrebne za večino aplikacij.

3.4.4 SQLite

SQLite [30], [2] je relacijska podatkovna baza, razvita v programskem jeziku C. Namenjena je za manjše količine podatkov (če jo primerjamo z drugim relacijskimi podatkovnimi bazami). Kljub majhnosti in enostavnosti, ponuja dovoljšen nabor funkcionalnosti za shranjevanje podatkov in poizvedovanje nad njimi z jezikom *SQL*.

SQLite je odprtokodni projekt in je zelo razširjen na vseh platformah, med njimi tudi *Android*, *iOS* in *Windows Phone*. Brez večjih omejitev jo vključimo v naše domorodne aplikacije. Pri hibridnih aplikacijah mora obstajati možnost delovanja baze v spodnji plasti in komunikacije oz. izmenjave ukazov in podatkov med spodnjo in zgornjo plastjo.

Prednosti:
Enostavnost in majhnost *SQLite* podatkovne baze je zelo primerna za mobilne platforme. Ravno zato je zelo razširjena in obstaja ogromno vmesnikov za uporabe le-te.

Ker je relacijska podatkovna baza, lahko shranimo tudi kompleksne strukture podatkov. S jezikom *SQL* lahko izvajamo enostavne ali kompleksne poizvedbe po bazi.

Slabosti:
SQLite podatkovna baza je razvita v programskem jeziku C, zato nam ni na voljo v brskalniku; posledično je ne moremo uporabiti v spletnih aplikacijah. Vsaj ne v tej meri, da bi se podatki shranjevali v lokalno podatkovno bazo.

Tudi pri hibridnih aplikacijah ni na voljo, če to ni podprto v vseh plasteh. Pri ogrodju *PhoneGap* so na voljo neuradni vtičniki za vsako platformo. Slabost je, da niso preverjene rešitve in v večini primerov vsebujejo veliko število napak.

3.5 Ugotovitve

Po preučitvi vseh zahtev in rešitev, smo prišli do naslednjih ugotovitev.

Za zaledni del smo uporabili *PhoneGap*. Omogočil nam je hitrejši razvoj, z manjšimi spremembami smo pokrili vse zahtevane platforme in po vrhu je še odprtokodni projekt.

Za čelni del smo uporabili *jQuery Mobile*, ker je še najbolj preverjena in stabilna rešitev. Ima ogromno vtičnikov oz. ima enostaven način za dodajanje lastnih dodatkov, kar nam je pomagalo pri razvoju. Hkrati imamo največ izkušenj z uporabo te knjižnice, zato ni bilo potrebe po dodatnem izobraževanju.

OpenStreetMap in *Leaflet* smo uporabili za prikazovanje lokacij na mapi. Možnost, da smo lahko mape shranili lokalno, je bila zmagovalna prednost pred ostalimi rešitvami.

Zaradi večje količine podatkov, potrebe po bolj kompleksnem poizvedovanju nad njimi ter obstoj dodatkov za *PhoneGap*, smo se odločili za *SQLite*. Obenem, če bi v bodoče razvijali domorodno aplikacijo, ne bi potrebovali spreminjati strukture podatkov ali načina poizvedovanja nad njimi.

Slika 3.3: Shema različnih plasti mobilne aplikacije in uporabljene tehnologije.

Poglavje 4

Uporabljene tehnologije

Uporabili smo tehnologije, za katere menimo, da so primerne za razvoj mobilnih oz. hibridnih aplikacij. Zaradi poznavanja izbranih tehnologij, nismo imeli problemov pri njihovi uporabi. Hkrati smo se zavedali, da so med seboj kompatibilne in stabilne, kar se odraža v večji kvaliteti in uporabnosti končne aplikacije.

4.1 Razvojno orodje

Na voljo smo imeli ogromno razvojnih orodji, vendar smo za vsako platformo uporabili najbolj stabilno, enostavno in hitro orodje. Hkrati so orodja morala omogočati razhroščevanje. Za *Android* smo uporabili *Eclipse* [20], za *iOS* *Xcode* [21] in za *Windows Phone Visual Studio* [22].

4.2 HTML5 in CSS3

HTML5 [23] je označevalni jezik in je naslednik zelo razširjenega standarda *HTML4*. Trenutno je v razvoju že verzija 5.1 [24]. Predstavlja glavne gradnike uporabniškega vmesnika oz. zaslonske maske celotne mobilne aplikacije.

CSS3 [25] je standard za oblikovanje spletnih strani. Stran zgradimo z *HTML5* gradniki, nato pa jo oblikovno dopolnimo s *CSS3*. Prednost je v tem, da ločimo več plasti razvoja spletni strani ali hibridne mobilne aplikacije ter povečamo preglednost in enostavnost razvoja in vzdrževanja.

4.3 JavaScript

JavaScript [4] je programski jezik, ki ga je razvil Brendan Eich leta 1995 pod imenom *Mocha*. Pozneje je bil preimenovan v *JavaScript*. Primarno se izvaja znotraj brskalnika, zato je zelo primeren za razvoj spletnih strani. V zadnjem času pridobiva na prepoznavnosti tudi pri razvoju mobilnih aplikacij.

Za razvoj naše aplikacije smo ga uporabili zaradi možnosti delovanja na različnih platformah. Če želimo razviti hibridno aplikacijo, smo bili primorani uporabiti *JavaScript* za komunikacijo s plastmi. Obenem nam služi tudi pri uporabniškem vmesniku, saj skrbi za ustrezno prikazovanje vseh gradnikov.

4.4 PhoneGap

PhoneGap [9] je ogrodje za razvoj mobilnih hibridnih aplikacij z uporabo tehnologij *JavaScript, HTML5 in CSS3*. Trenutno je aktualna verzija 3.1.0. in podpira mnogo različnih platform, med njimi *Android, iOS, Windows Phone, BlackBerry* in druge. Več o ogrodju *PhoneGap* smo že omenili v poglavju 3.1.2.

4.5 SQLite

SQLite [30], [2] je podatkovna baza, ki je razvita v programskem jeziku C. Razvita je bila že davnega leta 2000 in je do danes prejela mnogo posodobitev in popravkov. *SQLite* uporablja *SQL* za upravljanje nad podatki.

Na voljo je ogromno vmesnikov v različnih programskih jezikih kot npr. *BASIC, Delphi, PHP, Java, Python* itd. Uporabljena je v veliko znanih projektih oz. rešitvah kot npr. *Skype, Mozilla Firefox, Google Chrome* itd.

4.6 jQuery in jQuery Mobile

Knjižnici *jQuery* in *jQuery Mobile* sta razviti v programskem jeziku JavaScript in sta namenjeni za spletni razvoj. Podpirata vse popularne brskalnike, pri tem skrbiza tudi za kompatibilnost s starejšimi oz. še vedno aktualnimi verzijami brskalnikov. Zato pri uporabi teh knjižnic nimamo skrbi zaradi kompatiblnih nevarnosti.

Knjižnica *jQuery* je odprtokodna, razvita pod MIT licenco z začetki v letu 2006. Glede na statistike [31], je najbolj razširjena in uporabljena *JavaScript* knjižnica na spletu. Sintaksa je oblikovana v smeri enostavnega izbiranja gradnikov spletne strani, izvajanja animacij, določanja in lovljenja raznih dogodkov in za razvoj *Ajax* aplikacij. Poleg osnovne knjižnice so nam na voljo tudi vtičniki.

