UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Razvijalska ogrodja za podporo večim mobilnim platformam

(Developer frameworks for mobile cross-platform programming)

Ime in priimek: Saša Nikolić
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Branko Kavšek
Somentor: dr. Jernej Vičič

Koper, september 2013

Ključna dokumentacijska informacija

Ime in PRIIMEK: Saša NIKOLIĆ

Naslov zaključne naloge: Razvijalska ogrodja za podporo večim mobilnim platformam

Kraj: Koper

Leto: 2013

Število listov: 49 Število slik: 10 Število tabel: 0

Število prilog: 3 Število strani prilog: 11 Število referenc: 17

Mentor: doc. dr. Branko Kavšek

Somentor: dr. Jernej Vičič

UDK:

Ključne besede: razvijalska ogrodja, PhoneGap, Android, kamera, detekcija gibanja

Algoritem: odštevanje ozadja, primerjanje slikovnih točk

Izvleček:

Zaključna naloga predstavlja metodologije razvijanja mobilnih aplikacij. To področje je zelo obširno, zato so na začetku predstavljeni le splošni načini programiranja, ki omogočajo bralcu vpogled v svet mobilne tehnologije. V nadaljevanju so predstavljena preizkušena ogrodja za razvoj mobilnih aplikacij za več platform. Opisana so ogrodja PhoneGap, Appcelerator Titanium, RhoMobile Suite in Corona SDK. Izbira teh je temeljila na trenutni popularnosti, različnem načinu delovanja in raznolikosti v načinu razvijanja aplikacije. Natančneje je opisano ogrodje PhoneGap, ki je trenutno eno najbolj priljubljenih na tem področju. Izdelano aplikacijo v tem ogrodju smo primerjali z izvorno aplikacijo Android na podlagi časovne in strojne potratnosti. Obe aplikaciji omogočata identično funkcionalnost - detekcijo gibanja. Medtem ko aplikacija Android deluje popolnoma zadovoljivo, novo nastala aplikacija ne zadošča željenim potrebam. Težave, nastale ob razvoju le-te, so opisane v poglavju 4.2.

Key words documentation

Name and SURNAME: Saša NIKOLIĆ

Title of final project paper: Developer frameworks for mobile cross-platform programming

Place: Koper

Year: 2013

Num. of pages: 49 Num. of figures: 10 Num. of tables: 0

Num. of appendices: 3 Num. of appendix pages: 11 Num. of references: 17

Mentor: doc. dr. Branko Kavšek

Co-Mentor: dr. Jernej Vičič

UDK:

Keywords: development frameworks, PhoneGap, Android, camera, motion detection

Algorithm: background substraction, comparing pixels

Abstract:

The final project paper presents development strategies for mobile applications. Because this field is very extensive, only general methods of programming are presented at the beginning. These provide an easier look to the insight of the world of mobile technology. Some well-known Cross-platform frameworks are presented in the next chapters. In particular, PhoneGap, Appcelerator Titanium, RhoMobile Suite and Corona SDK usage is described. These frameworks were chosen because of their popularity, different ways of functionality and different programming languages. PhoneGap framework is specifically described because it is one of the easiest to use and probably the most popular at the moment. A PhoneGap application was created and compared with a native Android application. The comparison was based on the time and resource consumption. Both applications have the same goal - providing motion detection. While the native Android application runs sufficiently good, the PhoneGap application does not meet the starting requirements. Problems, that occured in the development process of making it, are described in chapter 4.2.

Zahvala

Zahvaljujem se mentorju, doc. dr. Branku Kavšku in somentorju, dr. Jerneju Vičiču, za strokovno pomoč ter usmerjanje pri izdelavi zaključne projektne naloge.

Zahvalil bi se tudi družini in prijateljem za pomoč pri iskanju slovničnih in tiskarskih napak.

Kazalo

Seznam slik

Seznam prilog

Seznam kratic

itd.	in tako dalje
npr.	na primer
Dr.	doktor
OS	operacijski sistem
SDK	Software Development Kit, orodja za razvoj programske opreme
HTML	HyperText Markup Language, označevalni jezik
GPS	Global Positioning System, sistem globalnega določanja položaja
PHP	Personal Home Page Tools, odprtokodni programski jezik
CSS	Cascading Style Sheets, slogovni jezik
XML	Extensible Markup Language, označevalni jezik
IDE	Interactive Development Environment, integrirano razvojo okolje, ki olajša programiranje
MBaaS	Mobile Backend As A Service, model za povezavo aplikacije s strežnikom "v oblaku"
MVC	Model–View–Controller, vrsta programske arhitekture

1 Uvod

Danes, ko se je informacijska tehnologija že zelo močno razvila, si življenja brez uporabe pametnih mobilnih naprav praktično niti ne moremo predstavljati. Z njimi lahko komuniciramo med seboj, poslušamo glasbo, zajemamo najzanimivejše trenutke s pomočjo vgrajene kamere in jih delimo z ostalimi, se povezujemo na splet, beremo novice, itd.

Začetki mobilnih naprav segajo v leto 1973. Tedaj sta tedanji šef Motorole John F. Mitchell in raziskovalec Dr. Martin Cooper predstavila prvi prenosni telefon, ki je bil težek dober kilogram in je bil za današnje pojme povsem primitiven, saj je bilo možno z njim opraviti le telefonski klic. Več v novici [10]. Še pred nekaj leti je bila večina mobilnih naprav "neumnih". Zmožne so bile le pisanja in prikazovanja kratkih sporočil ter elektronske pošte. Nekatere so bile celo opremljene s spletnim brskalnikom, ki pa je bil zmožen prikazati le enostavno besedilo in slike. Na začetku so take naprave pogaljali operacijski sistemi Symbian (2000), BlackBerry OS (2002) in Windows Mobile (2002). Ampak danes, trideset let kasneje, se je mobilna tehnologija razvila do te mere, da ne moremo preživeti dneva brez uporabe pametnih telefonov ali tabličnih računalnikov.

Leto 2007 je bilo leto velikih sprememb. S prihodom na trg prvega Applovega izdelka, pametnega telefona z imenom iPhone z nameščenim operacijskim sistemom iOS. V nasprotju s konkurenco, je ta baziral predvsem na aplikacijah. Ostali operacijski sistemi niso podpirali večopravilnosti aplikacij, saj so zajemali le navadna opravila in komunikacijo. Tedaj je bilo veliko pozornosti mobilnih razvijalcev posvečeno iPhone SDK - ogrodju s knjižnicami za implementacijo izvornih (native) iPhone aplikacij. Poleg tega pa je Apple ponudil razvijalcem še možnost razvijanja spletnih aplikacij znotraj orodja "Dashcode", ki je del iPhone SDK-ja. Kako to deluje, je opisano v prispevku [9]. To bila vzpobuda vsem razvijalcem spletnih aplikacij, da se preusmerijo v razvijanje aplikacij za pametne telefone. Leto kasneje so ostali proizvajalci množično pričeli s prodajo telefonov z Googlovim operacijskim sistemom Android in so sčasoma prevzeli več kot 50 % tržišča mobilnih naprav. Priljubljen je zaradi odprtokodnosti, enostavne prilagoditve uporabniškega vmesnika in hitrega razvoja. Obema pa je skupna ena točka, ki ju je popeljala do tolikšne priljubljenosti: spletna trgovina z aplikacijami. Na eni

strani imamo Apple App Store, ki prinaša ogromne zaslužke - 3.8 milijona evrov na dan, na drugi pa Google Play, ki ponuja boljše iskanje in vsebuje veliko več zabavnih aplikacij - 12 % več kot App Store. Natančnejši podatki so lepo grafično prikazani v [8].

Leta 2010 se je v bitko za večji tržni delež vrnil Microsoft z novim operacijskim sistemom poimenovanim Windows Phone. Ta je primarno ciljal na tržišče uporabnikov v nasprotju s predhodnimi različicami, saj je bil bolj podjetno uravnan. Dve leti za tem ga je nadomestil Windows Phone 8. Opremili so ga s povsem novim grafičnim vmesnikom poimenovanim Metro in s specifičnimi Windows aplikacijami kot so Microsoft SkyDrive, Office ter Internet Explorer. Po mobilnih napravah s tem operacijskim sistemom je znana predvsem finska Nokia.

Poleg zgoraj omenjenih na tržišče prodira tudi obetavni in povsem zanimiv operacijski sistem imenovan Tizen. Ta je voden pod okriljem Intla in Samsunga, njegovi glavni značaji pa so odprtokodnost, Linux kernel in dejstvo, da orodja za razvoj aplikacij temeljijo na JavaScript knjižnicah kot so jQuery ter jQuery Mobile. Zanimivost tega operacijskega sistema je, da lahko Android aplikacije povsem normalno delujejo tudi na tej platformi.

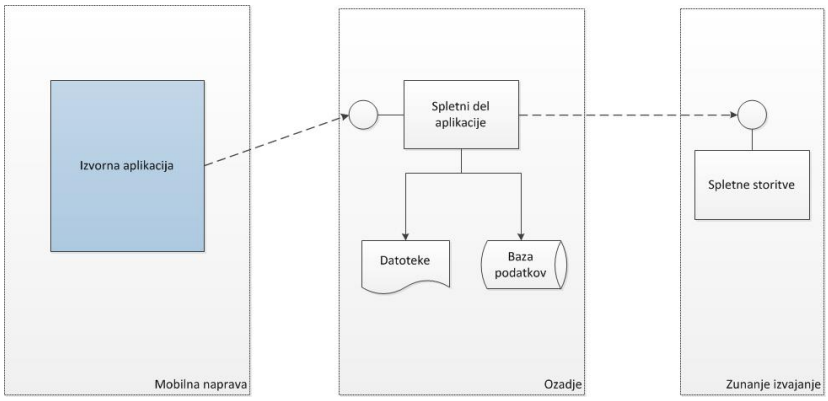
Vidimo lahko, da se je sčasoma razvilo veliko operacijskih sistemov. To predstavlja predvsem problem za razvijalce mobilnih aplikacij, saj se je zares težko odločiti katero platformo podpreti. Zaključna naloga predstavlja rešitve, s katerimi je možno programiranje aplikacij znotraj enega samega ogrodja za distribucijo na več mobilnih platfom. Na začetku so predstavljeni splošni oprijemi pri razvoju aplikacij. Opisano je programiranje izvornih aplikacij, strežniških ter uporabniških spletnih aplikacij, hibridnih in interpretiranih aplikacij. Sledeče poglavje zajema preizkušena razvijalska ogrodja, kot so PhoneGap, Appcelerator Titanium, RhoMobile Suite in Corona SDK. Ogrodje PhoneGap je v poglavju 3.2 natančneje opisano, saj je bilo izbrano za najprimernejšo rešitev pri razvoju željene aplikacije. Zadnje poglavje predstavlja primerjavo med izvorno aplikacijo za Android operacijski sistem ter aplikacijo, razvito v razvijalskem ogrodju PhoneGap.

