Razvoj višeplatformskih aplikacija - Apache Cordova

Horvat, Anamarija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Anamarija Horvat

RAZVOJ VIŠEPLATFORMSKIH APLIKACIJA-APACHE CORDOVA

Diplomski rad

Voditelj rada:
Dr. sc. Goran Igaly

Zagreb, 2016.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem se svojem mentoru dr.sc Goranu Igalyu te prijateljima koji su svojim savjetima, prijedlozima i podrškom pridonijeli izradi ovog diplomskog rada. Posebno se zahvaljujem svojoj majci koja mi je bila velika podrška kroz sve godine mog studija i omogućila mi da isti završim.

Sadržaj

Uvod

Brzim mijenjanjem verzija mobilnih platforma te ukidanje starih ili pojavljivanje novih dovelo je do potrebe razvoja softvera koji može razvijati višeplatformske aplikacije. Jedan takav softver, Apache Cordova, ćemo obraditi u ovom diplomskom radu. On služi za razvoj hibridnih aplikacija, mobilnih aplikacija za čiji se razvoj koriste alati kao za razvoj web aplikacija uz neke nativne programske dodatke. Takvi softveri imaju svoje mane i prednosti, a da bismo pokazali koje razvit ćemo jednu jednostavnu i jednu naprednu aplikaciju. Za razvoj jednostavnih aplikacija softveri koji omogućuju višeplatformski razvoj su bolji izbor od onih za nativni razvoj. Kod razvoja kompliciranijih aplikacija koje trebaju pristup nativnim dijelovima uređaja treba prvo proučiti je li to moguće napraviti pomoću ovakvog jednog softvera ili ipak moramo razvijati nativnu aplikaciju.

Poglavlje 1

Pametni telefoni

1.1 Prvi pametni telefoni

Prvi pametni telefon bio je zapravo mobitel s ugrađenim dlanovnikom. Dlanovnici su preteče pametnih telefona i o njima možete više saznati u [6]. Prvi takav pametni telefon na tržište je stavila Nokia 1996. godine pod nazivom Nokia 9000 Communicator. Imao je mogućnost korištenja elektroničke pošte, web preglednika, kalendara, adresara, kalkulatora, podsjetnika te slanja i primanja faksa.

Iako se on smatra prvim pametnim telefonom taj pojam nije korišten sve do pojave telefona Ericsson R380 kojeg je izdala tvrtka Ericsson Mobile Communications 2000. godine. To je ujedno prvi pametni telefon koji je imao ekran na dodir pomoću olovke (stylus) i prvi pametni telefon koji je koristio operacijski sustav Symbian. Symbian je nakon toga postao operacijski sustav koji se koristio na većini pametnih telefona sve do pojave Blackberry-a, iOS-a i Android-a, da bi ga na kraju ugasili 2011. godine.

Slika 1.1: Nokia 9000 Communicator

1.2 iPhone

1999. godine u Japanu se raširila upotreba pametnih telefona zbog pojave mobilnog interneta, a 2003. godine izašao je prvi Blackberry i postao popularan u SAD-u, no upotreba pametnih telefona nije se popularizirala u ostatku svijeta sve do 2007. godine kada je tvrtka Apple objavila prvi iPhone i to se općenito smatra početkom ere pametnih telefona.

iPhone je bio prvi pametni telefon s ekranom na dodir, umjesto tipkovnice imao je samo jedan gumb na prednjoj strani, "home button", te tri gumba sa strane, dva za pojačavanje odnosno smanjivanje volumena i jedan za uključivanje i isključivanje pametnog telefona. Na slici 1.2 možemo vidjeti kako je izgledao. Taj izgled postao je standard za razvoj pametnih telefona koji su slijedili i na takav izgled smo navikli i danas.

U početku iPhone nije dopuštao korisnicima instaliranje drugih aplikacija, na raspolaganju su imali samo aplikacije koje su dobili prilikom kupnje pametnog telefona. Razvijatelji aplikacija koji su htjeli obogatiti iPhone-a vlastito razvijenim sadržajem mogli su to učiniti jedino razvojem HTML web aplikacija koje su se pokretale u pregledniku iPhone-a. Postojale su neke ekstenzije koje su omogućavale da takve aplikacije izgledaju slično kao Apple-ove aplikacije, no pristup hardveru bio je jako ograničen.

2008. godine pojavio se App Store- usluga koja omogućuje korisnicima pronalaženje i preuzimanje mobilnih aplikacija razvijenih za iOS, operacijski sustav na kojem radi iPhone. To je otvorilo mogućnosti za razvoj aplikacija za isti. Aplikacije su se pisale koristeći programski jezik Objective C, jezik kojim je i napisan sam iOS. Takav razvoj mobilnih aplikacija imao je svoja ograničenja; razvojni programeri su trebali posjedovati Macintosh, osobno računalo tvrtke Apple čija cijena je bila veća od cijene ostalih osobnih računala koja su postojala na tržištu. Macintosh je bio potreban jer je samo na njemu postojao Xcode, integrirana razvojna okolina koja omogućuje razvoj aplikacija za iOS. Također dostupni su bili API-ji, aplikacijska programska sučelja, za koje je postojala dokumentacija. Uz sve to nakon što je aplikacija bila napravljena trebala je proći provjeru prije nego što bi Apple dozvolio da se pojavi na App Store-u.

iPone je tada dominirao tržištem pa proizvođačima aplikacija takva ograničenja nisu prestavljala problem jer su trebali razvijati na jednoj platformi, na jednom uređaju i za mali broj operacijskih sustava (različitih verzija iOS-a). Takav razvoj aplikacija postoji i danas uz neke sitne promjene u uvjetima, no uglavnom još uvijek ima skoro sva navedena ograničenja i iziskuje dosta financijskih resursa.