Vtičnik *jQuery Mobile* je namenjen za razvoj mobilnih aplikacij, saj vključuje dodatne funkcionalnosti ter teme za hiter in enostaven razvoj.

4.7 JSON

JSON [26] ali JavaScript Object Notation je podatkovni format, ki je primarno namenjen za podatkovno izmenjavo med različnimi pošiljatelji in prejemniki. Je neodvisen od platform in programskih jezikov in deluje na strukturi ključ-vrednost.

Isto količino podatkov predstavimo v mnogo prostorsko manjšem formatu v primerjavi z konkurenčnim formatom *XML*. Zato se veliko uporablja za prenos podatkov pri mobilnih aplikacij. Prenos podatkov je v večini primerov problem, saj se količina prenešenih podatkov odraža na večjem računu pri mobilnem operaterju. Na voljo je veliko knjižnic, ki omogočajo pretvorbo

objektov ali drugi kompleksnejših programskih struktur v *JSON* format in obratno. Zaradi teh razlogov, je *JSON* razširjen in pogosto uporabljen pri razvoju.

4.8 OpenStreetMap

Tehnologijo *OpenStreetMap* [18] smo predstavili v poglavju 3.3.3. Prej kot samo tehnologijo, *OpenStreetMap* predstavlja projekt skupnosti. Začetki izvirajo iz leta 2004, ko je Steve Coast ustvaril projekt s fokusom, da razvije mape za Veliko Britanijo. Kmalu zatem so se osredotočili tudi na druge države, saj so pridobili veliko podporo tako med ljudmi kot med različnimi podjetji. Projekt vključuje hrambo lokacijskih podatkov ter njihovo urejanje. Omogočajo tudi sam izvoz podatkov oz. njihov ogled na uradni spletni strani.

4.9 Leaflet

Leaflet [19] is *JavaScript* knjižnica za prikaz interaktivnih map na mobilnih napravah. Razvita je s fokusom na hitrosti, enostavnosti pri razvoju in uporabnosti pri prikazovanju map. Deluje tako znotraj vseh popularnih brskalnikov, ne samo tistih na mobilnih napravah.

Avtor je Vladimir Agafonkin, ki skupaj s skupino zvestih pomočnikov razvija in vzdržuje knjižnico. Omogoča razširitev osnovnih funkcionalnosti z vtičniki. Uporabljena je s strani večjih podjetji, med njimi FourSquare, Flickr, craiglist, Data.gov in drugi.

Poglavje 5

Spletna administracija

Namen spletne administracije je shranjevanje, agregiranje in urejanje podatkov na enem mestu. Cilj je omogočiti enostaven način za dodajanje novih ali urejanje obstoječih lokacij, urejanje podatkov o Univerzi v Ljubljani in mestu Ljubljani ter dodajanje in urejanje uporabnih nasvetov. Prednost je v enostavnem vmesniku, saj vsakdo, ki ima dovoljenje lahko dostopa do administracije preko spleta ter samostojno ureja vsebino. Tako smo zagotovili, da lahko v bodoče na projektu sodeluje več ljudi, ki skrbijo, da so podatki verodostojni.

Primaren namen diplomske naloge je razvoj mobilne aplikacije, vendar smo prišli do zaključka, da moramo vključiti tudi spletno administracijo. Tako smo dosegli, da mobilna aplikacija ima zagotovljen verodostojen vir podatkov in po potrebi osveži lokalne podatke z njihovo zadnjo verzijo.

5.1 Preučitev virov in formatov podatkov

Prvi korak je obsegal preučitev podatkov, ki so nam na voljo na spletu. To vključuje pregled, v kakšnem formatu so trenutni podatki in v kakšen format jih moramo shraniti, če jih želimo uporabiti v mobilni aplikaciji. Večina podatkov je bila na voljo na spletnih straneh ustanov, ki smo jih želeli vključiti. Mobilna aplikacija bi lahko črpala podatke kar direktno z njihovih spletnih

strani, vendar ima takšen pristop par nevarnosti:

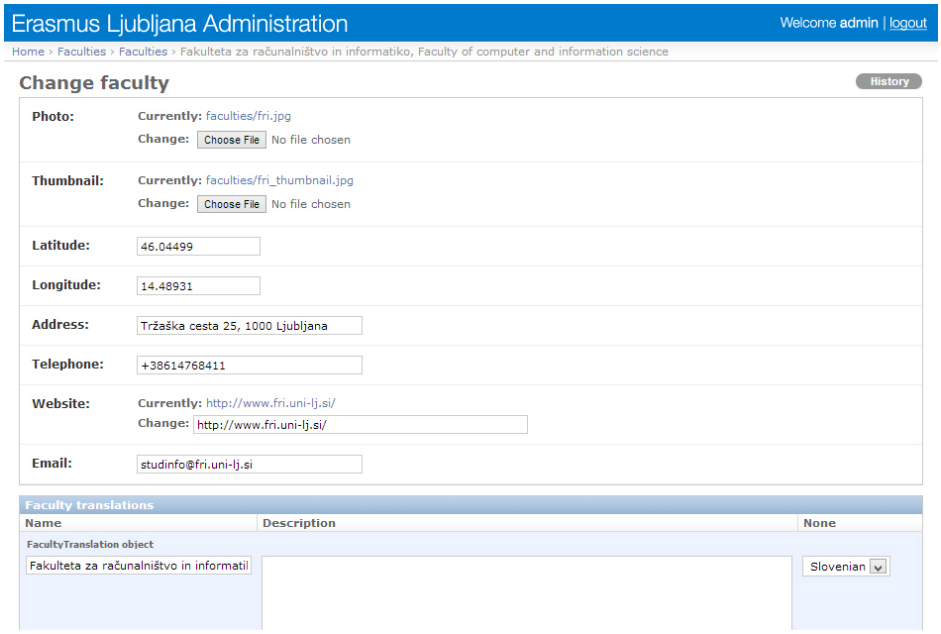
- začasno nedelovanje spletne strani ustanove lahko prepreči pridobitev oz. osvežitev podatkov;
- v primeru spremembe oblike strani, obstaja velika verjetnost, da podatki ne bodo več v ustreznem formatu;
- pri dostopu do potrebnih podatkov se poleg njih po nepotrebnem prenašajo tudi druge vsebine kot npr. slike, posnetki, reklame itd. kar se odraža v daljšem času prenosa in večji količini prenešenih podatkov;
- podatki lahko vsebujejo napake.

Z našim sistemom smo minimizirali te nevarnosti. En vir podatkov zmanjša potrebo po pregledovanju in črpanju podatkov iz različnih spletnih strani. Obenem smo vse podatke ročno preverili, zato se nevarnost neveljavnih podatkov zmanjša.

5.2 Razvoj administracije

Za razvoj administracije smo uporabili ogrodje *Django* [6], ki temelji na programskem jeziku *Python*. Razlog za našo izbiro tega ogrodja leži v preprosti uporabi ter hitremu razvoja. Za podatkovno bazo smo uporabili *MySQL*. Z ogrodjem *Django* smo ustvari ustrezne tabele v podatkovni bazi ter vzpostavili sistem na njihovo urejanje.