2 Motivacija

Glede na veliko število operacijskih sistemov na trgu se je sčasoma pojavila zamisel za razvoj enotne aplikacije. Ta bi delovala na vseh operacijskih sistemih, zato je postala zelo privlačen cilj. Tako je danes nastalo več različnih razvijalskih ogrodij, s katerimi je zastavljeni cilj možno doseči. Večina jih temelji na spletnih standardih - HTML in Javascript programskem jeziku, zato je programiranje za večino ljudi olajšano [1]. Toda sčasoma je prevladovalo mnenje, da so te aplikacije mnogo počasnejše v primerjavi z izvornimi aplikacijami. Ta problem je v tej zaključni projektni nalogi raziskan in podrobneje opisan. Do jasnega odgovora smo prišli tako, da smo primerjali dve aplikaciji, eno napisano v izvorni kodi za operacijski sistem Android, drugo pa v ogrodju, ki je namenjeno distribuciji aplikacije v tem trenutku najpopularnejšim platformam. Aplikacijo MotionSensor sva s sošolcem Matevžem Črnilogarjem leta 2012 implementirala v sklopu tekmovanja Primatijada 2012, na kateri sva zasedla drugo mesto in še v sklopu konference ERK istega leta [4]. Z njo lahko zaznavamo gibanje v prostoru s pomočjo vgrajene kamere na mobilnih napravah, ki jih poganja Android OS. Kot cilj zaključne naloge sem si zadal dejstvo, da se iz raziskovanja in preizkušanja novih mobilnih ogrodij za razvoj aplikacij naučim čim več novih programskih jezikov in metodologij programiranja, predvsem pa to, da bi izdelal končni produkt namenjen večim mobilnim platformam. Novo nastalo aplikacijo sem primerjal na podlagi hitrosti delovanja z zgoraj omenjeno, že obstoječo aplikacijo.

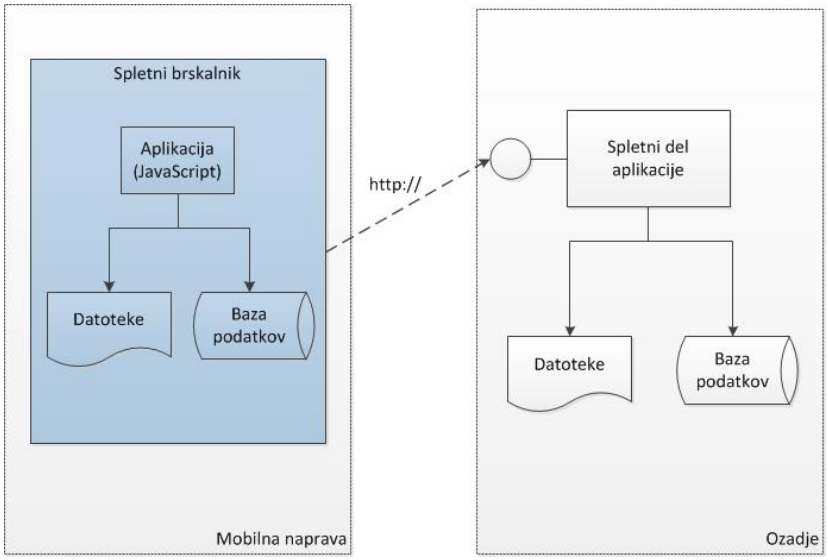
3 Metodologija

Programiranja mobilnih aplikacij se je možno lotiti na več načinov [6]. Najboljši način za kompleksnejše aplikacije je razvoj izvornih aplikacij za vsak operacijski sistem posebej, saj je tako možno popolnoma izkoristiti vse razpoložljive vire kot so procesor, notranji pomnilnik, kamera, GPS. Te aplikacije delujejo tako, da se na točno določeni napravi izvaja grafični vmesnik aplikacije in le nekaj programske logike, ves glavni del pa se izvede v ozadju ali celo na strežniku, od koder je mogoče hitro pridobivati ter pošiljati podatke. Zaradi predhodno vgrajenih funkcij v mobilnih napravah je samo delovanje zelo hitro. Če je potreba po aplikaciji taka, da je grafično zahtevna, zmogljiva in vsem dostopna na spletnih trgovinah za aplikacije, je to prava rešitev. V nasprotnem primeru so druge alternative boljše, saj je proces razvoja takih aplikacij časovno potraten in posledično drag.



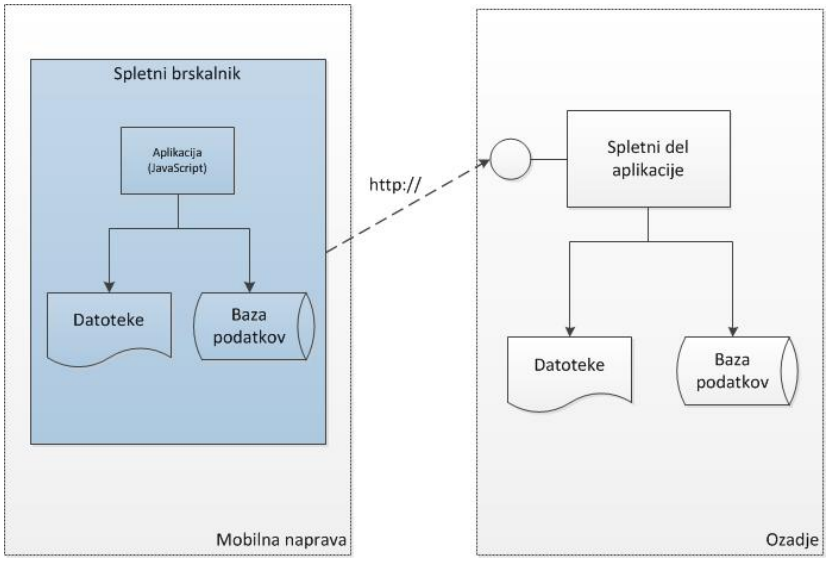
Slika 3.1: Izvorne aplikacije

Če potrebujemo le predstavitveno aplikacijo, ki bi delovala na vseh operacijskih sistemih, jo lahko enostavno implementiramo kot spletno stran in odpremo v mobilnih brskalnikih. To lahko naredimo v ogrodjih kot so Zurb Foundation in Twitter Bootstrap. Ti delujejo popolnoma samostojno na strežniku, le zaženemo jih preko brskalnika na napravi. S to metodo pa se aplikacij, ki bi uporabljale senzorje pametnih telefonov, ne da implementirati.



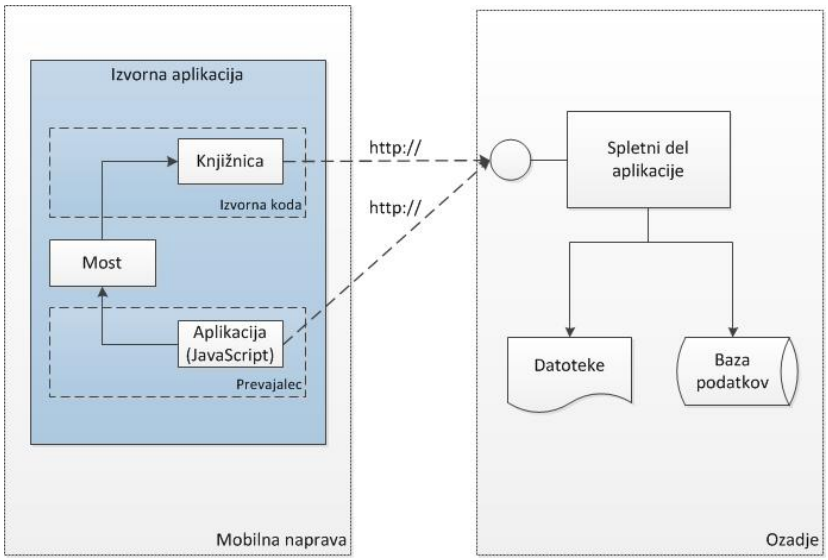
Slika 3.2: Server-side spletne aplikacije

Podobna zgornji metodi je tudi implementacija spletne strani, ki ne temelji na strežniškem delu, temveč na napravi. Ta način je priporočljiv, ko je cilj narediti prototip določene storitve oz. ko se potrebuje nekaj, kar izgleda kot aplikacija, vendar ne daje veliko pomena senzorjem. V brskalniku se požene grafični vmesnik v programskem jeziku Javascript spisane aplikacije, v ozadju pa se izvaja glavni del, spisan v Javi ali PHP-ju. Take aplikacije je moč implementirati v sledečih ogrodjih: jQuery mobile, iUI.js, zepto.js, joApp, Sencha, Wink Toolkit in jQTouch.



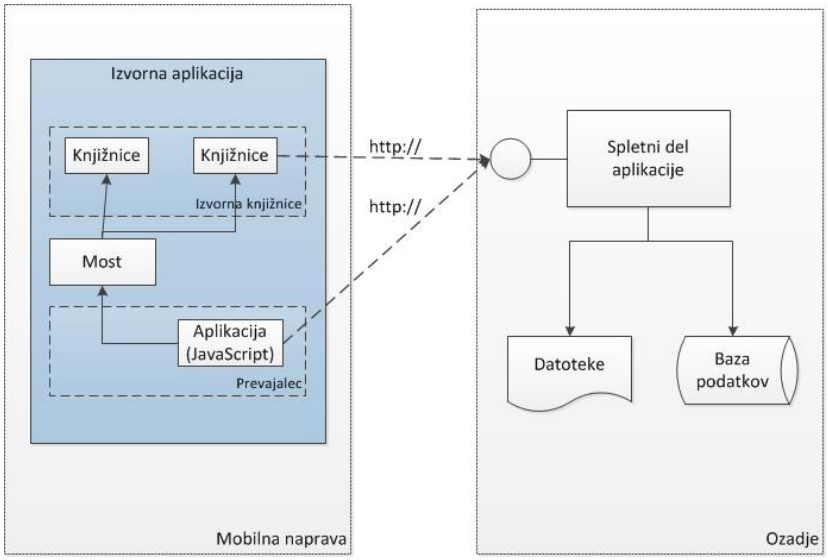
Slika 3.3: Client-side spletne aplikacije

Hibridni pristop je najbolj priporočljiv pri aplikacijah, ki jih je potrebno razviti v čim krajšem možnem času in hkrati zahtevajo uporabo strojne opreme mobilnih naprav. Ta deluje tako na podlagi spletnih programskih jezikov, v večini primerov HTML5, CSS in JavaScript, kot tudi v izvorni kodi. Hibridne aplikacije se izvajajo znotraj izvornega okolja, hkrati pa uporabljajo gradnik spletnega brskalnika za obdelavo HTML-ja in procesiranje JavaScript kode na sami napravi. Abstraktni sloj oz. most med spletno aplikacijo in izvorno kodo omogoča dostop do strojne opreme, katere ni mogoče uporabiti le s spletnimi aplikcijami. Primer ogrodja, s katerim je možno take aplikacije razviti, je Phonegap. Več informacij o tem pristopu programiranja je mogoče prebrati v [12].



Slika 3.4: Hibridne aplikacije

Mobilne aplikacije pa lahko tudi implementiramo na t.i. interpretirani način. Take aplikacije uporabljajo specifične lastnosti grafičnih vmesnikov različnih operacijskih sistemov, zato izgledajo podobno kot izvorne aplikacije, medtem ko pa programska logika deluje enako, ne glede na platformo. Implementirana je lahko z XML klici ali pa s kakšnim drugim opisnim jezikom. Ogrodja kot so MonoTouch (.NET) in Rhodes (Ruby) predstavljajo popolna okolja znanih programskih jezikov, v katerih lahko razvijamo takšne aplikacije. Med najboljše v tej kategoriji nedvomno sodi Appcelerator Titnium, ki omogoča implementacijo s pomočjo knjižnic, spisanih v JavaScriptu.