Slika 1.2: prvi iPhone

1.3 Android

Android je široko rasprostranjena platforma čiji je razvoj započeo 2003. godine. Google ju je preuzeo 2005. godine, a prvi pametni telefon temeljen na ovoj platformi na tržište je došao 2008. godine.

Na konzorciju 5. prosinca 2007. okupili su se proizvođači mobilnih telefona HTC, Samsung i Sony, kompanije koje pružaju usluge bežičnog interneta poput Sprint Nextel i T-Mobile-a, proizvođači čipseta kao Qualcoom i Texas Instruments i odlučili se ujediniti sa ciljem da razviju otvorene standarde za mobilne uređaje i predstavili Android. Prvi pametni telefon s Android operacijskim sustavom bio je HTC Dream poznat i pod imenom G1. Uskoro nakon toga slijedili su i mnogi drugi.

Prema istraživanju tvrke IDC i softverskog alata Statistica smatra se da je Android platforma na 75% pametnih telefona na tržištu. Do kolovoza 2015. proizvedeno je 24093 različitih uređaja od strane 1294 tvrtki. Za razliku od iPhone-a uređaji se razlikuju u skoro svemu; veličini ekrana, rezoluciji, procesoru, memoriji, dodatnim sadržajima to jest značajkama i verziji operacijskog sustava. Uskoro je ideja da se razvija aplikacija koja bi radila na svim uređajima postao cilj koji je jako teško ili nekad čak nemoguće postići ali cilj kojem svi teže. Više o pametnim telefonima možete pročitati u [13]

Slika 1.3: prvi Android uređaj

Poglavlje 2

Razvoj višeplatformskih aplikacija

Zbog velikog broja različitih uređaja koji rade na različitim operacijskim sustavima javila se potreba za razvojem aplikacija koje također mogu raditi na različitim uređajima različitih operacijskih sustava. S obzirom da se uređaji jako razlikuju hardverski i softverski to nije nimalo lagan zadatak. Aplikacije koje se rade za iPhone pišu se u drugačijem programskom jeziku, na drugačijem operacijskom sustavu i drugačijem računalu od onih koje se pišu za Android i ostalih. Za kompanije to uglavnom znači da trebaju zaposliti veliki broj programera i uložiti podosta financijskih resursa. Upravo zbog toga razvili su se različiti načini razvoja aplikacija koje se mogu napisati u jednom programskom jeziku i koristeći različite alate prilagoditi tako da rade na drugim operacijskim sustavima. U ovom poglavlju ćemo objasniti koji su sve načini prilagodbe mogući.

2.1 Višeplatformski prevoditelji

Višeplatformski prevoditelji su programski prevoditelji koji uzimaju izvorni kod napisan u jednom programskom jeziku i prevode ga u ekvivalentan izvorni kod u drugom programskom jeziku, npr. iz Pascala u C. Razlika između standardnih prevoditelja je ta što standardni prevoditelji prevode izvorni kod iz višeg programskog jezika u niži dok višeplatformski prevoditelji prevode izvorni kod u jezik iste apstrakcijske razine.

Druga stvar za koju se oni koriste je za prevođenje izvornog koda iz jedne verzije programskog jezika u drugu verziju programskog jezika npr. iz Python-a 2 u Python 3, te također za prevođenje iz starije verzije API-ja u noviju verziju ili refaktoriranje koda.

Višeplatformski prevoditelji mogu biti namijenjeni različitim programskih jezicima, za različite namjene- desktop aplikacije, web aplikacije te mobilne aplikacije. Poznatiji višeplatformski prevoditelji koji se mogu koristiti u svrhu razvoja mobilnih aplikacija su Haxe, Dart i XMLVM. Svi su otvorenog tipa i mogu se skinuti na njihovim službenim stranicama.