Administracija omogoča urejanje in pregled nad vsemi ustanovami, ki jih želimo prikazati v mobilni aplikaciji. Poleg tega omogoča tudi pregled in urejanje podatkov o Ljubljani ter uporabnih nasvetih. Administracija beleži čas zadnje spremembe. Ta čas uporabimo v mobilni aplikaciji za preverjanje, ali je na voljo nova verzija podatkov. Primer vmesnika za urejanja podatkov o fakulteti lahko vidimo na sliki 5.2.

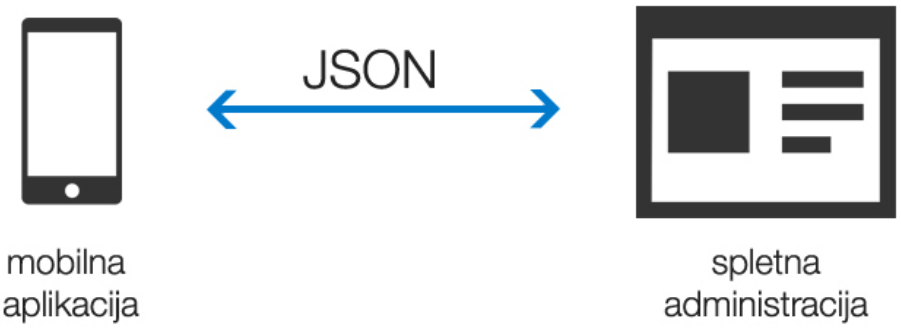


Slika 5.1: Prikaz uporabniškega vmesnika za dodajanje in urejanje podatkov o fakulteti.

5.3 Razvoj komunikacijskega vmesnika

Tretji in zadnji korak je bil razvoj vmesnika. Ko smo v prvem koraku spoznali nevarnosti črpanja nestrukturiranih podatkov z različnih spletnih strani, smo potrebovali ustrezen vmesnik za rešitev tega problema. Vmesnik služi kot dodatek administraciji ter ponuja mobilni aplikaciji dostop do strukturiranih podatkov. To pomeni, da so vsi podatki predstavljeni v isti strukturi, ki jo mobilna aplikacija nima težav prebrati.

Vmesnik je dostopen preko spletnega naslova. V trenutni verziji nismo implementirali nobenega mehanizma za omejitev dostopa. Vendar obstaja možnost nadgradnje, ki bi omogočal dostop do vmesnika preko ključev. Vsak odjemalec bi pridobil svoj ključ s katerim bi imel omogočen dostop. Za vsak ključ oz. odjemalca bi merili njegovo aktivnost ter mu v primeru zlorab omejili dostop. Ker imamo trenutno enega odjemalca podatkov, nismo videli potrebe po takšnem mehanizmu. Edina omejitev je, da so podatki na voljo samo za branje. Preko vmesnika se podatki ne morejo spreminjati.



Slika 5.2: Shema komunikacije mobilne aplikacije in spletne administracije.

Podatki so v formatu *JSON*. V primerjavi z drugimi formati, je mnogo bolj razširjen in na voljo je veliko knjižnic, ki znajo *JSON* format pretvoriti v objekte ali druge strukture. Tudi knjižnica jQuery to omogoča. Primer kode 5.1 prikazuje pretvorbo *JSON* formata v *JavaScript* format.

Primer kode 5.1: Pretvorba JSON formata v JavaScript s pomočjo knjižnice jQuery

```
var user = jQuery.parseJson({
  "first_name": "Janez",
  "last_name": "Novak"
});

console.log(user['first_name']) // izpise Janez
console.log(user['last_name']) // izpise Novak
```

5.3.1 Dostop do podatkov

Do podatkov dostopamo preko spletnih naslovov. Vsak sklop podatkov o lokacijah, Univerzi v Ljubljani, mestu Ljubljani in uporabnih nasvetih ima

definiran spletni naslov. Vsakdo, ki želi podatke pridobiti, mora narediti *GET* zahtevek na ustrezno spletno povezavo. Če je vse veljavno, vmesnik vrne podatke v formatu *JSON*. Primer lahko vidimo v primeru kode 5.2.

Primer kode 5.2: Primer podatkov od vmesnika v formatu JSON


```
{
  "meta": {
    "limit": 50,
    "next": null,
    "offset": 0,
    "previous": null,
    "total_count": 22
  },
  "objects": [
    {
      "address": "Trzaska cesta 25, 1000 Ljubljana",
      "email": "studinfo@fri.uni-lj.si",
      "latitude": 46.04499,
      "longitude": 14.48931,
      "telephone": "+38614768411",
          ...
```

Poglavje 6

Razvoj mobilne aplikacije

Razvoj mobilne aplikacije smo izvedli v sklopih. Tak pristop smo izbrali iz razloga, ker ponuja večji pregled nad samim potekom razvoja ter smo lahko posamezne dele takoj testirali. Za vsak sklop smo zastavili cilje, ki izpolnjuje vse zahteve, ki smo jih določili v prejšnjih poglavjih. Ker smo vložili veliko truda v definiranje strukture, specifikacije in katere tehnologije bomo uporabili pri razvoju mobilne aplikacije, je sam razvoj potekal tekoče.

Sklopi so urejeni po vrstnem redu razvoja in so vsebovali naslednje korake:

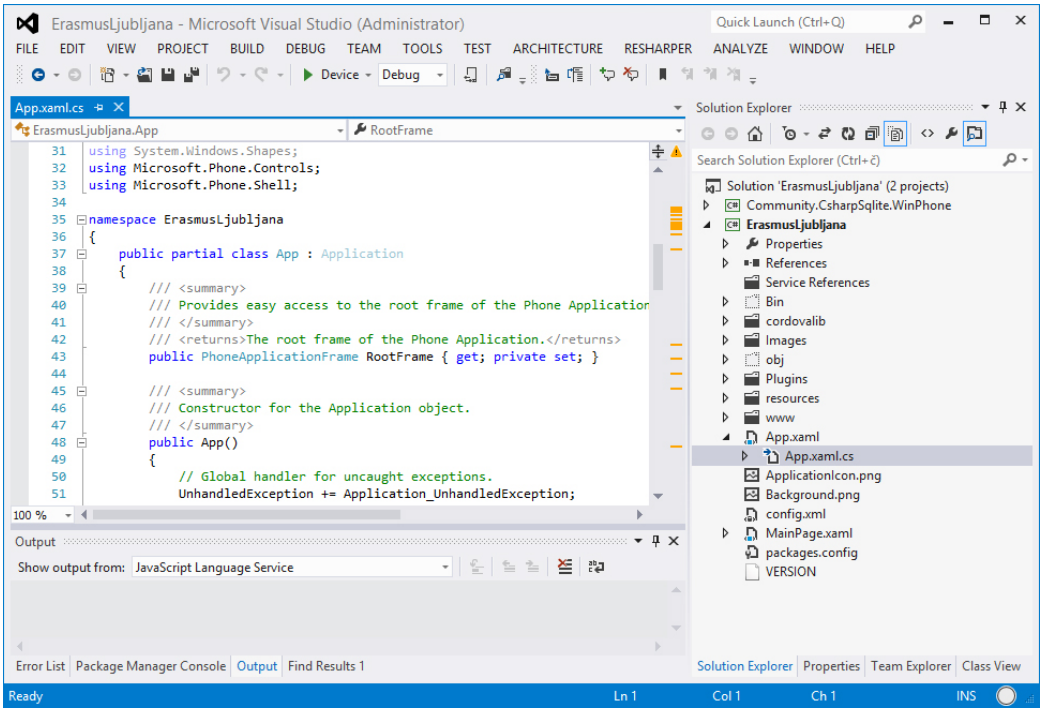
- priprava razvojnega okolja;
- razvoj mobilne aplikacije;
- testiranje mobilne aplikacije.