Slika 3.5: Interpretirane aplikacije

Aplikaciji delujeta z enakim ciljem, saj želimo dobiti čim bolj natančno predstavo o odzivnosti. Ideja implementacije detektorja gibanja s pomočjo vgrajene kamere na mobilnih napravah se zdi mamljiva in predvsem poučna, saj je za njeno učinkovito delovanje potrebno dostopati do kamere ter preučiti optimalnejše algoritme za zaznavo sprememb. Četudi je teh aplikacij kar nekaj, se njihove implementacije in nameni precej razlikujejo. Uporabljeni algoritem je v poglavju 4.1 tudi predstavljen.

Zaradi hitrega porasta števila naprav s prednameščenim Android OS in njihove cenovne dostopnosti smo se odločili za implementacijo aplikacije za ta operacijski sistem. Na drugi strani pa smo preizkusili večinoma popularna razvijalska ogrodja za podporo večim platformam, ki so opisana v nadaljevanju. Odločili smo se za ogrodje z imenom PhoneGap zaradi dobro spisanih knjižnic, izredno hitrega razvoja in popularnosti. Zelo pomembna prednost tega ogrodja je tudi ta, da je povsem zastonj za distribucijo ene aplikacije. Obe aplikaciji sta spisani v programskem okolju Eclipse, zaradi enostavnosti in odlične podpore izbranim programskim jezikom.

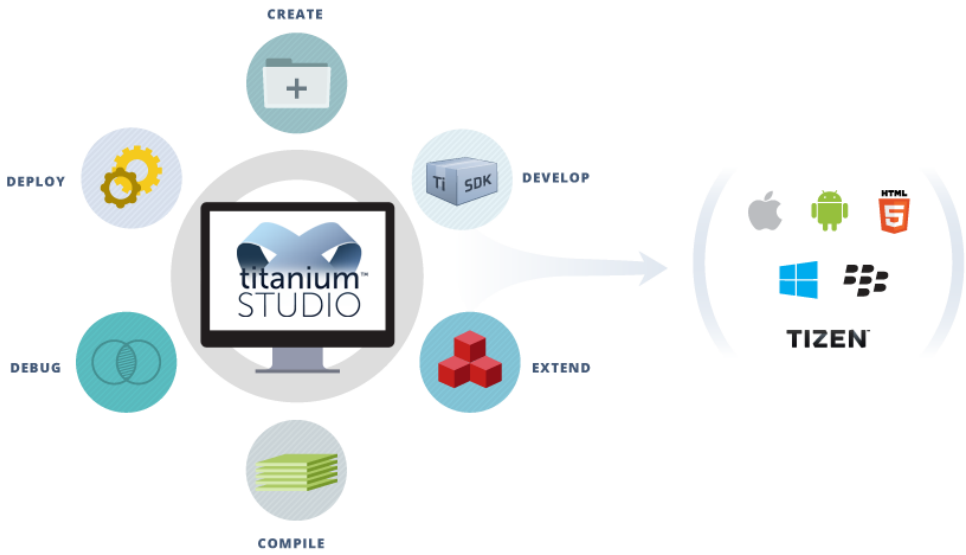
3.1 Preizkušena ogrodja

Za preprostejšo odločitev pri izbiri razvijalskega ogrodja za razvoj aplikacije na več mobilnih platform je potrebno poznati čim več alternativ. Potrebno je preučiti njihovo delovanje, način implementacije in predvsem njihove prednosti ter slabosti. V tem poglavju so predstavljena tri razvijalska ogrodja, ki se razlikujejo tako v sami arhitekturi ogrodja kot tudi v načinu razvijanja same aplikacije.

3.1.1 Appcelerator Titanium

Appcelerator Titanium, ogrodje za razvoj aplikacij za telefone, tablične računalnike in namizne računalnike, poznamo že od leta 2006. Razvija ga podjetje Appcelerator Inc., ki ima sedež v Silicijevi dolini v ZDA. Ogrodje trenutno podpira najpopularnejše operacijske sisteme kot so iOS, Android in Tizen. Podpora za Blackberry OS je še v začetni fazi, za Windows 8 pa naj bi izšla že proti koncu leta 2013. Da bi z njim razvili aplikacijo, je potrebno poznati programske jezike spletnih aplikacij (HTML, Javascript), posebej pa se je še potrebno naučiti knjižnice (Titnium API [15]), ki se bistveno razlikujejo od običajnih. Ogromno število dobro spisanih knjižnic je možno uporabiti s posebnim programom, imenovanim Titanium Studio, ali pa preko programa Eclipse, kjer pa je potrebno namestiti še posebno integrirano razvojno okolje (IDE). Titanium Studio poenostavlja proces razvoja mobilnih aplikacij, tako da razvijalcem omogoča hiter razvoj, testiranje in objavo končnih produktov na več različnih sistemih hkrati. Z integriranim sistemom storitev v oblaku (MBaaS), sistemom takojšnjega razhroščevanja in zelo uporabnim ogrodjem imenovanim Alloy se Titanium Studio uvršča med najbolj celovite ter hkrati enostavne mobilne platforme za razvoj mobilnih aplikacij. Appcelerator Alloy razvija odprtokodni projekt imenovan Alloy za Titanium. To ogrodje temelji na programski arhitekturi, ki je znana pod imenom Model-view Controller (MVC). Pomembno je zato, ker zagotavlja preprost model za ločevanje uporabiškega vmesnika aplikacije s poslovno logiko in podatkovnimi modeli. S tem lahko enostavno prihranimo veliko dragocenega časa in kode pri razvoju, zaradi česar je kasnejše branje kode ter njena ponovna uporaba v drugih projektih bistveno lažja. Tradicionalna mobilna razvojna ogrodja ne omogočajo tako enostavnega ločevanja, zato me je sčasoma Titanium Studio ob uporabi zelo navdušil. Tako je možno pri programiranju izbrati bolj stukturiran pristop, kar se izkaže za veliko prednost, predvsem pri bolj kompleksnih projektih.

Po tem, ko je programska koda aplikacije spisana, sledi več korakov prevajanja. Sprva Titanium prevede spisano Javascript kodo in jo pomanjša. Nato se zgradi odvisnostna hierarhija vseh uporabljenih knjižnic. Pri dejanskem prevajanju kode se ustvarijo datoteke in v njih specifični deli kode, ki so potrebni, da aplikacija deluje na željenem operacijskem sistemu. Tako dobimo izvorno aplikacijo, katero je potrebno prevesti, za kar poskrbi kar sam Titanium tako, da pokliče specifično orodje za prevajanje (npr. xcodebuild za aplikacije, ki delujejo na iOS). Na iOS je JavaScript koda prevedena v Base64 binarne simbole, na Android OS pa v kodo, predstavljeno v bitih. Spisana koda torej ni nikoli prevedena v programske jezike Objective-C ali Java. V tem je tudi razlika med tem ogrodjem in aplikacijami, spisanimi za točno določen operacijski sistem. Če povzamem, Titanium uporablja dodatno plast procesiranja pri pripravi aplikacije v primerjavi z izvornimi aplikacijami.



Slika 3.6: Življenski cikel razvoja aplikacije v programskem okolju Titanium Studio [16]

Lahko je opaziti, da vso programsko kodo, ki jo uporabimo v programu, ogrodje interpretira kot izvorno kodo za določen operacijski sistem. Tako so pojavna okna, gumbi, stikala, menijske vrstice, animacije, itd. povsem enaka, kot če bi jih spisali v Objective-C ali Javi. Razvijalci imajo tudi skoraj popoln dostop do vseh spisanih knjižnic, ki obsegajo kamero, video, audio, kontaktni imenik, geolokacijo, datotečni sistem in še mnogo drugih. Zaradi uporabe Javascript kode za programsko logiko in njenega prevajanja naj bi bila zmogljivost aplikacije kanček slabša od izvornih, vse ostalo pa je enako.

3.1.1.1 Slabosti

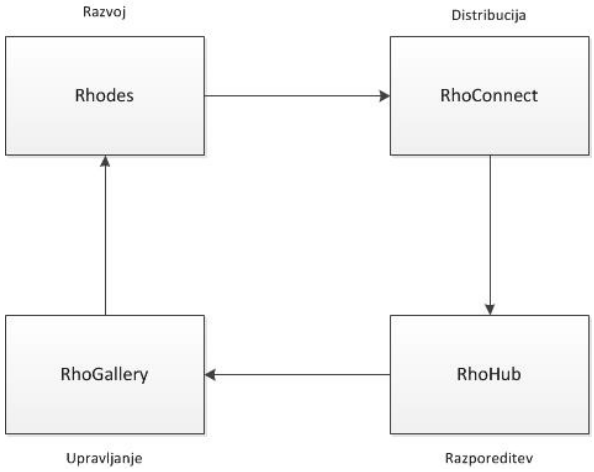
Glede na to, da so naprave z prednameščenim operacijskim sistemom Blackberry in Windows Phone 8 dandanes precej priljubljene, je velika škoda, da se trenutno podpira le iOS, Android ter Tizen. Druga negativna stvar je njihov poslovni oz. tržni model, ki ni ravno prijazen do mladih razvijalcev. Da bi razvili aplikacijo v tem ogrodju, se je potrebno registrirati. Tu imamo na voljo tri pakete: Titanium Community Edition, ki je zastonj, Appcelerator Platform (Public Cloud Enterprise Edition), za katerega je potrebno plačati preračunanih 743 € na mesec in Appcelerator Platform (Virtual Private Cloud Enterprise Edition), ki je najdražji in stane preračunanih 1986 € na mesec[1]. Zadnja dva paketa sta namenjena predvsem podjetjem, ponujajo pa dodatne funkcionalnosti kot so pregledovalnik kode, več prostora za štoritve v oblaku", spremljanje uspešnosti aplikacije, itd. Kot zadnje pa velja omeniti še njihovo zelo počasno asistenco. Ti za odgovore potrebujejo kar nekaj dni, ali pa navadnim uporabnikom celo sploh ne odgovorijo [7].

3.1.2 RhoMobile Suite

Rhomobile Inc. je podjetje, ki je danes pod okriljem ameriške Motorole. Ustanovljeno je bilo leta 2008, z namenom zagotavljanja računalniške programske opreme, s katero je moč implementirati sodobne mobilne aplikcije. Leta 2009 je bilo podjetje razglašeno na priznani konferenci Interop kot najboljše zagonsko (startup) podjetje leta. Njihova platforma za razvoj aplikacij, RhoMobile Suite, se od ostalih razlikuje po tem, da ne temelji le na spletnih programskih jezikih, temveč je v uporabi tudi Ruby [14]. Trenutno podpira vse popularnejše operacijske sisteme: iOS, Windows Mobile, Android, Symbian in BlackBerry. To ogrodje je precej obsežno, saj ponuja odlične poslovne rešitve preko svojih štirih glavnih izdelkov kot so Rhodes, RhoConnect, Rhohub in RhoGallery [11]. Rhodes je odprtokodno ogrodje za razvoj aplikacij za pametne telefone. Te so popolnoma izvorne aplikcije, s katerimi je mogoče uporabiti vsak del strojne opreme na sodobnejših napravah. Ogrodje je zelo podobno popularnemu Rails ogrodju. Tako kot Appcelerator Titanium, tudi to ogrodje temelji na strojni arhitekturi Model-view Controller (MVC), kar bistveno olajša programiranje. RhoConnect je sinhronizacijsko ogrodje sestavljeno iz dveh delov – odjemalca na napravi in strežnika, ki deluje s pomočjo Ruby-ja. Glavna naloga tega je, da skrbi za pravočasnost pri distribuciji podatkov znotraj aplikacije. Informacije se shranjujejo lokalno na vsaki napravi, tako da

[1]Cene so preračunane po trenutnih tečajih na spletni menjalnici Banke Koper, dne 29.8.2013.

ta deluje tudi tedaj, ko ni mogoče vzpostaviti povezave s spletom. RhoConnect server povezuje aplikacije v ozadju tako, da sledi večim napravam in pregleduje, katere podatke potrebujejo ter jih samo pošilja preko izvornih knjižnic. RhoHub je gostiteljsko razvojno okolje za Rhodes in RhoConnect. Aplikacija se prenese na strežnik Heroku, kjer za grafični vmesnik poskrbi RhoHub. Poleg tega še bistveno pripomore pri skupinskem sodelovanju ter poteku dela znotraj projekta, saj zagotavlja podporo sistemu za sledenje verzijam Git, ki je prav temu namenjena. RhoGallery, kot zadnji del zbirke ogrodij, omogoča upravljanje zbirke aplikacij v skupine, kamor se lahko uporabnike povabi. Aplikacije, kasneje tudi posodobitve, so namenjene vsaki skupini posebej in se tako enostavno namestijo na njihovo napravo. RhoGallery je ena izmed prvih rešitev za upravljanje mobilnih aplikacij preko spleta (na določeni domeni).