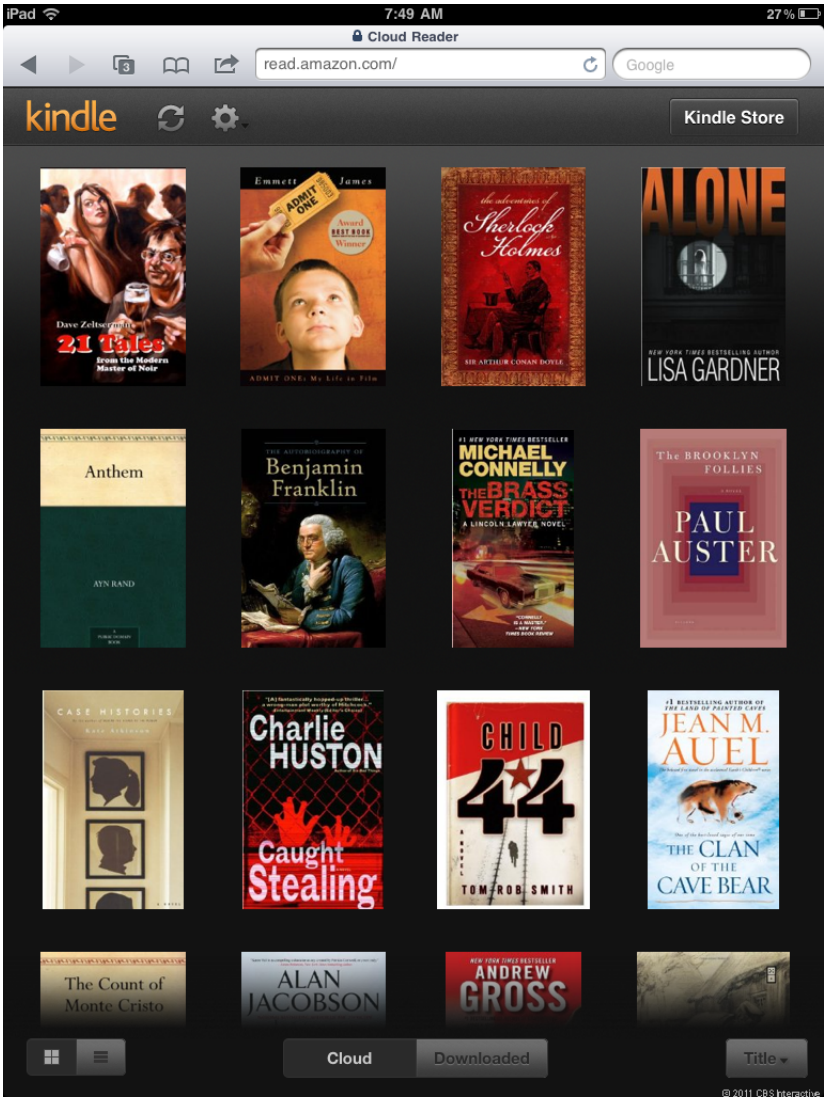
Postoje višeplatformski prevoditelji na višoj razini koji prevode izvorni kod u nativne aplikacije. Najpoznatiji su Xamarin i Appceleratorov Titanium. Xamarin prevodi aplikacije pisane u C#-u u nativne aplikacije za Android, iOS i Windows Phone. Može se koristiti unutar Microsoft Visual Studija ili se može se njihove stranice skinuti Xamarin Studio. Također se može koristiti Xamarin Test Cloud gdje se mogu testirati napisane aplikacije. Appceleratorov Titanium prevodi JavaScript u nativne aplikacije, a sam prijevod se događa na mobilnom telefonu koristeći JavaScript jezgru, Mozzilin Rhino na Android-u i BlackBerry-u te Javascriptcore na iOS-u. Upravo zbog toga što se prijevod događa tek prilikom pokretanja aplikacije, te aplikacije rade sporije od nativnih aplikacija. Druga negativna stvar je da se aplikacije mogu besplatno razvijati no ne mogu se pustiti na tržište, cijena korištenja kreće se od 40$ do 259$ mjesečno.

Korištenje višetplatformskih prevoditelja odlično je rješenje kada brzo želimo razviti aplikaciju, stoga ih neki developeri koriste za razvoj prototipa. Ako je to alat za razvoj konačne aplikacije treba prvo istražiti i obratiti pozornost na moguće nedostatke kao npr. kod Appceleratorovog Titanium-a, za neke aplikacije možda nema potrebe pisati izvorni kod za ciljane operativne sustave no za neke će možda biti potrebno izabrati neko drugo rješenje.

2.2 HTML 5 Web aplikacije

Jedna stvar koje je zajednička svim operacijskim sustavima jest da imaju ugrađen web preglednik. Pojavom HTML-a 5 2014. godine web aplikacije mogu izgledati i imati iste funkcionalnosti kao nativne mobilne aplikacije stoga je i to jedan od mogućih načina razvoja višeplatformskih aplikacija. HTML 5 je uveo mogućnost lokalne pohrane podataka, geolokaciju, audio i video koristeći Canvas, mogućnost rada bez pristupa internetu, validacije formi bez korištenja javascript-a i mnoge druge.

Najbolji primjer takve aplikacije je Amazon Kindle web aplikacija Kindle Cloud Reader. Nastala je kao reakcija ne Apple-ovu politiku poslovanja, naime Apple je zabranio da mobilne aplikacije imaju poveznice na vanjske linkove za kupovanje digitalnih knjiga ili pretplata na časopise. Aplikacija jako liči na Kindle-ove Android i iOS aplikacije i podržana je za Safari 5+, Google Chrome 20+, Firefox 10+ i Internet Explorer 10+. Prijavom na svoj Amazon račun dobiva se pristup e-knjižari. Naravno za to je potrebna internetska konekcija, no nakon što smo kupili knjigu Amazon omogućuje spremanje do 50MB na uređaj pa ih možemo čitati i bez pristupa internetu kao što je moguće na Android i iOS aplikacijama. Na slici 2.1 prikazan je izgled ove web aplikacije.



Slika 2.1: Kindle Cloud Reader na iPad-u

Jedna mana HTML 5 aplikacija je više sociološka nego tehnička, većina ljudi danas očekuju da se aplikacije nalaze u trgovinama za aplikacije i navikli su na instalacijski proces uređaja kojeg koriste, web preglednik doživljavaju kao sredstvo za traženje informacija a ne kao alat za korištenje aplikacija. Druge mane su da nisu svi API-i podržani te da je ograničena količina podataka koji se mogu lokalno pohraniti.

2.3 HTML 5 hibridne aplikacije

Sljedeće moguće rješenje je razvijati aplikacije koristeći kombinaciju HTML-a 5, Javascripta i dodatnih biblioteka te umatanjem u relativno tanki omotač nativnog koda (native-code wrapper). Prednost ovog pristupa naspram navedenih u prethodna dva poglavlja je da kombinira prednosti svakog od ova dva pristupa - HTML 5 i nativnih aplikacija. Programski jezik je isti za sve ciljane operacijske sustave, API-ji su također isti za podržane platforme. Nadalje, pristup hardveru uređaja puno je veći nego kod HTML 5 web aplikacija, lokalna pohrana podataka omogućuje prijelaz limita od 5 Mb, slike mogu biti poslane (upload) sa uređaja, procesi se mogu izvršavati u pozadini uređaja i tako dalje.

Vodeći u ovom području je razvojni okvir PhoneGap danas doniran Apache Sofware Foundation i preimenovan u Apache Cordova. Omogućuje razvoj aplikacija za Android, iOS, BlackBerry i Windows Phone. Pošto je glavna tema ovoga rada o njemu ćemo više reći u sljedećim poglavljima.