6.1 Priprava razvojnega okolja

Priprava razvojnega okolja je namenjena za razvoj mobilne aplikacije. Na voljo so nam bila različna orodja, vendar smo večino razvoja izvedli v orodju *Visual Studio*. Vsekakor smo uporabili tudi *Eclipse* in *Xcode*, katerih naloga je bila pomoč pri testiranju same aplikacije na platformah *Android* in *iOS*.

Začeli smo z razvojem za platformo *Windows Phone*. To vključuje prenos knjižnice Windows Phone SDK ter namestitev na računalniku. Za ustvari-

tev *PhoneGap* aplikacije smo uporabili predlogo, ki je na voljo na spletu. Ko smo s tem uspešno zaključili, smo vzpostavili tudi testno napravo. Kljub temu, da je na voljo emulator za izvajanje aplikacij, nismo bili zelo navdušeni nad njim. Glavne pomanjkljivosti se skrivajo v počasnosti in slabem testiranju uporabniške izkušnje. Zelo težko ali nemogoče je simulirati gibe, ki jih izvajamo na mobilni napravo preko računalnika.



Slika 6.1: Razvojno orodje Visual Studio s projektom ErasmusLjubljana.

Da smo lahko uporabili testno napravo *HTC 8X*, smo jo morali prvo odkleniti za razvoj. Sam proces je bil preprost, samo potrebovali smo ustrezen uporabniški račun, ki dokazuje, da smo vključeni v Univerzo v Ljubljani. S tem nam ni bilo treba plačati letne naročnine.

Podobni koraki so potekali tudi pri vzpostavitvi okolja za platformi *Android* in *iOS*. Pri obeh smo morali prenesti in namestiti ustrezni uradni knjižnici, ki sta nam omogočili razvoj. Za testiranje na *Android* napravi nam ni bilo potrebno odkleniti naprave. Za *iOS* oz. za testiranje na mobilni napravi iPhone pa smo morali pridobiti razvijalski ključ.

Glavni del aplikacije smo razvili s spletnimi tehnologijami. Eden od načinov razvoja in testiranja je tudi znotraj brskalnika, ki omogoča napredno razhroščevanje. Vendar zaradi enostavnega razvoja kar direktno na mobilni napravi, se tega načina nismo posluževali.

6.2 Razvoj mobilne aplikacije

Sklop razvoja mobilne aplikacije smo razdelili v manjše podsklope. Z razdelitvijo smo dosegli bolj pregledno in lažje vzdržljivo kodo. Vsak sklop je neodvisen, vendar je kompatibilen z drugimi sklopi. V naslednjih poglavjih bomo vsak sklop predstavili in vključili primere, ki so relavantni za delovanje celotne aplikacije.

6.2.1 Osnovna struktura

Vsaka platforma ima drugačno strukturo aplikacije. Fokusirali smo se samo na hibridni del aplikacije, ki je razvita s spletnimi tehnologijami in je ista za vse platforme. Osnovna struktura je sestavljena iz 8 delov:

index.html

Osnovna datoteka, v katero smo vključili vse potrebe knjižnice in stile ter definirali obliko začetne strani.

cordova-2.7.0.js

JavaScript knjižnica ogrodja *PhoneGap*, ki vsebuje vse potrebne metode za komunikacijo z domorodnim delom ogrodja. Za vsako platformo moramo vključiti specifično knjižnico, saj se razlikujejo.

css/

Mapa, v kateri so definirani vsi stili. Vključuje stile za knjižnico *jQuery Mobile*, *Leaflet* ter stile, ki smo jih sami definirali.

fixtures/

V mapi fixtures so shranjeni vsi začetni podatki, ki se ob prvem zagonu

aplikacije shranijo v prazno SQLite podatkovno bazo. Podatki so v formatu *JSON* in slike v formatu *JPG*. Obstaja možnost, da bi začetne podatke prenesli s spleta, vendar je ta način počasnejši in potrebuje internetno povezavo.

img/

V mapi *img/* so shranjene vse slike, ki so uporabljene pri oblikovanju aplikacije. Ne vsebuje slik, ki se prenesejo s spleta, ker se te shranijo v to posebej ustvarjeno mapo.

js/

Mapa vsebuje logiko za celotno aplikacijo. Samo vsebino ter namen vsakega dela bomo predstavili v naslednjih poglavjih.

locales/

V mapi so shranjene datoteke za prevode. Trenutno vsebuje prevode za Slovenščino in Angleščino in se po potrebi lahko jeziki enostavno dodajajo.

tiles/

Mapa *tiles* hrani podatke za lokalno prikazovanje mape. Mapa je predstavljena v obliki slik v formatu *PNG*, ki so strukturirane v mape. Ime mape določa povečavo in lokacijo mape.

views/

V zadnji mapi views so shranjene *HTML* datoteke, ki predstavljajo različne poglede. To vključuje npr. predstavitev fakultete, nastavitve, proces osveževanja podatkov itd.

6.2.2 Poslovna logika

Vzpostavitev aplikacije

Pri zagonu aplikacije domorodni del vzpostavi brskalnik znotraj aplikacije. V brskalniku izvede lokalno datoteko *index.html*, ki vključuje klic *JavaScript*

metode. Logika za zagon aplikacije je locirana v *js/app.js*. Poleg ostalih metod imamo glavno metodo z imenom *initialize*.

Primer kode 6.1: Primer vzpostavitvene kode mobilne aplikacije

```
initialize: function()
{
  // attach event and wait until device is ready
  document.addEventListener('deviceready', this.onDeviceReady,
      false);

  $(document).on("mobileinit", function()
  {
    // Setting default page and dialog transition to none
    $.mobile.defaultPageTransition = 'slide';
    $.mobile.defaultDialogTransition = 'none';
    // PhoneGap Support options
    $.mobile.allowCrossDomainPages = true;
    $.mobile.phonegapNavigationEnabled = true;
    $.support.cors = true;
    // DOM caching
    $.mobile.page.prototype.options.domCache = true;
    $.mobile.buttonMarkup.hoverDelay = 0;

    // Remove page from DOM when it's being replaced
    $('div[data-role="page"]').on('pagehide', function(event, ui)
    {
      $(event.currentTarget).remove();
    });
  });
},
onDeviceReady: function()
{
  // load settings
  app.settings(function()
```

```
  {
    // start application
    app.start();
  });
},
```

Naloga te metode je, da doda dogodek, ki se izvede, ko je PhoneGap ogrodje naloženo oz. je mobilna naprava pripravljena. Namen je, da moramo prvo počakati, da se ogrodje vzpostavi in da lahko dostopamo do komponent naprave. Poleg tega določimo tudi nastavitve za knjižnico *jQuery Mobile*, da je kompatibilna z ogrodjem PhoneGap.

Ko je naprava pripravljena, se izvede metoda *onDeviceReady*, ki naloži preostali del aplikacije. V našem primeru naloži nastavitve, ki so shranjene v lokalni shrambi. Ko to uspešno konča, kliče metodo *start*.