Slika 3.7: Rhomobile okolje je sestavljeno iz več delov: Rhodes, RhoConnect, RhoHub in RhoGallery.

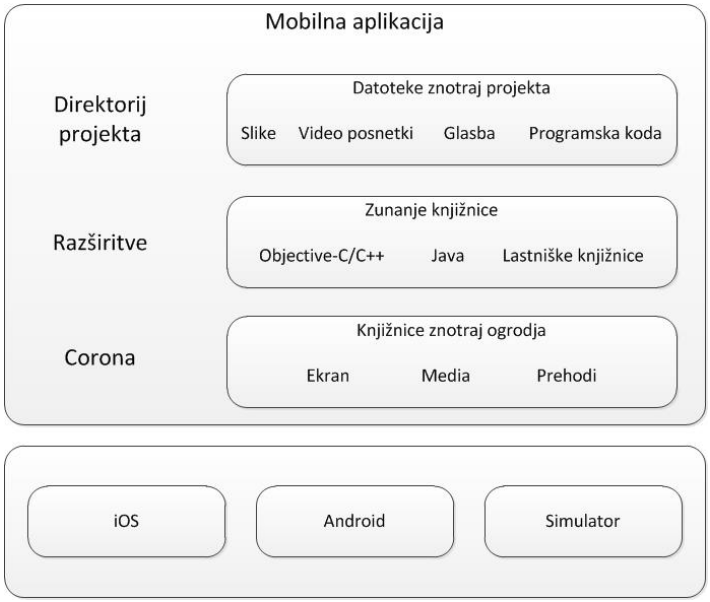
3.1.2.1 Slabosti

Trditve o tem, da RhoMobile deluje povsem enako na vseh platformah, le deloma držijo. V primeru, da je podatkovna baza velika in želimo kompleksnejšo aplikacijo z veliko podstranmi in da je bila aplikacija na začetku spisana za Android OS ali iOS, obstaja velika verjetnost, da taka aplikacija ne bo delovala na Windows Mobile oz. BlackBerry OS. Za prilagoditev takih aplikacij prvemu operacijskemu sistemu bi bilo potrebno preurediti kodo grafičnega vmesnika, da bi pa delovala na BlackBerry OS, pa bi bilo potrebno popraviti kodo zaradi težav s pomnilnikom. Sledeča negativna lastnost je ta, da sem za nastavitve delovnega okolja za prevajanje aplikacij na lokalnem sistemu za

vse operacijske sisteme porabil nekaj dni, kar se zdi še nesprejemljivo. Toda to napako se da odpraviti z uporabo zgoraj omenjenega okolja v oblaku, poimenovanega RhoHub. Kot zadnje pa velja omeniti, da je ogrodje Rhodes plačljivo za podjetja, ki želijo svoje aplikacije tržiti, in sicer znašajo dobrih 500 na aplikacijo. Cene RhoConnect ogrodja so odvisne od načrtovanega števila uporabnikov, ki naj bi se povezali na strežnik.

3.1.3 Corona SDK

Corona SDK je razvijalsko ogrodje mobilnih aplikacij, pri čemer posebnost je ta, da se za razvoj aplikacije uporablja skriptni jezik Lua in programska jezika C++/Open GL. Razvilo ga je ameriško podjetje Corona Labs Inc. konec leta 2009, s sedežem v Silicijevi dolini [2]. Sprva je bilo namenjeno zgolj Applovim pametnim telefonom iPhone, leto kasneje pa še napravam z Android operacijskim sistemom. Cilj Corone je drastično izboljšati razvoj grafično-zahtevnih aplikacij, zato odločitev o uporabi Lue. Ta je namreč eden hitrejših skriptnih jezikov. Dokumentacija je tudi odlično spisana in povsem razumljiva [3]. Corona SDK se od ostalih razvijalskih ogrodij precej razlikuje. Za programiranje se ne uporablja programskega okolja s sistemom za razhroščevanje kot npr. Eclipse, temveč običajni urejevalnik besedila in grafični urejevalnik za izdelavo slik. Arhitektura ogrodja je enostavna in sestavljena iz treh delov. Programski del aplikacije se nahaja v skupnem direktoriju. V njem tako najdemo slike ali video posnetke, ki jih želimo uporabiti v aplikaciji, glasbene datoteke in programsko kodo. Ta je spisana v eni sami datoteki imenovani main.lua. Od tu lahko direktno kličemo knjižnice implementirane v samem ogrodju in s skriptnim jezikom Lua ali pa zunanje, ki temeljijo na ostalih programskih jezikih. Ob zagonu programa Corona Simulator se izvede prevedba in zagon naše aplikacije.



Slika 3.8: Arhitektura Corone SDK

3.1.3.1 Slabosti

To ogrodje je odlično, če je potreba po razvoju mobilne igre. Navdušuje predvsem enostavnost uporabe in zmogljivost uporabljenega skriptnega jezika. Ni pa vse zlato, kar se sveti. Corona namreč temelji na visokoravenskih funkcijah za grafično delo. Medtem ko to drastično zmanjša kompleksnost kode, zmanjšuje tudi fleksibilnost ogrodja. Če knjižnice ne zagotavljajo funkcij, ki jih aplikacija potrebuje, postane pisanje kode v Lui časovno in performančno potratnejše. Zaprtokodnost ogrodja je po mojem mnenju naslednja pomanjkljivost. S tem se odpravljanje napak znotraj ogrodja odvija mnogo počasneje, kot če bi razvijalci imeli vpogled v notranjost ogrodja in prispevali svoje popravke. Enako kot pri zgoraj opisanih ogrodjih je tudi to plačljivo za aplikacije, ki jih želimo objaviti in z njimi tržiti.

3.2 Izbrano ogrodje

Med vsemi testiranimi ogrodji je PhoneGap najpreprostejše za uporabo ter med razvijalci na tem področju najbolj popularno. V nadaljevanju je natančneje opisana zgodovina razvoja tega ogrodja, njena arhitektura ter način delovanja.

3.2.1 PhoneGap

PhoneGap je zmogljivo, odprtokodno ogrodje za razvoj mobilnih aplikacij na večih napravah z uporabo standardnih spletnih tehnologij kot so HTML, CSS in JavaScript. Trenutno obstaja veliko število spisanih knjižnic, začrtani pa so tudi jasni načrti za prihodnost. PhoneGap trenutno podpira naslednje operacijske sisteme: iOS, Android, webOS, Windows phone 7 in 8, Symbian, BlackBerry OS, Bada ter TIzen. V kratkem pa se bodo temu seznamu pridružili še Firefox OS ter Ubuntu. S tem, ko je Nokia javno naznanila preusmeritev iz Symbiana na Windows Phone, obstaja verjetnost, da bo v prihodnosti PhoneGap opustil podporo le-temu.

3.2.1.1 Zgodovina

Razvoj PhoneGapa se je začel leta 2008 na iPhoneDevCampu s strani podjetja Nitobi, s ciljem enostavno in učinkovito razviti apikacijo ter jo razdeliti na čim več različnih naprav [17]. Projekt se je razvil z ekipo razvijalcev, ki so delo opravljali ob koncu tedna, da bi ustvarili okostje začrtane platforme – osnovno funkcionalnost in izvorno aplikcijsko okolje, potrebno za zagon spletnih aplikacij na napravah iPhone. Takoj zatem, ko jim je to uspelo, so se lotili podpore za operacijske sisteme Android in BlackBerry. Sledila so različna priznanja za vložen trud. Leta 2009 je ogrodje zmagalo na tekmovanju Web 2.0 Expo Launch-Pad v kategoriji izbira občinstva. Čez čas je podpora prišla še za ostale strojne platforme, s tem pa tudi optimizirane knjižnice za zagotavljanje enakega delovanja na vsaki platformi posebej.

IBM se je kasneje vključil v projekt in pričel z uporabo Eclipsa kot programskega okolja za razvoj aplikacij. Adobe je leta 2011 uradno napovedal nakup podjetja Nitobi Software. Takoj za tem je bila programska koda ogrodja licencirana s strani Apache Software Foundation. Sprva so ime projekta spremenili v Apache Callback, danes pa je

znano kot Apache Cordova. S tem so razvijalci dosegli odprto skrbništvo do projekta, kar pomeni, da bo tudi v prihodnosti za uporabnike ostalo odprtokodno, stabilno in zastonj. Razvojna ekipa, ki je delala na projektu PhoneGap v svojem prostem času, se je tako znašla v položaju, ko lahko na tem projektu delajo s polno paro in ga tako izjemno hitro razvijajo. To se kaže v tem, da je za prihodnost objavljen dobro izoblikovan načrt, ki je javno dostopen vsem uporabnikom. Nova verzija ogrodja je namreč razvita in je na voljo razvijalcem vsak sledeči mesec.

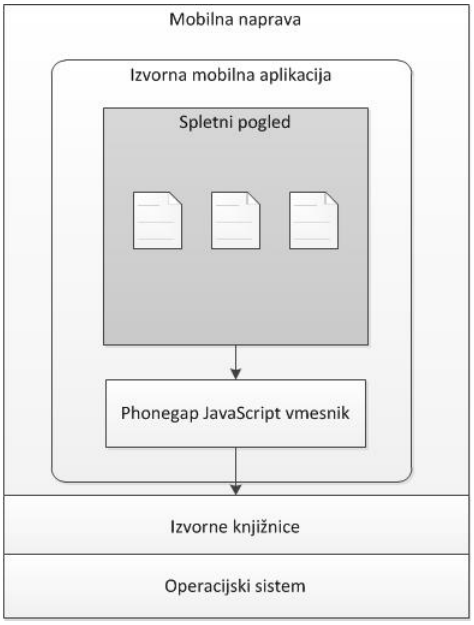
Zgodnje verzije PhoneGapa so za razvoj iOS aplikacije zahtevale uporabo Applovih računalnikov, za Windows Mobile aplikacije pa je bilo potrebno imeti računalnik z nameščenim Windows operacijskim sistemom. Po letu 2012 se je to spremenilo s prihodom storitve "PhoneGap Build", ki omogoča prenos spisane programske kode na spletni strežnik, ki generira aplikacijo namenjeno vsem mobilnim platformam.