Makar je ovo danas najpopularniji način za razvoj višeplatformskih aplikacija ima jednu veliku manu, zahtijeva SDK za svaku ciljnu platformu. Android-ov SDK zahtijeva instalaciju Jave, iOS-ov Swift i Mac računalo, Windows Phone .NET Framework SDK sa .NET-om i taj proces nabavljanja istih nije lagan zadatak. Danas zato postoje softveri koji olakšavaju taj postupak, najpoznatiji su Adobe PhoneGap, Telerik, Visual Studio, App builder, Ionic no postoje i mnogi drugi. Oni obično imaju u sebi već instalirane SDK-ove (neke) i/ili imaju mogućnost simuliranja aplikacija na emulatorima.

Velika većina aplikacija može se napraviti koristeći hibridni način no aplikacije koje zahtijevaju dubinski pristup hardveru poput npr. Tasker aplikacije za Android, te igre koje su grafički zahtjevne ipak se ne mogu razviti na ovaj način. Kada se oblikuje ideja za aplikaciju potrebno je stoga proučiti što će ona sve trebati raditi i čemu imati pristup te tada odlučiti je li moguće izbjeći nativan razvoj ako je to je definitivno bolje rješenje jer je brže i zahtijeva manje ljudskih i financijskih resursa.

Poglavlje 3

Apache Cordova-Uvod

Apache Cordova je besplatan razvojni okvir za razvoj višetplatformskih hibridnih aplikacija koristeći HTML 5, CSS3 i JavaScript. Sastoji se od sljedećih komponenti:

- Programskog koda za okvir nativne aplikacije za svaku podržanu mobilnu platformu. Taj okvir služi za renderiranje HTML 5 aplikacije na uređaju.

- Liste API-ja koji omogućuju web aplikaciji koja se izvršava unutar okvira pristup nativnim mogućnostima uređaja i njegovim API-jima kojima je inače nemoguće pristupiti preko web preglednika.

- Alata koji omogućuju i olakšavaju razvoj aplikacije poput alata za upravljanje programskim dodacima (plug-in), alata za gradnju aplikacije, alata za testiranje aplikacije tj. emulatora i simulatora mobilnih uređaja.

Većina razvojnih programera, kada prvi put čuje za Apache Cordovu, misle da je web aplikacija prevedena u nativan jezik za svaku podržanu mobilnu platformu, Objective-C za iOS, Javu za Android. No ovdje se ne radi o višeplatformskom prevodiocu, ovo je hibridna aplikacija, što znači da se web aplikacija izvršava nemodificirana unutar ljuske nativne aplikacije. Unutar nativne aplikacije, aplikacijsko korisničko sučelje sastoji se samo od web pogleda (web view) koji prekriva cijeli ekran. Web view je komponenta koju ima svaki programski jezik za razvoj mobilnih aplikacija koja omogućuje prikaz web sadržaja obično HTML stranica unutar prozora nativne aplikacije. Kada se aplikacija pokrene, učitava se početna stranica web aplikacije u web view i predaje mu se kontrola što omogućuje korisniku interakciju s web aplikacijom. Prilikom te interakcije, linkovi ili JavaScript kod unutar aplikacije mogu učitati druge sadržaje iz dokumenata koji su zapakirani sa aplikacijom ili preko internetske veze pristupajući nekom web serveru. Web aplikacija koja se izvršava u web view-u ima sve mogućnosti kao obična web aplikacija, može otvarati druge HTML

stranice (lokalne ili na nekom serveru), JavaScript služi za implementiranje logike aplikacije poput upravljanja prikaza informacija na web stranici, pokretanja medijskih sadržaja (audio, video), računskih operacija, slanja/primanja sadržaja na/sa servera i slično. Sam izgled aplikacije uređuje se direktno u HTML-u dodavanjem atributa poput fonta, boje, proreda i slično ili se zasebno implementira u CSS-u, ekvivaletno kao pri izradi obične web stranice.

Po ovom što smo do sada rekli zvuči kao da je jedina razlika između web aplikacije i aplikacije napravljene pomoću Apache Cordova to što se web aplikacija ne izvršava u pregledniku već u web view-u nativne aplikacije, no postoji veoma bitna razlika. Web aplikacije nemaju pristup drugim aplikacijama na mobilnom uređaju, njegovom hardveru, i nativnim API-jima, dok sve to aplikacije napravljene u Apachi Cordova mogu. Web aplikacije na primjer ne mogu pristupiti kontaktima u mobilnom uređaju, akcelerometru, kameri, mikrofonu i slično niti mogu provjeriti stanje internetske veze uređaja. Većina mobilnih aplikacija danas zahtijeva pristup nekim od tih stvari da bi korisnicima bila zanimljiva i to je ujedno razlog zašto je Apache Cordova i nastao. Pristup dodatnim mogućnostima koje mobilni uređaj pruža omogućen je preko Cordovinih plug-ina. Ovisno o potrebi u projekt možemo dodavati i uklanjati plug-inove. Trenutno podržani plug-inovi su sljedeći: Battery status (stanje baterije), Camera (kamera), Console (konzola), Contacts (kontakti), Device (uređaj), Device Motion (kretanje uređaja), Device Orientation (orijentacija uređaja), Dialogs (dijalozi), File (datoteka), File Transfer (prijenos datoteka), Geolocation (geolokacija), Globalization (globalizacija), Inappbrowser (preglednik unutar aplikacije), Media (mediji), Media Capture (snimanje medija), Network information (mrežni podaci), Splash Screen (naslovni ekran), Vibration (vibracija), Statusbar (statusni redak), Whitelist (popis dopuštenih), Legacy Whitelist (zastarjeli Whitelist).