Metoda *start* prvo vzpostavi *router* ali krmilnik, katerega naloga je, da izvede določeno kodo ob obisku določene podstrani v aplikaciji. Prednost je, da smo kodo lažje strukturirali teh ločili podstrani. Poleg krmilnika se vzpostavi tudi podatkovna baza. V primeru uspešnosti, se zažene prezentacijski del aplikacije.

Primer kode 6.2: Prikaz vzpsotavitve poslovne logike mobilne aplikacije

```
start: function()
{
  // initialize router
  app.router.start();

  // start orm to create connection to database
  app.orm.start({
    success: function()
    {
      // everything went OK, load main controller
      app.controllers.MainController();
    },
```

```
    error: function(e)
    {
      alert('error');
    }
  });
},
```

Podatkovna baza

Logiko za dostop do podatkovne baze smo shranili v datoteko *js/orm.js*, ki skrbi za vzpostavitev povezave do *SQLite* podatkovne baze in pripravo tabel ter začetnih podatkov. Hkrati vsebuje vso logiko za poizvedovanje po podatkovni bazi.

Z ločevanje logike smo dosegli večjo preglednost kode. Za vsako poizvedbo smo razvili svojo metodo, ki vsebuje klic do podatkovne baze. To metodo smo uporabili v drugih plasteh aplikacije. V primeru, če bi morali kakorkoli spreminjati strukturo podatkov, bi naredili spremembo samo v *js/orm.js* datoteki. Z vidika vzdrževanja je to velika prednost.

Primer kode 6.3: Prikaz vzpostavitve povezave do lokalne podatkovne baze in priprava vseh potrebnih podatkov

```
start: function(options)
{
  // create connection to database ErasmusLjubljana and wait until
     connection is successful
  this.db = window.sqlitePlugin.openDatabase('ErasmusLjubljana',
     "1.0", "database", -1, function()
  {
    // try to load dummy data from database to check if database is
       already created and populated
    app.orm.db.transaction(function(tx)
    {
      tx.executeSql("SELECT id FROM faculties LIMIT 1", [],
```

```
        function()
    {
      // everything OK, notify that database started successfully
      options.success(false);
    });
  },
  function()
  {
    // database is empty, so first we need to create tables and
       second populate it with default data
    app.orm.db.transaction(app.orm._createTables, options.error,
       function()
    {
      app.orm._fixtures(options.success, options.error);
    });
  });
});
},
```

Za preverjanje, ali podatkovna baza že vsebuje podatke, smo zaradi pomanjkanja funkcionalnosti za preverbo obstoja tabel v bazi uporabili drugačen pristop. Naš pristop temelji na dejstvu, da prvo poskušamo naložiti podatek z *SQL* poizvedbo. Če pride do napake, smatramo to napako kot signal, da podatki še niso naloženi. Zato se požene metoda, ki ustvari vse tabele in jih popolni z začetnimi podatki.

Polnitev z začetnimi podatki deluje na principu, da preberemo lokalno shranjene podatke, ki so v formatu *JSON*. Primer nalaganja podatkov za fakultete je prikazan v primeru kode 6.4.

Primer kode 6.4: Primer shranjevanja lokalnih podatkov v formatu JSON v SQLite podatkovno bazo

```
var path = window.location.href.replace('index.html', '');
$.getJSON(path + "fixtures/faculties.json", function(faculties)
```

```
{
  app.orm.faculties.sync(faculties, success, error);
});
```

Za poizvedovanje nad podatki se uporablja *SQL*. Jezik JavaScript omogoča ustvarjanje struktur, ki vsebujejo metode. Tako smo dosegli, da smo ustvarili imena metod oz. poti to metod, ki so razumljive. Kodo v primeru 6.5 kličemo *faculties.find.by.id(1, 'en', function()...)*.

Primer kode 6.5: Primer pridobitve fakultete po ključu id iz podatkovne baze

```
faculties: {
  find: {
    by: {
      id: function(id, language, success)
      {
        // app.orm.query method accepts SQL, parameters and success
           callback
        app.orm.query('SELECT * FROM faculties f JOIN
           faculty_translation t ON f.id=t.id WHERE t.language=?
           AND f.id=? LIMIT 1', [language, id], function(tx,
           results)
        {
          success(results.rows.item(0));
        });
      },
      ...
    },
  },
},
```

S shranjevanjem iste logike na enem mestu, smo poleg preglednosti dosegli tudi manjšo velikost končne aplikacije.

Podpora večjezičnosti

V aplikacijo smo že od samega začetka vgradili sistem za večjezičnost. Ta deluje na preprostem sistemu, kjer imamo v datoteki za vsak jezik shranjene prevedene fraze v formatu *JSON* po pravilu ključ-prevod 6.6.

Primer kode 6.6: Primer datoteke ki vsebuje prevode po sistemu ključ:prevod

```
app.languages["en"] = {
  'title': 'Erasmus Ljubljana',
  'settings': 'Settings',
  ...
}
```

Dodatno smo razvili vtičnih za knjižnico *jQuery*, ki se kliče pri vzpostavitvi strani znotraj mobilne aplikacije. Primer je prikazan na primeru kode 6.7.

Primer kode 6.7: Primer klica metode za prevod strani

```
$(document).on('pageinit', function(event)
{
  var page = $(event.target);
  page.translate(app.languages[app.language]);
});
```

V prezentacijskem delu smo v *HTML* datotekah definirali elemente, ki jih želimo prevesti. Te smo morali podati v ustreznem formatu {{ *ključ* }}. Primer lahko vidimo v primeru kode 6.8.

Primer kode 6.8: Primer podane in prevedene HTML kode

```
// provided code
<div>{{ title }}</div>
// translated code
<div>Erasmus Ljubljana</div>
```

Ločevanje strani

Kot je bilo omenjeno že v prejšnjih poglavjih, smo se držali pravila, da poskušamo logiko mobilne aplikacije čimbolj razdeliti. Ena od logik je tudi ločevanje strani v *js/routers.js* in *js/controllers.js*. To vključuje mehanizem, ki zazna, ali smo prešli na neko drugo spletno stran znotraj aplikacije ter izvede ustrezno metodo. Katero metodo mora mehanizem izvesti smo določili v krmilniku. Primer določitve je prikazano v primeru kode 6.9.

Primer kode 6.9: Primer določitve metode za izvedbo ob navigaciji na določeno stran

```
this.router = new $.mobile.Router([
{
  "/views/faculties.html":
  {
    handler: app.controllers.FacultiesController,
    events: "c",
    argsre: true
  }
},
...
```

Razlog za takšen pristop je, da črpamo podatke iz podatkovne baze. To pomeni, da ko preidemo na neko stran npr. predstavitev fakultete, moramo iz podatkovne baze naložiti ustrezne podatke ter jih prikazati. Logiko za pridobitev podatkov smo za vsako stran definirali v svoji metodi. Primer je prikazan v primeru kode 6.10.

Primer kode 6.10: Primer metode za pridobitev vseh fakultet iz podatkovne baze in njihov prikaz

```
app.controllers.FacultiesController = function(page)
{
  app.orm.faculties.find.all(app.language, function(faculties)
  {
```

```
    // process data and render view
  });
};
```

Takšna logika je zelo prisotna pri različnih ogrodjih, ki temeljijo na pristopu *MVC*. Pristop *MVC* predlaga ločevanja treh slojev: podatkovni sloj, kontrolni sloj in predstavitveni sloj. Tudi mi sledimo temu pristopu in ravno z ločevanjem strani smo realizirali kontrolni sloj.