3.2.1.2 Način delovanja

Kot že zgoraj omenjeno, PhoneGap omogoča uporabo spletnih programskih jezikov za razvoj mobilnih aplikacij tako za pametne telefone kot za tablične računalnike. Razvijalec razvije spletno aplikcijo, katero s posebnimi orodji, ki jih ponuja PhoneGap, prevede v izvorno aplikacijo za vsak podprt mobilni operacijski sistem. Grafični vmesnik znotraj izvorne aplikacije je praktično sestavljen iz enega samega okna, ki vsebuje t.i. spletni pogled. Ta poskrbi za prevajanje spletne vsebine, napisane v spletnem programskem jeziku (npr. HTML). Ko se aplikacija zažene, se sprva vanj naloži začetna stran, tipično zapisana v datoteki z imenom index.html. Nato je možna interakcija med uporabnikom in aplikacijo. Ob tem se lahko nove vsebine nalagajo s pomočjo povezav ali Javascript kodo. Vsebina aplikacije je lahko zapakirana znotraj aplikacije same ali pa se nahaja na spletnem strežniku, od koder se prenese preko spletne povezave. Izgled aplikacije določimo z velikostjo pisave, črtami, razmaki in barvami. Implementacija teh nastavitev je preprosta z uporabo programskega jezika CSS. Praktično vse, kar lahko razvijalec naredi v spletni aplikaciji, lahko naredi tudi znotraj PhoneGapa.

Toda spletni brskalniki na mobilnih napravah nimajo dostopa do ostale strojne opreme na napravi kot so npr. kamera, kompas ali mikrofon. Da bi lahko naredili uporabno aplikacijo, je potrebno dostopati do teh komponent zunaj spletnega okolja. Zato PhoneGap zagotavlja zbirko JavaScript knjižnic, ki to omogočajo. Ko uporabnik znotraj aplikacije pokliče knjižnico PhoneGapa s pomočjo JavaScripta, se ta v posebnem sloju prevede v primerno izvorno knjižnico za določeno funkcijo. To je razvidno v

naslednji sliki (Slika 3.9).



Slika 3.9: Interakcija znotraj PhoneGap aplikacije

Razlike med dostopi na različnih operacijskimih sistemih je lahko razbrati v naslednjem primeru, kjer je obravnavan dostop do kamere. Za zajemanje slike s pomočjo Phonegapa, bi JavaScript koda izgledala tako:

```
navigator.camera.getPicture( onSuccess, onFail );
```

Aplikacija s parametri poda imena dveh funkcij, ki se izvedeta, ko se je zajemanje slike končalo (onSuccess) oz. ko je prišlo do napake (onFail). Na Android operacijskem sistemu bi se ob izvedbi te kode prevedla v naslednjo:

```
navigator.camera.getPicture( onSuccess, onFail );
```

Na BlackBerryju OS bi koda, ki se izvaja v ozadju, bila videti tako:

```
Player player = Manager.createPlayer("capture://video");
player.realize();
player.start();
VideoControl vc = (VideoControl) player.getControl("VideoControl");
```

```
viewFinder = (Field)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE,
             "net.rim.device.api.ui.Field");
scrnMain.add(viewFinder);
vc.setDisplayFullScreen(true);
String imageType = "encoding=jpeg\&width=1024\&height=768\&quality=fine";
byte[] theImageBytes = vc.getSnapshot(imageType);
Bitmap image = Bitmap.createBitmapFromBytes(imageBytes, 0, imageBytes.length, 5);
BitmapField bitmapField = new BitmapField();
bitmapField.setBitmap(image);
scrnMain.add(bitmapField);
```

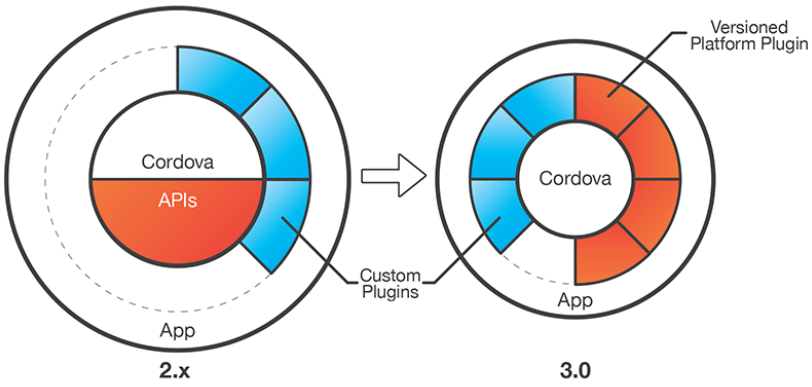
Na iOSu pa bi koda bila takšna:

```
UIImagePickerController *imgPckr = [[UIImagePickerController alloc] init];
imgPckr.sourceType = UIImagePickerControllerSourceTypeCamera;
imgPckr.delegate = self;
imgPckr.allowsImageEditing = NO;
[self presentModalViewController:imgPckr animated:YES];
```

Lahko je opaziti, kako PhoneGap poenostavi razvijanje aplikacij za več mobilnih platform hkrati. Razvijalec tako z enim samim klicem knjižnice dobi želeno storitev takoj, brez dodatnega znanja vseh ostalih programskih jezikov, ki bi jih potreboval, če bi aplikacijo delal za vsak operacijski sistem posebej.

Phonegap trenutno zajema knjižnice, ki podpirajo merilnik pospeška, kamero, zajemanje fotografij, glasbe in video posnetkov, kompas, razne povezave, delo s kontakti, podatke o napravi, dogodke, datoteke, geolokacijo, funkcije v zvezi z globalizacijo, brskalnik, obdelavo glasbenih datotek, obvestila, pozdravno okno ter možnosti za shranjevanje podatkov [13].

Kot pri vseh razvojnih orodjih, dejanska funkcionalnost, ki jo ta podpira, ni dovolj. V teh primerih pridejo prav PhoneGap vtičniki, ki razširjajo osnovno funkcionalnost. Zaradi dobre funkcionalnosti ogrodja je popularnost precej hitro narasla. To je pripomoglo k temu, da ima danes PhoneGap zelo aktivno skupnost razvijalcev. Ko uporabnik vidi luknjo v ogrodju, se kaj hitro ta luknja odpravi s pomočjo vtičnikov. S prihodom nove verzije, Adobe PhoneGap 3.0, se je arhitektura implementacije drastično spremenila, predvsem z novo implementacijo le-teh (Slika 3.10). Vtičnike je tako danes dosti lažje namestiti in odstraniti kot kadarkoli prej. Da bi to dosegli, potrebujemo le en sam ukaz.



Slika 3.10: Primerjava arhitekture med PhoneGap 2.x in PhoneGap 3.0

Ne glede na to, ali spletna aplikacija uporablja PhoneGap knjižnice ali ne, jo je potrebno ob koncu zapakirati v izvorno aplikacijo, ki bo delovala na napravi. To lahko dosežemo z orodjem Adobe PhoneGap Build, ki temelji na prevajanju aplikacij "v oblaku". Za delovanje le-tega je potrebna konfiguracija ene same datoteke, ki ji pravimo config.xml. Kot že samo ime pove, so v njej zapisane nastavitve aplikacije. Konfiguracijska datoteka in datoteke s spletno vsebino (HTML, CSS in JavaScript datoteke) se nato prenesejo na Adobe PhoneGap Build strežnik, kjer se prevedejo v izvorne aplikacije za vsak podprt operacijski sistem. Do te programske opreme lahko razvijalec enostavno dostopa preko spletnega brskalnika. Velja omeniti, da je prevajanje zastonj za eno aplikacijo, če ne želimo deliti kode z ostalimi uporabniki na spletu. Za prevedbo 25 aplikacij pa je cena preračunanih 7.44 € na mesec[2]. V nasprotnem primeru, torej če delimo kodo z ostalimi uporabniki, je uporaba zastonj. Za webOS in Symbian platforme po prevajanju dobimo binarno datoteko, ki je pripravljena na distribucijo. Za Android, iOS in BlackBerry pa je potrebno predložiti pravilne certifikate oz. t.i. podpisovalne ključe, da bi omogočili objavo na spletnih trgovinah z aplikacijami.

3.2.1.3 Slabosti

Negativne strani PhoneGapa je težje opredeliti. Za ljudi, ki se v tehnologijo ne razumejo najbolje in za vse ostale, je potrebno začeti z najbolj opazno omejitvijo. Vse mobilne platforme se nenehno spreminjajo in razvijajo. Podjetja, ki stojijo za njimi, imajo svoje interese in vizije prihodnosti, katerim se mora PhoneGap nenehno prilagajati. HTML in Javascript podpora za te platforme se lahko sproti precej spreminja,

[2]Cene so preračunane po trenutnih tečajih na spletni menjalnici Banke Koper, dne 29.8.2013.

kar je lahko nadležno za razvijalce. Ti morajo poskrbeti za razne patente, tržne in pravne stvari ter vse ostalo za uporabo šifriranja v svojih aplikacijah. Drugi problem je potratnost aplikacije. Mogoče je opaziti, da aplikacije zasedajo in porabijo precej več pomnilnika kot izvorne aplikacije. Razlog je v samem razvoju aplikacije, saj razvijalec nima direktnega dostopa do pomnilnika. Preveliko zasedanje pomnilnika hkrati tudi pomeni prekinitev delovanja nekaterih ostalih procesov, kar je lahko za nekatere moteče, saj je ob vnovičnem zagonu le-teh potrebno počakati, da se aplikacija naloži nazaj v pomnilnik.

4 Rezultati

4.1 Izvorna aplikacija Android

Cilj aplikacije je torej detekcija gibanja s pomočjo vgrajene kamere. Kot pri vseh tovrstnih rešitvah, je potrebno avtomatično zajemanje slik ob določenem časovnem intervalu, zajeti kader pa mora biti vedno enak. Sprememba gibanja se določi na podlgi primerjanja zadnjih dveh nastalih slik in iskanja različnih točk med njima. Pri tem je potrebno paziti na porabo procesorja in delovnega spomina, saj nimajo vse naprave z Android operacijskim sistemom enake strojne specifikacije. Pomembno je, da zajete slike aplikacija primerja med seboj v realnem času. Ključno je tudi poskrbeti za čim manjšo porabo baterije, saj se hitro izprazni. Detektor v tem primeru več ne deluje. Implementran je algoritem iz [5] z dodano novo funkcionalnostjo. Ko je gibanje v prostoru zaznano, aplikacija pošlje elektronsko sporočilo, v kateri je zapisan trenutni položaj, kjer se naprava nahaja. Priložena je slika, kjer je razvidno, kaj se v prostoru dogaja.

Potreben pogoj je, da naprava ob zajemanju slike miruje. S tem omogočimo statično ozadje in uporabo algoritma, ki deluje po metodi odštevanja ozadja (ang. *background substraction*). Algoritem poskrbi za primerjavo statične slike ozadja (ang. *background*) s kasneje zajetimi slikami in za iskanje razlik med njima (ang *foreground*). Preverjanje poteka na osnovi svetilnosti v posameznih točkah in ne na barvi, kot je običajno. V primeru slabše svetlobe v prostoru, to pripomore k učinkovitejšemu delovanju . Problem nastane, če se v ozadju dogajajo majhne spremembe, kot so npr. premikanje zavese, oblakov ali sonca. Da teh sprememb ne bi upoštevali, je potrebno uvesti mejno vrednost pri pregledovanju razlik, ki je poljubno nastavljiva v začetnem oknu.