Kad programer napiše aplikaciju koja koristi neki od Cordovinih API-ja, ta aplikacija koristi taj API pomoću JavaScript-a. Poseban sloj unutar aplikacije prevodi Cordovin API u odgovarajući nativan API. To je potrebno jer na primjer način na koji Android pristupa kameri nije isti kao iPhone-ov način pristupanja kameri. Taj sloj za prevođenje API-ja omogućuje programeru da napiše jedno sučelje koje se onda u pozadini (u okviru aplikacije) prevodi u odgovarajući API za svaku podržanu mobilnu platformu. Da bi napravili fotografiju kamerom koristeći Cordovu i standardne postavke za API, JavaScript kod bi izgledao ovako:

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

Parametri su dvije funkcije, `onSucces` i `onFail`. Prva je pozvana ako je fotografija uspješno napravljena, a druga ako se dogodila neka greška. Na Android-u kod koji se poziva u pozadini bi mogao izgledati poput:

```
camera.takePicture( shutterCallback, rawCallback,
jpegCallback );
```

a na iPhone-u taj kod bi mogao izgledati kao:

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

Ovaj primjer ne pokriva cjelokupan proces uzimanja fotografije (način na koji se obrađuju greške niti proces koji se dešava nakon što je fotografija napravljena), ali ilustrira kako Apache Cordova olakšava višeplatformsko programiranje. Programer napiše jednu funkciju za korištenje API-ja, a Cordova ga prevede u svaku podržanu platformu. Programer ne treba imati znanje o razvoju nativnih Android ili iPhone aplikacija, zamarati se kodom koji se dešava u pozadini niti razmišljati kako napraviti da nešto radi na više uređaja već se može fokusirati na funkcionalnost i dizajn svoje aplikacije.

Apache Cordova trenutno podržava sljedeće operacijske sustave:

- Android
- BlackBerry 10
- iOS
- Ubuntu
- Windows Phone 8
- Windows 8.1
- Windows 10

Aplikacija koju pišemo ima isti kod za svaki operacijski sustav, no ne može se testirati niti napraviti na svakom, tj. aplikacije za Android i BlackBerry 10 možemo raditi na Windows-u, Linux-u i Mac-u, dok iOS samo na Mac-u, Windows Phone i Windows (8.1 i 10) samo na Windows-u, te Ubunutu samo na Ubuntu. U sljedećem poglavlju objasnit ćemo kako se instalira Apache Cordova, te kako omogućujemo razvoj za sve platforme.

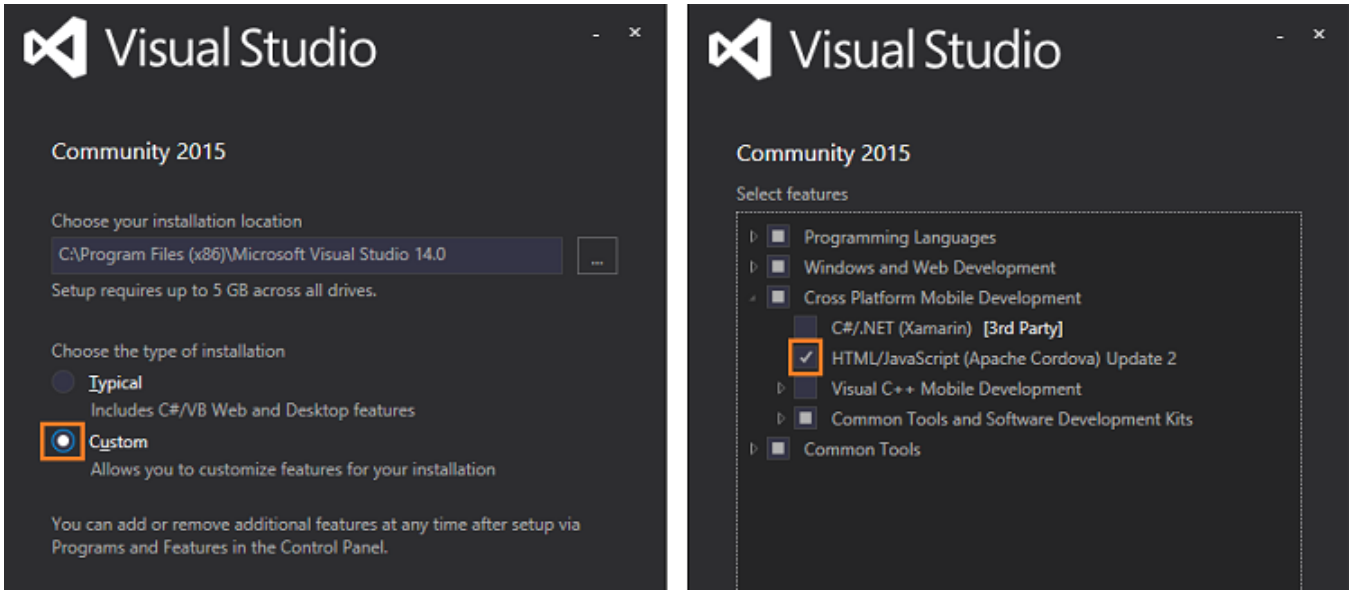
Poglavlje 4

Apache Cordova-instalacija

Kao što je spomenuto u poglavlju 2, Apache Cordova olakšava razvoj višeplatformskih aplikacija no sam postupak instalacije je zato dug i teži nego što bi bio kada bismo razvijali nativnu Android aplikaciju koristeći na primjer Android Studio. Apache Cordova zahtijeva instalaciju Apache Cordova frameworka, komandnog sučelja, JavaScript sloja, plug-inova, SDK-ova za svaku ciljanu platformu, i mnogih drugih stvari. Samo korištenje se onda svodi na pisanje koda u nekom tekstualnom editoru, a kod se onda gradi i spaja u aplikaciju koristeći komandno sučelje. Na takav način programiranja moderni programeri nisu naviknuti. No također smo spomenuli u drugom poglavlju da ima više razvojnih okolina koje olakšavaju postupak instalacije, proces gradnje i povezivanja potrebnih datoteka za Apache Cordovu. Ovdje smo izabrali Visual Studio kao razvojnu okolinu jer ima ugrađeno sve za razvoj Android, Windows Phone i Windows aplikacija, jedino što zahtijeva dodatne instalacije je ako nam je ciljni operacijski sustav iOS.