Osveževanje podatkov

Mobilna aplikacija vsebuje mehanizem, ki je definiran v *js/api.js*, za osveževanje podatkov. Mehanizem je sestavljen iz 3 sklopov:

- preverjanje ali je na voljo nova verzija podatkov;
- prenos nove verzija podatkov;
- shranjevanje podatkov v lokalno podatkovno bazo.

Prvi sklop vzpostavi povezavo z vmesnikom spletne administracije, kjer pridobi čas zadnje spremembe podatkov. Čas primerja z lokalno shranjenim časom, ki ga je aplikacija shranila pri zadnjem osveževanju podatkov. Primer je prikazan v primeru kode 6.11. Če je potreba po osvežitvi podatkov, se sproži drugi sklop.

Primer kode 6.11: Primer metode preverjanje obstoja nove verzije podatkov

```
update: function(update, latest)
{
  $.getJSON(this.url + 'lastupdate/', function(data)
  {
    var lastupdate = parseInt(data.objects[0].updated_at);
    if (app.lastupdate == null || app.lastupdate < lastupdate)
    {
      // new version of data is available
```

```
      update(lastupdate);
    }
    else
    {
      // we already have the latest version of data
      latest();
    }

  });
},
```

Drugi sklop osveževanja podatkov prenese vse podatke s spleta v mobilno aplikacijo. Podatki so shranjeni v formatu *JSON*. Ker podatki vsebujejo tudi povezave do slik, mora mehanizem vso slikovno gradivo prenesti in shraniti v lokalno mapo.

Programski jezik JavaScript omogoča asinhrono izvajanje programske kode. Naš mehanizem uporablja asinhrone klice za prenos slik. Problem se pojavi, ko želimo prenesti seznam slik, vendar ne vemo, kdaj so vse slike prenešene in se lahko začne naslednji sklop. To lahko rešimo z ugnezdenjem klicev metod, kot je prikazano v primeru 6.13. Problem se pojavi, ko imamo veliko metod, ker hitro izgubimo na preglednosti.

Primer kode 6.12: Primer ugnezdenega izvajanja metod

```
// methodX(callback)
method1(function()
{
  method2(function()
  {
    method3(function()
    {
      // successfully executed all methods
    }
```

```
  }
}
```

V našem primeru smo uporabili dodatno knjižnico *async.js*, ki med drugim omogoča sinhrono izvajanje kode.

Primer kode 6.13: Primer sinhronega izvajanje kode z async.js

```
var series = [];
series.push(function(callback)
{
  method1(function()
  {
    callback(null);
  });
});
series.push(function(callback)
{
  method2(function()
  {
    callback(null);
  });
});
series.push(function(callback)
{
  method3(function()
  {
    callback(null);
  });
});
async.series(series, function()
{
  // successfully executed all methods
});
```

Tako smo dosegli, da prvo iz *JSON* podatkov preberemo katere vse slike moramo prenesti. Nato iteriramo preko seznama ter gradimo nove metode za prenos. Na koncu, ko se vse metode izvedejo oz. ko uspešno prenesemo vse slike, lahko preidemo na naslednjo fazo, ki vključuje shranjevanje podatkov v podatkovno bazo.

6.2.3 Prezentacijska logika

Prezentacijska logika vključuje vse *HTML* datoteke oz. poglede (*views*), ki skupaj s stili določajo samo obliko mobilne aplikacije. Poleg tega logika vsebuje tudi mehanizme, ki omogočajo prenos dinamičnih podatkov v poglede. Večino podatkov naložimo znotraj metod, ki so definirane v *js/controllers.js*. Podatki so v obliki *JavaScript* objektov.

Razvili smo vtičnik za *jQuery*, ki nam pomaga za lažje določanje vsebine pozameznik pogledov. Vtičnik je prikazan v primeru kode 6.17.

Primer kode 6.14: Vtičnik za lažje delo z dinamičnimi podatki in pogledi

```
/**
* jQuery Mobile Render
* Copyright (c) 2013 Erol Merdanovic
* https://github.com/FrEaKmAn/jquery-mobile-render
*/
(function($)
{
  $.fn.render = function(data)
  {
    return this.each(function()
    {
      $.each(data, function(path, value)
      {
        var attribute = null;
        if (path.match(/.*\[(\w+)\]/gi))
        {
```

```
        attribute = path.replace(/.*\[(\w+)\]/gi, '$1');
        path = path.replace(/(.*)\[\w+\]/gi, '$1');
      }

      var element = $(this).find(path);
      if (element.get(0) !== undefined)
      {
        if (attribute !== null)
        {
          element.attr(attribute, value);
        }
        else
        {
          switch (element.get(0).tagName.toLowerCase())
          {
            case 'img':
              element.attr('src', value);
              break;
            default:
              element.html(value);
          }
        }
      }
    }.bind(this));
  });
};
})(jQuery);
```

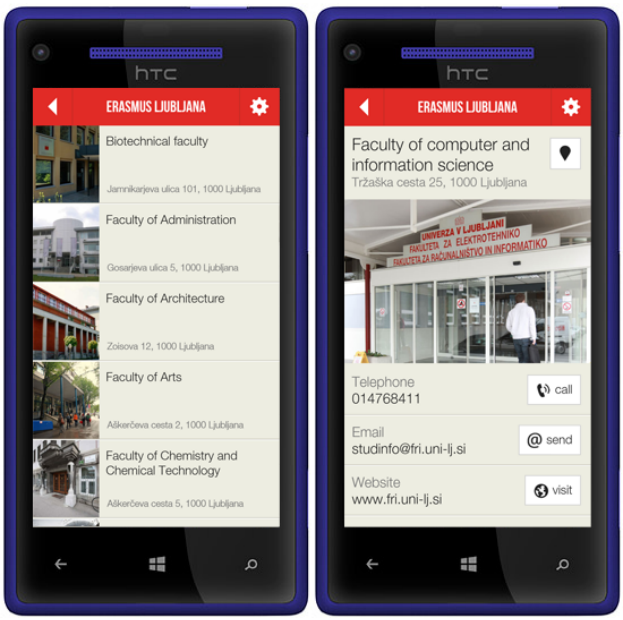
Primer uporabe vtičnika je prikazan v primeru kode 6.15. V sliki

Primer kode 6.15: Primer uporabe vtičnika za lažje delo z dinamičnimi podatki in pogledi

```
var page = $('#page');
page.render({
```

```
  '.title': 'Faculty of Computer and Information Science',
  'a.website[href]': 'http://www.fri.uni-lj.si/'
});
```

Tak pristop smo uporabili v celotni mobilni aplikaciji. Na sliki 6.2.3 je prikazan seznam vseh fakultet, ki smo jih naložili iz podatkovne baze. Hkrati ob kliku na fakulteto v seznamu dobimo dodatne podrobnosti. Podrobnosti vsebujejo ime in lokacijo, sliko, osnovni opis ter kontaktne informacije. Aplikacija se zaveda, kateri podatki predstavljajo katero informacijo, zato jih zna ustrezno prikazati in zraven ponuditi gumb, ki izvede ustrezno akcijo. Za primer vzemimo gumb *call* pri telefonski številki, ki nas ob kliku na njega avtomatsko preusmeri v aplikacijo za klicanje in pri tem telefonsko številko avtomatsko določi.