Aplikacija potrebuje za svoje delovanje dostop do kamere, lokacije telefona in spleta. Dostope vključimo v datoteki Manifest z značko <*uses-permission ...*>, kjer je parameter željeni del strojne opreme. Za pridobitev dostopa do kamere je bilo potrebno definirati svoje razrede s pomočjo že nekaterih knjižnic, saj v tem primeru zaznavamo gibanje že v predogledu slik in ne v slikah polne ločljivosti. S tem se zmanjša čas,

porabljen za iskanje razlik, saj je velikost slike občutno manjša. Enostaven dostop do spleta je omogočen s specifično značko v Manifest datoteki, za samo sestavo elektronskega sporočila o zaznanem gibanju pa je v uporabi knjižnica JavaMail. Do podatkov o lokaciji pridemo z uporabo knjižnic *LocationManager* in *LocationListener*. Koordinate lahko dobimo tako z uporabo GPS signala kot preko Wi-Fi omrežja. Po pridobitvi vseh dostopov se lahko prične dejansko programiranje.

Prvi korak je pridobitev neobdelanih slik iz kamere. To je implementirano s klicom metode *Camera.PreviewCallback*, ki se izvede vsakič, ko je slika na razpolago. Nato je treba poskrbeti za časovni zamik med zajemanjem slik, da ne bi prišlo do slikanja, medtem ko se zajema druga slika. To je prikazano v sledečem delu kode.

```
if (mMotionDetection.detect(data)) {
    zdaj = System.currentTimeMillis();
    if (zdaj > mReferenceTime + PICTURE_DELAY) {
        mReferenceTime = now + PICTURE_DELAY;
        camera.takePicture(null, null, this);
    }
}
```

V drugem koraku je potrebno poskrbeti za sliko ozadja. Prednost algoritma je v YUV kodiranju barv. YUV format se od standardnega RGB formata razlikuje v prvih dveh bitih, ki vsebuje vrednosti o svetlobi. Predstavljamo si ga lahko kot črno-belo verzijo dobljene slike. Zaradi zaznavanja detekcije na podlagi spremembe v svetlobi, je ta algoritem znatno hitrejši, kot če bi primerjali navadne slike v RGB zapisu.

```
if (Ozadje == null){
    Ozadje = ustvariSliko(data, velikost, vrednostTocke);
}
boolean gibanjeZaznano = false;
slika = ustvariSliko(data, velikost, vrednostTocke);
gibanjeZaznano = slika.jeDrugacna(Ozadje,mejnaVredTocke,mejnaVred);
Ozadje = Slika;
return(gibanjeZaznano);
```

Glavni del algoritma predstavlja primerjava slik glede na svetlobne vrednosti v slikovnih točkah. V tem delu kode se preverja, če so točke znotraj prej definiranih mejnih vrednosti. Če je najdena točka drugačna od prvotne, povečamo števec za ena. Ta števec je bistvenega pomena, saj z njim štejemo število različnih točk. Npr. če imamo sliko velikosti *320 x 200* in mejno vrednost postavljeno na 3 %, bi moral biti števec večji od 1920, da lahko javimo spremembo. Omeniti velja, da sta v uporabi dva tipa mejnih vrednosti – 10 % za vrednosti v posamezni slikovni točki, 3 % pa je dovoljenega odstopanja na celotni sliki.

Algoritem 1: Detekcija gibanja

Vhod: Višina, širina, mejna vrednost, podatki

Izhod: Detekcija gibanja

```
1  public boolean jeDrugacna(Slika other, int pixel_mejnaVrednost, int
   mejnaVrednost)
2  if !assertImage(other) then
3      return false;
4  byte[] ostaliPodatki = other.get(); intskupnoSteviloRazlicnihPixlov = 0;
5  int velikost = Visina * Sirina;
6  for i = 0, ij = 0; i < Visina; i++
7      for j = 0; j < Sirina; j++, ij++
8          intpix = (0xff & ((int) mData[ij])) - 16;
9          intostaliPixli = (0xff & ((int) ostaliPodatki[ij])) - 16;
10         if pix < 0 then
11             pix = 0;
12         if pix > 255 then
13             pix = 255;
14         if ostaliPixli < 0 then
15             ostaliPixli = 0;
16         if ostaliPixli > 255 then
17             ostaliPixli = 255;
18         if Math.abs(pix − ostaliPixli) >= pixel_mejnaVrednost then
19             skupnoSteviloRazlicnihPixlov++;
20 if skupnoSteviloRazlicnihPixlov == 0 then
21     skupnoSteviloRazlicnihPixlov = 1;
       Log.d(TAG, Številorazlicnihpixlov : " + skupnoSteviloRazlicnihPixlov +
       »" + (100/(velikost/skupnoSteviloRazlicnihPixlov)) + "%");
```

Izbrani algoritem zagotavlja dobre rezultate v zaprtih prostorih zaradi delovanja na podlagi svetlobe. Aplikacija je bila testirana na Samsug in HTC napravah ter na dveh različnih verzijah Android operacijskega sistema – 2.3.3 (*Gingerbread*) in 4.1.2 (*Jelly Bean*). Ideja, da bi ob delovanju onemogočili predogled zajemanja slik, se je izkazala za veliki plus, saj tako privarčujemo na dragoceni porabi baterije. To je tudi razlog, da je v času delovanja aplikacije zaslon črne barve. Ob različnih svetlobnih pogojih je aplikacija zadovoljivo reagirala in kar je najpomembnejše, delovala samostojno. Med delovanjem ni bilo opaženih kakršnihkoli prekinitev delovanja.

4.2 Aplikacija PhoneGap

Programiranje z ogrodjem PhoneGap je na začetku projekta izgledalo enostavno, saj je znanje na področju spletne tehnologije razmeroma lahko osvojiti. Po razvoju nekaj enostavnih aplikacij se lahko že lotimo implementiranja željene aplikacije - detektor gibanja. Na spletu česa podobnega ni bilo moč zaslediti, zato je bila želja toliko večja. Na poti smo naleteli na več problemov. Ti so na koncu tega poglavja navedeni.

Pred fazo programiranja je bilo potrebno projekt ustrezno pripraviti. Z novo verzijo Adobe Phonegapa je začetek zelo olajšan, saj je mogoče aplikacijo ustvariti in pognati kar iz standardne Windows konzole. To je moč doseči z vpisom naslednjih treh ukazov. Spodnji primer prikazuje, kako se ustvari nov projekt in se zažene v virtualni napravi z Android operacijskim sistemom.

```
$ phonegap create moja-aplikacija
$ cd moja-aplikacija
$ phonegap run android
```

Prva pognana aplikacija je enostavna, saj na zaslon izpiše le standrdno frazo, ki jo pogosto srečamo v svetu programiranja: "Hello World". Sedaj se lahko v izbranem programskem okolju ustvari projekt iz obstoječe kode in se prične s programiranjem. Sprva je potrebno v datoteki *AndroidManifest.xml* aplikaciji dodeliti pravice za dostop do kamere, zunanjega pomnilnika in spleta. Odločil sem se za podporo vseh verzij Android OS od 2.3.3 dalje, saj so naprave s prejšnjimi verzijami strojno precej manj zmogljive. Dejansko programsko kodo se piše v datoteki *index.html*, ki se nahaja znotraj direktorijev *assets* in *www*. Uporablja se izključno HTML ter JavaScript.

Na začetni strani aplikacije je nameščen gumb, s katerim se požene kamera. Pod gumbom se po zajemanju slik prikažeta še obe dobljeni sliki, ki pa nista vidni. Vstavljene so v t.i. *kanvas* okvirje, zaradi lažjega dela v nadaljevanju. Na vrhu se nahaja vnosno polje, kjer je potrebno določiti mejno vrednost. Z njo se uravnava natančnost preverjanja sprememb med zajetima slikama. To je mogoče enostavno implementirati znotraj *body* značke.

```
<body>
    <form class="threshold">
       Mejna vrednost (%): <input type="text" id="threshold">
    </form>
    <button class="camera" onclick="capturePhoto();">Zajemi sliki</button>
    <div id="pictures" style="display:none;">
     <img id="smallPhoto1" width="100px" height="100px"/>
     <img id="smallPhoto2" width="100px" height="100px"/>
    </div>
    <div id="canvases" style="display: none;">
     <canvas id="myCanvas1" width="100px" height="100px"></canvas>
     <canvas id="myCanvas2" width="100px" height="100px"></canvas>
    </div>
</body>
```

Sedaj je potrebno implementirati funkcionalnost. S funkcijo *capturePhoto()* poženemo kamero. Meja (*limit*) je nastavljena na dva, kar pomeni, da je potrebno zajeti dve sliki. Tu nastopi prvi problem PhoneGapa. Ogrodje še ne ponuja možnosti za avtomatično slikanje, zato je za njeno delovanje potrebna interakcija uporabnika. Poskrbeti je treba, da je dobljen kader v obeh slikah enak, za kar je potrebna mirna roka. Pa tudi to včasih ne zadostuje, saj je dobljena slika precej odvisna še od same izostritve kamere.

```
function capturePhoto() {
     navigator.device.capture.captureImage(captureSuccess, captureError, {limit: 2});
}
```

Ob uspešni izvedbi zgornjega dela kode se pokliče funkcija *captureSuccess*, katera poskrbi za pridobitev shranjene slike in za njihov prikaz na zaslonu. V spremenljivke *path1* in *path2* zapišemo celotno pot do slik. Če pa med zajemanjem slike pride do napake, se izvede funkcija *captureError*, ki uporabniku izpiše opozorilo.

```
function captureSuccess(mediaFiles) {
    var i, path1, path2, len;
        for (i = 0, len = mediaFiles.length; i < len; i += 1) {
           path1 = mediaFiles[0].fullPath;
           path2 = mediaFiles[1].fullPath;
      }
        document.getElementById('majhnaSlika1').src = path1;
        document.getElementById('majhnaSlika2').src = path2;
}
```

Nato sledi vstavljanje slik v t. i. *kanvas*. Okvirje sprva prikažemo na zaslonu. Preko dodeljene identifikacijske oznake ju poiščemo in ustvarimo dvodimenzionalno polje. Vanj vstavimo dobljene slike in jih pomanjšamo na velikost *100 x 100*. Tako bo iskanje razlik med slikama veliko lažje.

```
var canvases = document.getElementById('canvases');
canvases.style.display = 'block';

var kanvas1 = document.getElementById('myCanvas1');
var kanvas2 = document.getElementById('myCanvas2');

var context1 = kanvas1.getContext('2d');
var context2 = kanvas2.getContext('2d');

var img1 = new Image();
img1.onload = function() {
    context1.drawImage(img1,0,0,100,100);
};
img1.src = document.getElementById('smallPhoto1').src;

var img2 = new Image();
img2.onload = function(){
    context2.drawImage(img2,0,0,100,100);
};
img2.src = document.getElementById('smallPhoto2').src;
```

Glavni del aplikacije predstavlja primerjanje slik med seboj. Programska koda, predstavljena v nadaljevanju, je še vedno znotraj funkcije *CaptureSuccess*. Ker imamo slike razmeroma majnih velikosti, ju je enostavno shraniti v polje slikovnih točk. Za primerjavo se s for zanko sprehodimo čez celotno dolžino dobljenih polj in preverjamo točke na enakih mestih. Upoštevati je potrebno določeno mejno vrednost. Če je število

drugačnih slikovnih točk večje od mejne vrednosti, se uporabniku na zaslon izpiše sporočilo s točnim številom zaznanih spremenjenih točk. V nasprotnem primeru gibanje ni bilo zaznano.