4.1 Instalacija Visual Studija

Visual studio postoji u više verzija; Community, Professional i Enterprise. Prvi je besplatan, ostala dva se plaćaju. Razlikuju se uglavnom u podršci i dodatnim mogućnostima, prvi i drugi su za samostalan razvoj aplikacija, a treći je za velike firme. Proces instalacije je isti kod svakog koji se odabere sa njihove službene stranice. Ako nam je cilj napraviti Windows Phone i Windows aplikacije potrebno je imati barem Windows 8 instaliran na računalu. Na početku instalacije treba odabrati **Custom** i onda selektirati **HTML/JavaScript(Apache Cordova)**. Vidi sliku 4.1 Ako već imamo instaliran Visual Studio, možemo



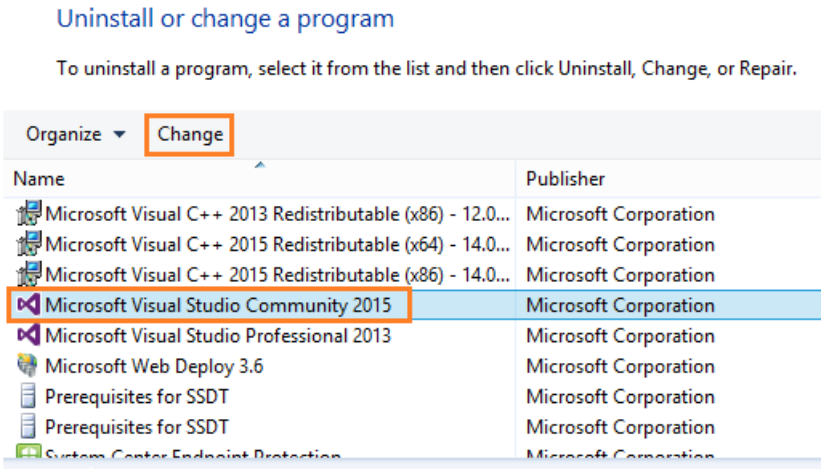
Slika 4.1: Instalacija Visual Studija

ga modificirati to jest naknadno instalirati Apache Cordova tools. Da bismo to učinili trebamo otvoriti **Control Panel ->Programs and Features**, izabrati **Visual Studio 2015** i onda kliknuti **Change**, Vidi sliku 4.2

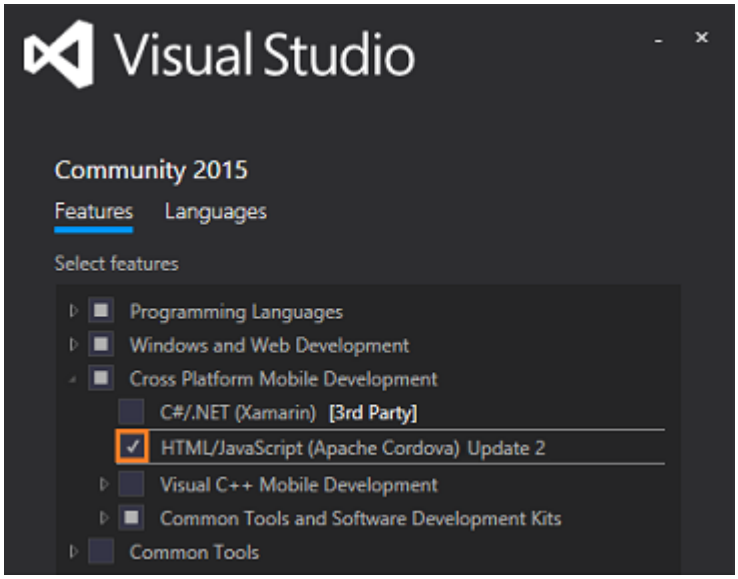
Nakon toga otvorit će se postavke u kojima odaberemo **Modify** i onda pod **Features** selektiramo **HTML/JavaScript(Apache Cordova)** nakon čega kliknemo **Next** pa **Update** gumb. Vidi sliku 4.3

Nakon što smo instalirali Apache Cordova tools, trebamo ih nadograditi. U Visual Studiju u gornjem meniju odabermo **Tools ->Extensions and Updates**. Otvorit će se prozor u kojem u na kartici **Update** izaberemo **Product Updates** i ako se pojavi **Visual Studio tools for Apache Cordova** selektiramo to i kliknemo **Update** gumb. Vidi sliku 4.4

Visual Studio sam instalira dodatne komponente koje su potrebne za razvoj Android aplikacija. Kada se ne bi koristila ova razvojna okolina to bismo trebali napraviti sami. Popis dodatnih komponenti nalazi se u tablici 4.1