Slika 6.2: Prikaz uporabniškega vmesnika za prikaz vseh fakultet ter podrobnosti izbrane fakultete.

Mapa

Za mapo smo uporabili Leaflet ter OpenStreetMap. Sama inicializacija mape je preprosta, potrebujemo *HTML* element, ki mu določimo atribut *id*, po katerem ga lahko referenciramo. Nato po navodilih, ki so na voljo v dokumentacija ustvarimo mape in jim določimo center ter faktor povečave. Zatem določimo iz katere mape naj pobira slike formata *PNG*, ki predstavljajo posamezne dele mape.

Primer kode 6.16: Primer vzpostavitve mape z nastavitvami

```
app.map = L.map('map', { zoomControl: true }).setView([46.0526,
    14.50745], 12);

app.tiles = L.tileLayer('tiles/{z}/{x}/{y}.png', {
        maxZoom: 17,
        minZoom: 12,
        errorTileUrl: 'tiles/blank.png',
        attribution: '&copy; <a
            href="http://osm.org/copyright">OpenStreetMap</a>
            contributors'
}).addTo(app.map);
```

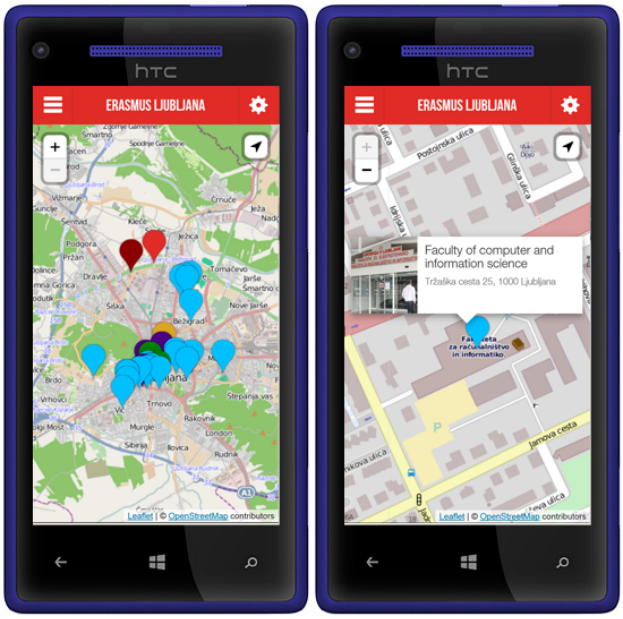
Po potrebi lahko dodajamo tudi elemente na mapo. V našem primeru smo dodajali lokacije ustanov.

Primer kode 6.17: Primer dodajanja lokacije na mapo

```
var icon = L.icon({ iconUrl: 'img/icons/map_pin_faculty.png',
    iconSize: [45, 45], iconAnchor: [22, 45] });
var marker = new L.Marker([46.0529, 14.50783], { icon: icon });
marker.addTo(app.map);
```

Za vsak tip lokacije (fakulteta, veleposlaništvo, knjižnica, študentski dom al druga javna ustanova) smo uporabi svojo ikono, ki predstavlja lokacijo. Tako smo dosegli, da je večja preglednost in lažja uporaba. Na sliki 6.2.3

lahko vidimo, kako lokacije izgledajo na mapi. Ko kliknemo na izbrano lokacijo, dobimo okno, ki nam podaja dodatne informacije.



Slika 6.3: Prikaz lokacij na mapi ter podrobnosti o vsaki lokaciji.

Stranska plošča

Stranska plošča oz. *side panel*, ki je definirana v *js/panels.js* je namenjena, da uporabnik hitro dostopa do vseh lokacij. Poleg lokacij, stranska plošča vsebuje tudi povezavo do ostalih informacij kot so informacije o Univerzi v Ljubljani, o mestu Ljubljani in seznamu uporabnih nasvetov.

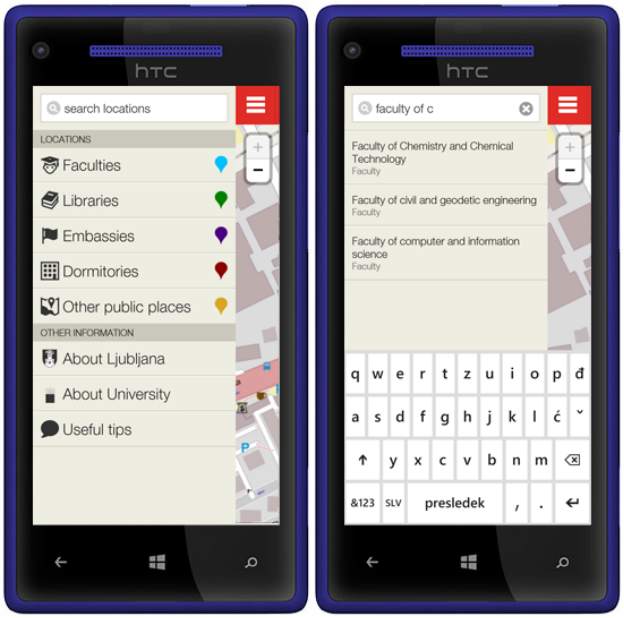
Na vrhu je vnosno polje, ki je namenjeno za iskanje po lokacijah. Ko uporabnik vpisuje iskalni niz npr. ime fakultete, so mu izpisujejo vsi možni rezultati. Mehanizem za delovanje prvo pridobi podatke iz podatkovne baze in jih shrani. Ob uporabi preverja, ali ime lokacije vsebuje iskalni niz ter v primeru ujemanja, lokacijo doda v seznam rezultatov. Koda mehanizma je prikazana v primeru kode 6.18. Delovanje mehanizma je prikazano na sliki 6.2.3.

Primer kode 6.18: Koda mehanizma za iskanje po lokacijah

```
find: function(query, success)
{
  var results = [];
  // create search regex, which is global and case-insensitive
  var re = new RegExp(query, 'gi');

  $.each(app.search.items, function(i, item)
  {
    if (item['name'].search(re) > -1)
    {
      results.push(item);
    }
  });

  success(results);
}
```



Slika 6.4: Prikaz stranske plošče z vsemi informaciji in med iskanjem lokacije.

Nastavitve

Nastavitve so del aplikacije, kjer uporabnik lahko:

- spreminja jezik (v trenutni verziji sta na voljo Slovenščina in Angleščina);
- omogoča in onemogoča *GPS* delovanje;
- preverja, če je na voljo nova verzija podatkov;
- prebere informacije o mobilni aplikaciji;
- pridobi kontaktni elektronski naslov za sporočanje napak ali idej.

6.3 Testiranje mobilne aplikacije

Testiranje mobilne aplikacije smo izvedli ročno. Zavedamo se, da obstajajo načini in knjižnice za avtomatizirano preverjanje delovanja aplikacije, vendar smo bili mnenja, da v trenutno fazi je ročno preverjanje zadovoljivo. Testiranje je vključevalo testiranje samega delovanja aplikacije na različnih platformah in napravah. Pri tem smo ugotovili par napak, ki so posledica različnih verzij brskalnikov znotraj platform in njihovih pravil prikazovanje *HTML* elementov. Napake smo tudi odpravili.