Algoritem 2: Detekcija gibanja - PhoneGap

Vhod: Sliki v kanvasu

Izhod: Stevilo različnih slikovnih točk

```
1  imageData1 = context1.getImageData(0, 0, kanvas1.width, kanvas1.height);
   imageData2 = context2.getImageData(0, 0, kanvas2.width, kanvas2.height);
2  pixels1 = imageData1.data; pixels2 = imageData2.data;
3  mv = document.getElementById('threshold').value; if mv == then
4      mv = 30;
5  mejnaVrednost = mv/100 * pixels1.length;
6  count = 0;
7  for i = 0, il = pixels1.length; i < il; i++
8      if pixels1[i]! = pixels2[i] then
9          count++;
10 if count >= mejnaVrednost then
11     alert("Gibanje zaznano! Stevilo razlicnih pixlov :"+ count);
12 sicer
13     alert("Gibanje ni bilo zaznano!");
```

Aplikacija deluje, vendar ne dosega vseh zastavljenih ciljev. Tako kot izvorno Android aplikacijo, je bila tudi ta testirana na HTC in Samsung napravah z operacijskim sistemom Android. Delovanje aplikacije na ostalih platformah ni bilo mogoče preveriti, saj v emulatorji ne podpirajo strojne opreme. Zelo verjetno pa je, da bi delovale podobno kot testirana verzija. Zajemanje slik je težko in neuporabno zaradi uporabnikove interakcije. Delo s predogledi slik v PhoneGapu ni omogočeno, kar je še ena pomanjkljivost. Shranjevanje na zunanji spomin je precej časovno potratno, samo obdelovanje slik pa porabi precej procesorskega časa.

5 Zaključek

Zaključna naloga predstavlja opis področja razvoja aplikacij za mobilne operacijske sisteme. Poudarek je na razvijalskih ogrodjih, ki podpirajo distribucijo izdelane aplikacije večim mobilnim platformam. Predstavljena so ogrodja PhoneGap, Appcelerator Titanium, RhoMobile in Corona SDK. Podrobneje smo si ogledali njihovo delovanje, prednosti in slabosti.

Delo predstavlja tudi razvoj dveh aplikacij. Prva je implementirana specifično za Android OS, druga pa v odprtokodnem ogrodju Phonegap, ki temelji na spletnih tehnologijah. Zanj smo se odločili zaradi enostavnosti uporabe, podprtosti vseh trenutno popularnejših operacijskih sistemov in dobro urejenih knjižnic za dostop do strojne opreme. Ob uporabi ogrodja PhoneGap smo naleteli na nekaj težav, katera so v zadnjem delu tudi opisana. V primerjavi s polno funkcionalno aplikacijo, spisano v izvorni kodi za operacijski sistem Android, so rezultati precej slabši. Delovanje je namreč precej počasno in ne dosega prvotnih ciljev, ki so zaznavanje gibanja v prostoru ter nemoteno avtomatsko delovanje. Ker kakovost ne zadošča zastavljenim produkcijskim merilom, je še ne nameravam objaviti na spletnih trgovinah z aplikacijami.

Zavedati se moramo, da je to ogrodje namenjeno predvsem enostavnim spletnim aplikacijam in ne toliko takšnim, ki za svoje delovanje uporabljajo kompleksnejše operacije s strojno opremo. Kljub temu pa je dokaj novo in se razmeroma hitro razvija ter dopolnjuje pod okriljem močnega podjetja – Adobe. Zato bo v razmeroma kratkem času pritegnilo še večjo množico razvijalcev in bo brez dvoma postalo odlična alternativa razvijalcem, ki prisegajo na izvorne aplikacije.

Literatura

[1] S. Allen, V. Graupera in L. Lee, *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*, Apress, Berkeley, Kalifornija, ZDA, 2010. *(Citirano na strani 3.)*

[2] B. G. Burton inD. Ed, *Mobile App Development with Corona, Getting Started*, Burtons Media Group, Abilene, Texas, ZDA, 2011. *(Citirano na strani 12.)*

[3] Corona, *Dokumentacija Corona SDK* , http://docs.coronalabs.com, 2013. *(Citirano na strani 12.)*

[4] M. Črnilogar, S. Nikolić in J. Vičič, Detekcija gibanja na operacijskem sistemu Android. V *Zbornik enaindvajsete mednarodne Elektrotehniške in računalniške konference ERK*, 2012, 265-268. *(Citirano na strani 3.)*

[5] M. Dinacci, *How to detect motion on an Android device*, http://www.intransitione.com/blog/how-to-detect-motion-on-an-android-device, 2012. *(Citirano na strani 20.)*

[6] P. Friese, *Cross-Platform Mobile Development*, predstavljeno na Code Generation 2011 konferenci, Cambridge, VB, 2011. *(Citirano na strani 4.)*

[7] J. Jardin, *Appcelerator mobile dev – CH 2: Understaind the Titanium framework*, http://johnjardin.ukuvuma.co.za/2013/04/27/appcelerator-mobile-dev-chapter-2-understanding-the-titanium-framework, 2013. *(Citirano na strani 10.)*

[8] J. Koetsier, *Comparing Apples and Googles: The App Store vs. Google Play (infographic)*, http://venturebeat.com/2013/07/17/comparing-apples-and-googles-the-app-store-vs-google-play-infographic, 2013. *(Citirano na strani 2.)*

[9] W. M. Lee, *Build Web Apps for iPhone using Dashcode*, http://mobiforge.com/developing/story/build-web-apps-iphone-using-dashcode, 2009. *(Citirano na strani 1.)*

[10] S. Miller, Motorola Executive Helped Spur Cellphone Revolution, Oversaw Ill-Fated Iridium Project, *The Wall Street Journal* (2009), 10. *(Citirano na strani 1.)*

[11] A. Nalwaya, *Rhomobile, Step-by-step instructions to build an enterprise mobile web application from scratch*, Packt Publishing Ltd., Birmingham, VB, 2011. *(Citirano na strani 10.)*

[12] J. Pelletier, *Mobile App Manual: The Blueprint: How to Start Creating Mobile Apps Using jQuery Mobile and PhoneGap Build*, Withinsight, 2013. *(Citirano na strani 6.)*

[13] PhoneGap, *Dokumentacija PhoneGap*, http://docs.phonegap.com, 2013. *(Citirano na strani 17.)*

[14] RhoMobile, *Dokumentacija RhoMobile*, http://docs.rhomobile.com, 2013. *(Citirano na strani 10.)*

[15] A. Titanium, *Dokumentacija Appcelerator Titanium*, http://docs.appcelerator.com/titanium, 2013. *(Citirano na strani 8.)*

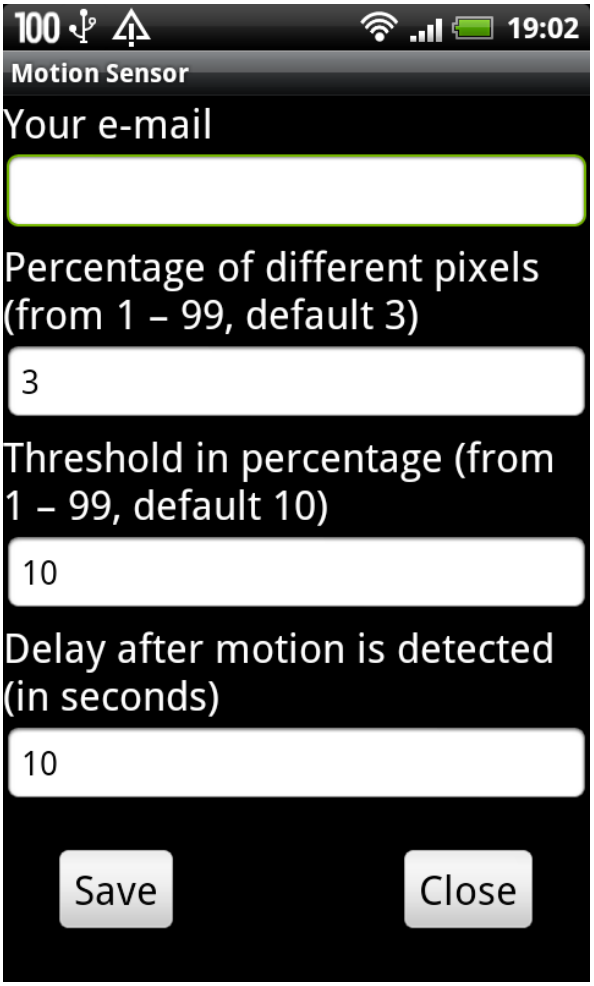
[16] A. Titanium Studio, *Dokumentacija Appcelerator Titanium Studio*, http://www.appcelerator.com/platform/titanium-studio, 2013. *(Citirano na strani 9.)*

[17] J. M. Wargo, *PhoneGap Essentials: Building Cross-Platform Mobile Apps*, Addison-Wesley Professional, 2012. *(Citirano na strani 14.)*

Priloge

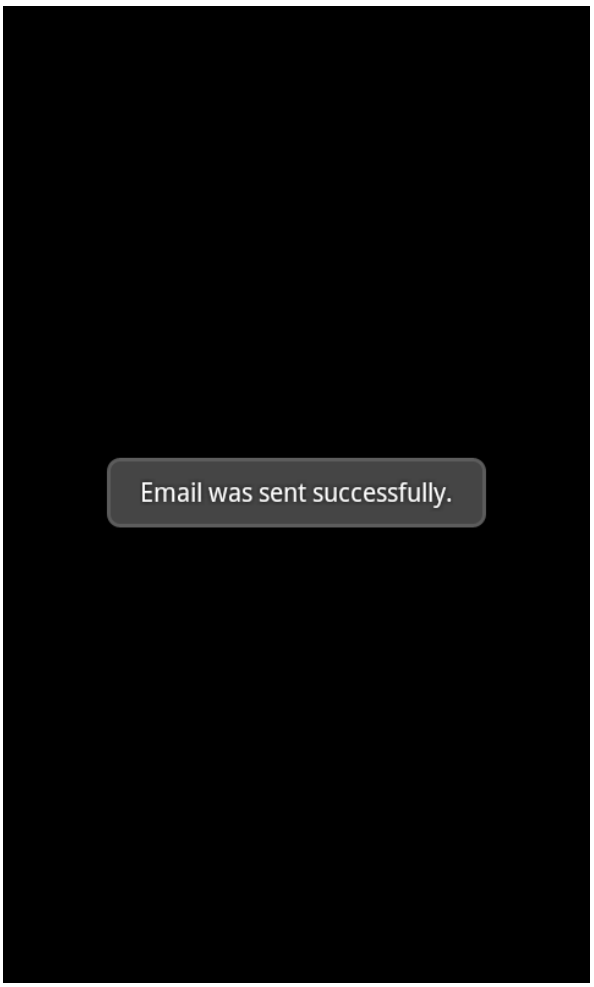
Priloga A: Aplikacija Android

Priložene so slike, s katerimi je v jasnih korakih razvidno delovanje aplikacije, razvite specifično za Android operacijski sistem.