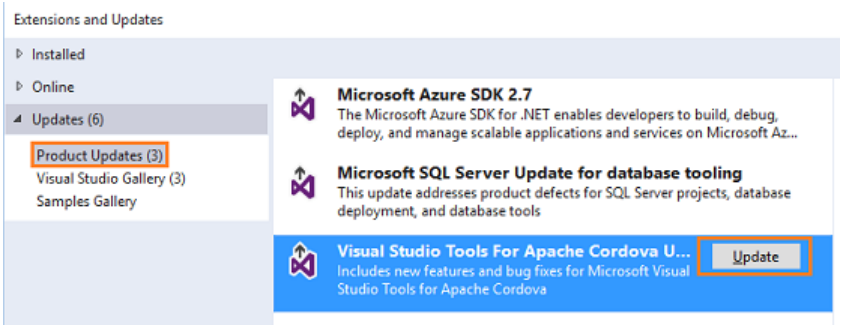


Slika 4.2: Modifikacija Visual Studija



Slika 4.3: Naknadni odabir paketa

Komponenta	**Zašto je Visual Studio Instalira**
Apache Ant 1.8.0 ili noviji	Za razvoj Android aplikacija
32-bit Oracle Java JDK 7	Za razvoj Android aplikacija
Android SDK	Za razvoj Android aplikacija i pokretanje Apache Ripple simulatora
Joyent Node.js	Za integraciju komandnog sučelja Apache Cordove sa Apache Ripple simulatorom
Git CLI	Omogućuje dodavanje Cordovinih plug-inova sa Git-a



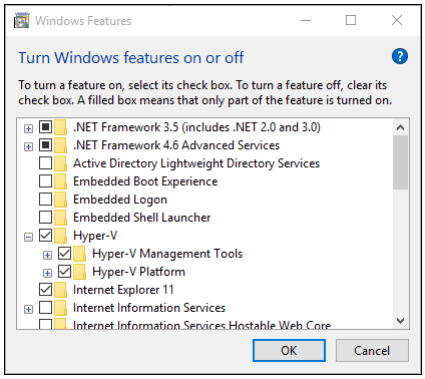
Slika 4.4: Nadogradnja

Za korištenje Apache Ripple simulatora potrebno je instalirati i Google Chrome, koji se može preuzeti s interneta. S tim smo završili svu potrebnu instalaciju za razvoj Android aplikacija.

Kako bismo mogli razvijati aplikacije za Windows i Windows Phone već smo rekli treba imati instaliran Windows 8.1 ili kasniju verziju na računalu, no potrebno je još omogućiti Hyper-V u Windows postavkama. Hyper-V omogućuje simuliranje Windows Phone operacijskog sustava. Ovdje ćemo reći kako se to radi na Windows 10 no na Windows 8.1 je slično. Kliknemo desnim gumbom miša na **Windows** gumb i selektiramo **Programs and Features**. Zatim odaberemo **Turn Windows Features on or off**, pronađemo **Hyper-V** označimo ga i kliknemo **OK**, vidi sliku 4.5. S ovime smo završili instalaciju Apache Cordova na Windowsima, u sljedećem poglavlju ćemo reći kako se pomoću Visual Studija razvijaju aplikacije kada je ciljni operacijski sustav iOS.

4.2 Ciljanje iOS-a sa Visual Studijom

Da bismo mogli razvijati aplikacije za iOS potrebno je imati računalo Macintosh. Apple je bio i ostao tvrtka sa veoma zatvorenom politikom pa vjerojatno ni u budućnosti nećemo moći razvijati aplikacije bez njega. Ne treba nužno imati fizičko računalo Macintosh, može se unajmiti Mac server na internetu, Mac u oblaku, no tvrtke uglavnom imaju bar jedno računalo Macintosh. Na njemu ne trebamo pisati kod aplikacije no potreban je za povezivanje, gradnju i testiranje aplikacije. Ako više programera radi na istoj aplikaciji, mogu to činiti svaki na svojem Windows računalu i koristiti jedno zajedničko Macintosh računalo za testiranje. Takav pristup je najčešći u malim ili novim firmama, te firmama koje koriste ovakav hibridni razvoj aplikacija jer su računala Macintosh skuplja u odnosu na druga.



Slika 4.5: Omogućivanje Hyper-a V

Sada ćemo objasniti što sve treba instalirati na računalu Macintosh kako bismo mogli koristiti Visual Studio za razvoj iOS aplikacija.

Instalacija potrebnih programa

Prvo treba instalirati Node.js na računalo. Node.js može se naći besplatno na adresi https://nodejs.org/en/download/releases/. Verzija koja je kompatibilna sa svim verzijama Cordove je 0.12.9, a ako želimo koristiti verzije 4x ili 5x projekt treba koristiti Cordova CLI 5.4.1 ili noviji. S obzirom da i za Node.js i za Apache Cordovu često izlaze nove verzije najbolje je provjeriti na adresi http://taco.visualstudio.com/en-us/docs/change-node-version/install-the-most-compatible-version-of-nodejs-012x koju verziju treba instalirati na računalo.

Nakon što smo našli pravu verziju i instalirali je treba instalirati Xcode. Xcode se može preuzeti sa adrese https://developer.apple.com/xcode/download/. On omogućuje razvoj iOS aplikacija neovisno o Apache Cordovi, to jest da bi se razvile bilo kakve iOS aplikacije potrebno ga je imati. Nakon toga u **Launchpad**-u treba otvoriti **Xcode** i pojavit će se dijalog koji nas pita prihvaćamo li pravila korištenja Xcode-a, trebamo kliknuti **Accept** i zatvoriti prozor. Ovo se možda čini kao čudan korak no bez prihvaćanja licence Xcode se ne može koristiti makar je instaliran. Sada treba instalirati alate za korištenje komandne linije Xcode-a. To se napravi tako da se otvori **Terminal** i upiše se sljedeća naredba:

```
xcode-select --install
```

Zatim napišemo sljedeću naredbu:

```
sudo npm install -g remotebuild
```

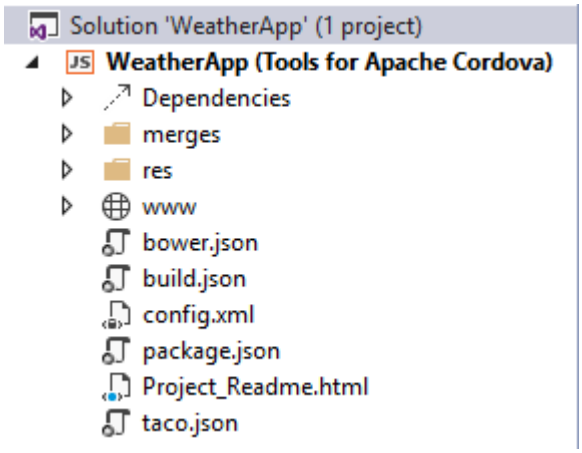
Ona instalira ”remote agent”. To je siguran server koji se vrti na računalu Macintosh, na njega se treba spojiti sa Windows računala kako bismo mogli graditi, pokretati i debugirati iOS aplikacije iz Visual Studija.