Poleg testiranja pravilnega delovanja mobilne aplikacije smo testirali tudi uporabniško izkušnjo za zahteve in scenarije, ki smo jih določili v prejšnjih poglavjih. Fokusirali smo se na pravilo, da mora biti dostop do informacij čimbolj enostaven. Hkrati morajo biti določene informacije vizualno predstavljene. Z mapami mislimo, da smo to zelo uspešno dosegli. Poleg tega smo lahko dostopali do večine informacij za minimalnim številom klikov, kar je z vidika uporabniške izkušnje zelo uspešno.

Poglavje 7

Sklepne ugotovitve

Tehnologije, ki smo jih uporabili, so se izkazale za odlično izbiro. Ogrodje *PhoneGap* odlično služi za ustvarjanje hibridnih aplikacij. Sama zasnova nam omogoča, da po potrebi lahko ogrodje razširimo z vtičniki, kar se je izkazalo za ključni faktor pri izbiri podatkovne baze *SQLite*. Tudi za razvoj čelnega dela je *jQuery Mobile* opravil svoje delo. Zavedati se moramo, da je to ogrodje, ki ga uporablja ogromno ljudi in bo podobno tudi v prihodnosti, zato lahko pričakujemo nadaljni razvoj in nove uporabne funkcionalnosti. Z lokalnim shranjevanjem map smo dosegli, da uporabniki ne potrebujejo internetne povezave, vendar še vedno lahko enostavno preverijo lokacijo izbrane ustanove. To se nam zdi pomembna dodatna vrednost k mobilni aplikaciji Erasmus Ljubljana, ki izboljša uporabnost in priročnost uporabe aplikacije pri vsakodnevnih nalogah.

Slabosti razvoja hibridnih aplikacij vidimo predvsem v delovanju na vseh platformah. Neka platforma ima lahko odličen in napreden razvoj, vendar če ostale platforme ne slediju tempu, potem PhoneGap in ostala ogrodja ne morejo implementirati novosti. Poleg tega določene platforme namerno onemogočajo razvoj hibridnih aplikacij in se samo fokusirajo na razvoj domorodnih aplikacij. Vse to vodi k počasnejšemu napredku.

Druga poglavitna slabost hibridnih aplikacij je hitrost. Pri testiranju uporabniškega vmesnika je bilo opazno počasnejše delovanje. Hkrati po po-

govoru z različnimi ljudmi, ki se ukvarjajo z razvojem mobilnih aplikacij, je bilo opazno, da je to velik problem. Programski jezik *JavaScript* je jezik prihodnosti. Izvaja se znotraj brskalnika, ki se razlikuje od platforme do platforme. Obstajajo standardi, ki določajo, katere vse funkcionalnosti bi morali današnji brskalniki podpirati. Nažalost se vsi ne držijo teh standardov. Ampak lahko pričakujemo, da z novimi verzija mobilnih naprav z večjo procesorsko močjo in naprednejšimi brskalniki bodo to težave preteklosti. Hkrati je *HTML5* odlično orodje za ustvarjanje iger, vendar še nekaj časa zaradi slabih performančnih zmogljivosti ne bo mogoče ustvarjati procesorsko zahtevnih hibridnih iger za mobilne naprave. Glede na trg mobilnih iger, se to lahko v kratkem spremeni.

Proces ugotavljanja zahtev in želja bodočih uporabnikov oz. študentov nam je omogočil, da smo spoznali tematiko Erasmus izmenjav ter težave študentov. Mislimo, da bo mobilna aplikacija Erasmus Ljubljana izpolnila pričakovanja. Sam potek ustvarjanja aplikacije smo jasno definirali, zato nismo imeli večjih težav pri samem razvoju. Hkrati smo vse podatke pridobili na spletu ter v primeru nejasnosti so nam bili v pomoč uradne osebe v posameznih ustanovah.

Ker želimo, da bi podobni projekti zrastli tudi v drugih mestih, smo se odločili, da vso kodo, ki smo jo razvili, ponudimo javnosti kot odprtokodni projekt. Upamo, da bodo ostali uporabili kodo, jo prilagodili svojim željam ter ustvarili mobilne aplikacije za svoja mesta. Pomembno je, da obstajajo mobilne aplikacije, ki olajšajo študentov na izmenjavi lažje pridobivanje informacij in posledično naredijo študij bolj vesel in uspešen. Ravno to je naš cilj.

Literatura

[1] John M. Wargo, *PhoneGap Essentials: Building Cross-platform Mobile Apps*. Addison-Wesley, 2012.

[2] Michael J. Young, *JavaScript: Just the Basics - A Primer for the Complete Beginner*. Self Reliant Press, 2012.

[3] Jon Reid, *jQuery Mobile*. O'Reilly Media, 2011.

[4] Mark Lassoff, *Javascript for Beginners*. LearnToProgram Incorporated, 2013.

[5] Dr. E. Balagurusamy, *Programming In C#*. Tata McGraw-Hill Education, 2008.

[6] Marty Alchin, *Pro Django*. Apress, 2013.

[7] (2013) Ljubljanska univerza ostaja med 500 najboljšimi univerzami na svetu. Dostopno na:
http://www.dnevnik.si/slovenija/v-ospredju/ljubljanska-univerza-ostaja-med-500-najboljsimi-univerzami-na-svetu

[8] (2013) Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC. Dostopno na:
http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[9] (2013) Phonegap. Dostopno na:
http://www.phonegap.com

[10] (2013) PhoneGap podprte platforme. Dostopno na:
http://phonegap.com/about/feature/

[11] (2013) Programski jezik Java. Dostopno na:
http://www.java.com/

[12] (2013) Programski jezik Objective C. Dostopno na:
http://en.wikipedia.org/wiki/Objective-C

[13] (2013) jQuery Mobile. Dostopno na:
http://jquerymobile.com/

[14] (2013) Zepto. Dostopno na:
http://zeptojs.com/

[15] (2013) Sencha Touch. Dostopno na:
http://www.sencha.com/products/touch/

[16] (2013) Google Maps. Dostopno na:
http://maps.google.com/

[17] (2013) Bing Maps. Dostopno na:
http://www.bing.com/maps/

[18] (2013) OpenStreetMap. Dostopno na:
http://www.openstreetmap.org/

[19] (2013) Leaflet. Dostopno na:
http://www.leafletjs.com/

[20] (2013) Eclipse. Dostopno na:
http://www.eclipse.org/

[21] (2013) Xcode. Dostopno na:
https://developer.apple.com/xcode/

[22] (2013) Visual Studio. Dostopno na:
http://www.microsoft.com/visualstudio/

[23] (2013) HTML5. Dostopno na:
http://www.w3.org/TR/html5/

[24] (2013) HTML5.1. Dostopno na:
http://www.w3.org/TR/html51/

[25] (2013) CSS3. Dostopno na:
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

[26] (2013) JSON. Dostopno na:
http://www.json.org

[27] (2013) Web Storage Database. Dostopno na:
http://dev.w3.org/html5/webstorage/

[28] (2013) Web SQL Database. Dostopno na:
http://dev.w3.org/html5/webdatabase/

[29] (2013) Indexed Database. Dostopno na:
http://www.w3.org/TR/IndexedDB/

[30] (2013) SQLite. Dostopno na:
http://www.sqlite.org/

[31] (2013) Usage of JavaScript libraries for websites. Dostopno na:
http://w3techs.com/technologies/overview/javascript_library/all