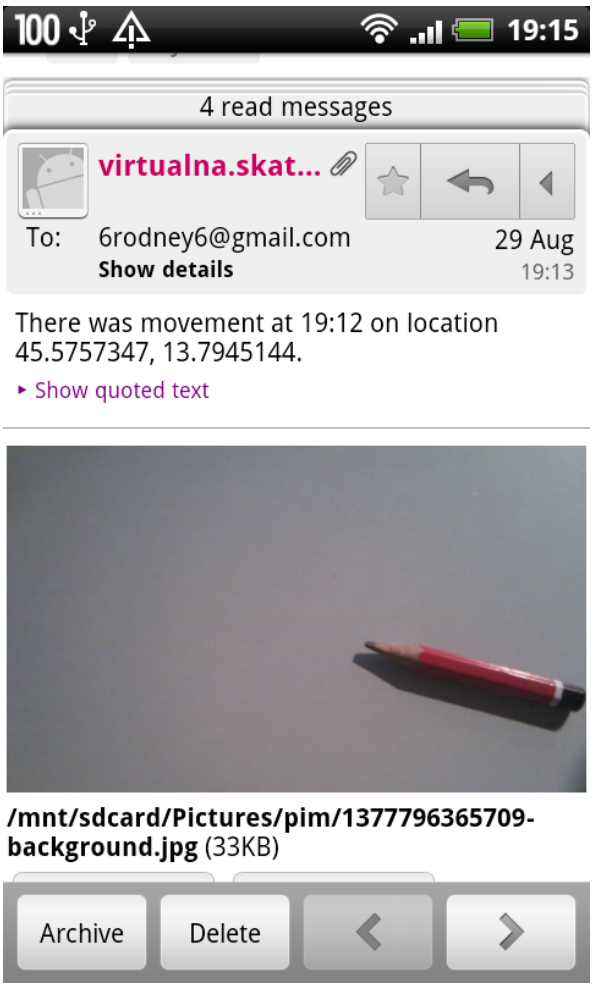


Slika A.1: Začetna stran aplikacije Android, kjer je potrebno vpisati svoj poštni naslov, mejno vrednost ter časovni zamik.

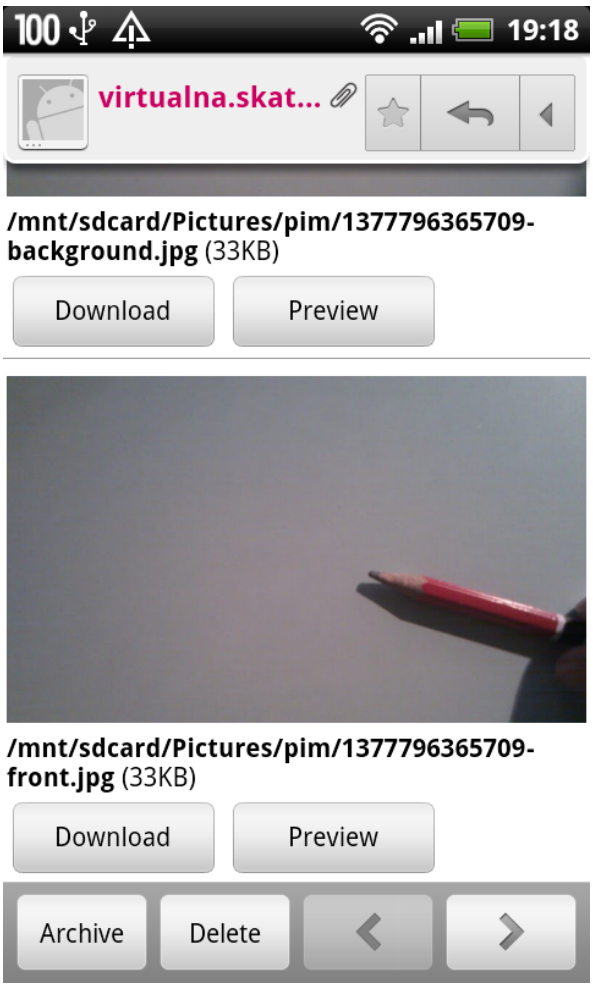
Slika A.2: V drugem koraku je potrebno pognati kamero.



Slika A.3: Ko je gibanje zaznano, se na ekran izpiše opozorilo, da je bilo poslano e-poštno sporočilo.



Slika A.4: V e-poštnem sporočilu je mogoče prebrati uro in lokacijo, kjer je bilo gibanje zaznano, priložene so tudi slike kot dokaz.



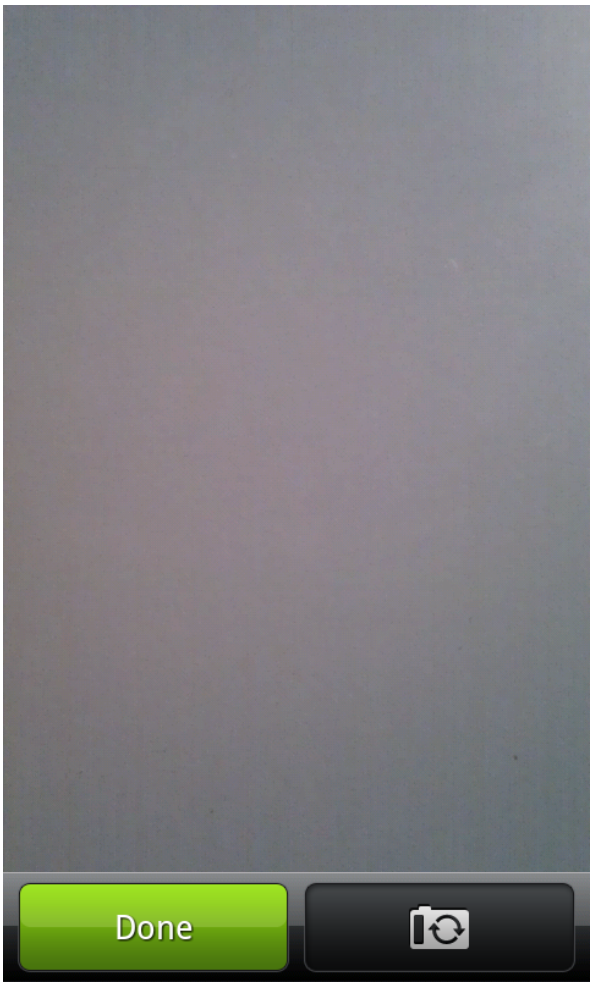
Slika A.5: Priložena je še druga slika, kjer je razvidno, da je do gibanja prišlo, čeprav le minimalnega (saj je mejna vrednost bila le 3 odstotna).

Priloga B: Aplikacija PhoneGap

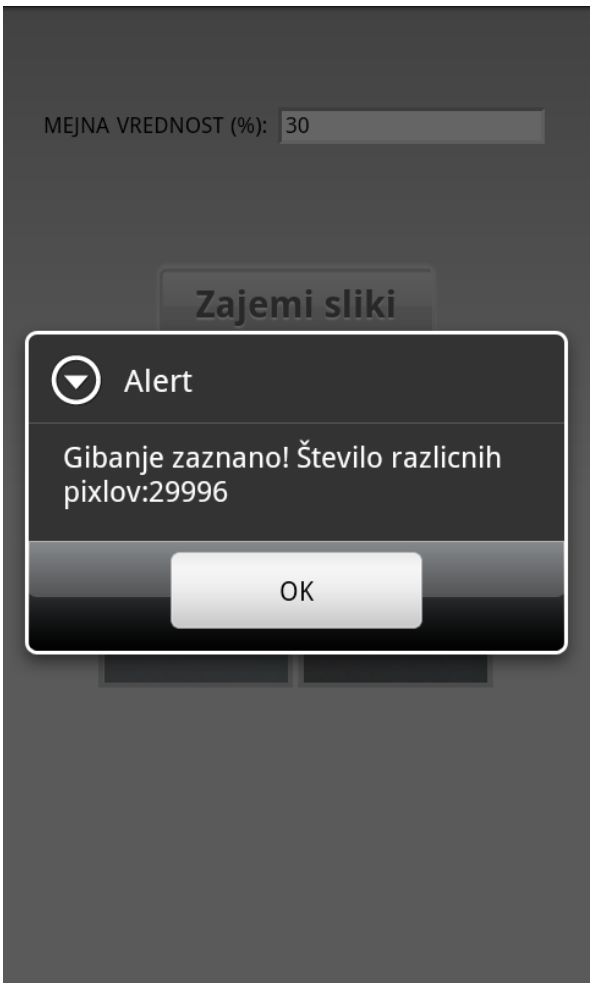
V drugi prilogi so podane slike aplikacije, razvite za več operacijskih sistemov v ogrodju PhoneGap.

Slika B.1: Začetna stran aplikacije PhoneGap, kjer je potrebno vpisati željeno mejno vrednost ter pognati kamero.

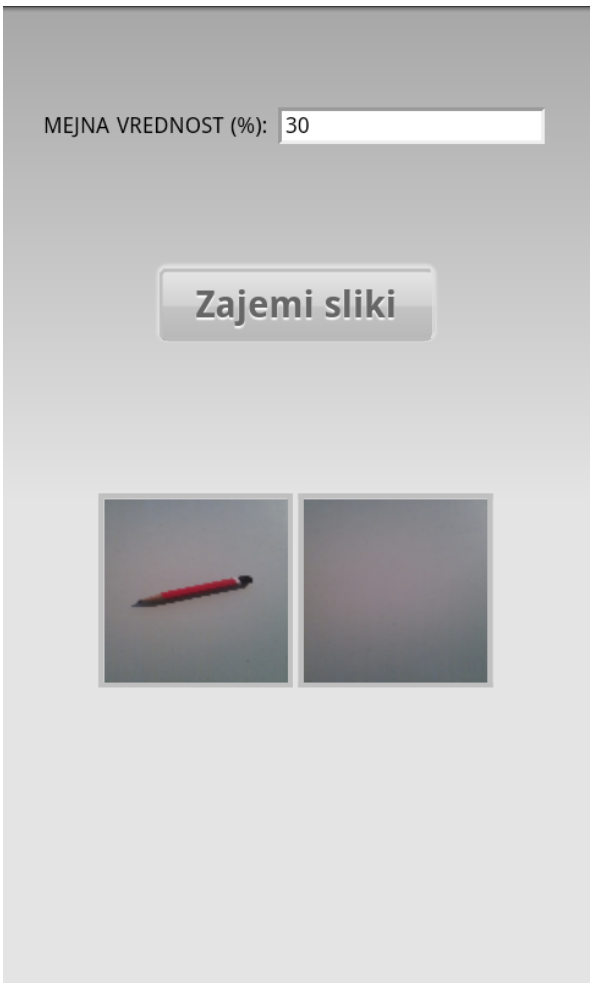
Slika B.2: Po zajemanju slike je potrebno sliko potrditi s pritiskom na gumb.



Slika B.3: Zajeti je potrebno še drugo sliko.



Slika B.4: Ob zaznanem gibanju se na zaslon izpiše opozorilo, poleg tega je navedeno še število različnih slikovnih točk.



Slika B.5: Končni izgled aplikacije, po potrditvi opozorilnega okna s pritiskom na gumb.

Priloga C: Zgoščenka z obema aplikacijama

Tretja priloga je zgoščenka, na kateri sta oba končana projekta. Odpremo ju lahko z razvijalskimi okolji, kot so Eclipse, Android Studio, NetBeans, itd.