Poglavlje 5

Razvoj prazne aplikacije

Kreiranje novog projekta

U gornjem meniju kliknemo na **New ->New project** te pod **Javascript** izaberemo **Apache Cordova Apps**, te desno selektiramo **Blank App (Apache Cordova) Javascript**. U donjim poljima unesemo ime projekta i lokaciju gdje ćemo ga spremiti te kliknemo **Create**. Ova aplikacija pisana je u JavaScriptu, no Visual studio također ima mogućnosti pisanja aplikacija u programskom jeziku TypeScript. Ako želimo pisati aplikaciju u njemu onda umjesto **JavaScript** kod kreiranja novog projekta izaberemo **TypeScript - >Apache Cordova Apps**. Nakon kreiranja projekt bi trebao imati dokumente i datoteke u Solution Explorer-u kao na slici 5.1



Slika 5.1: Solution Explorer

Dokumenata i mapa ima puno, no neke nećemo trebati koristiti. Sada ćemo reći osnovno o njima, a kasnije ćemo detaljnije o onima koji su bili potrebni za razvoj aplikacije.

Konfiguracijski dokumenti projekta

- **bower.json**
  On služi za dodavanja Bower paketa u aplikaciju. Bower paket je neki JavaScript plug-in. JavaScript Plug-in se obično sastoji od HTML, CSS i JavaScript dokumenata, a Bower ih sve zapakira u jedan dokument što olakašava proces instalacije. Za razvoj ove aplikacije nije bio potreban, no često se u razvoju aplikacija koriste neki JavaScript plug-inovi koji su zapakirani pomoću Bower-a. Kako instalirati i koristiti Bower pakete u Visual Studiju može se pročitati na adresi https://taco.visualstudio.com/en-us/docs/tutorial-using-bower/.

- **build.json**
  Ovaj dokument je potreban ako želimo staviti Android aplikaciju u Google Store, o njemu ćemo više reći u kasnijim poglavljima.

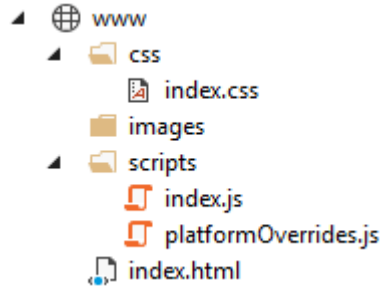
- **config.xml**
  U njemu se mijenjaju postavke aplikacije, poput za koje operacijske sustave se razvija, dodavanje plug-inova i slično. Potrebno ga je urediti kod razvoja svake aplikacije pa ćemo više reći o njemu u sljedećim poglavljima.

- **taco.json**
  Definira koja verzija Apache Cordove će se koristiti za gradnju aplikacije. Defaultna vrijednost je najnovija koja je instalirana stoga ovaj dokument ne treba mijenjati osim ako ne želimo ciljati neku stariju verziju.

- **package.json**
  On služi za osnovni pregled potrebnih datoteka za gradnju aplikacije i ostalih metapodataka. Ne mora se ispuniti, no poželjno je ako će se kod dijeliti s drugim ljudima.

Mape u projektu

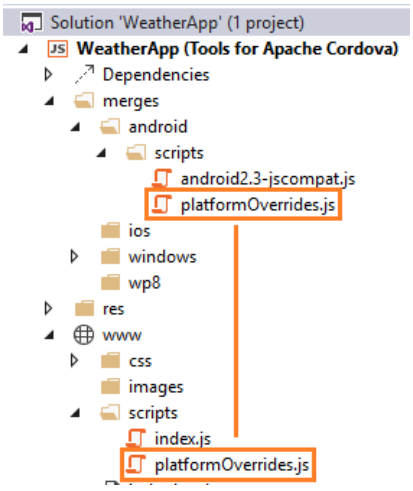
- **www**
  Ovo je najbitnija mapa. U njoj se nalaze HTML, JavaScript i CSS dokumenti koji se koriste za razvoj aplikacije. Ovdje se također nalazi i mapa sa slikama koje će aplikacija koristiti, te se može dodati i mapa sa zvukom što ćemo mi napraviti kasnije jer je potrebna za razvoj Weather Alarm aplikacije. U početku ta mapa izgleda kao na slici 5.2



Slika 5.2: Sadržaj ”www” mape

- **merges**

  Ova mapa sadrži HTML, JavaScript i CSS dokumente koji su specifični za neku platformu. Možda bismo htjeli da aplikacija različito izgleda na različitim platformama, ili možda neki dokumenti trebaju biti drugačiji jer nešto što bi radilo na Android uređaju možda neće raditi na iOS-u. Kako to radi? Svi dokumenti i mape iz **merges/*imeplatforme*** kopiraju se prilikom gradnje aplikacije u glavnu, **www** mapu, a oni koji imaju isto ime se prebrišu i zamijene s onima iz **merges/*imeplatforme***. Na primjer **scripts/platformOverrides.js** datoteka će se zamijeniti sa **merges/android/scripts/platformoverrides.js** na Androidu, kao na slici 5.3. Na ovaj način pišemo platformski specifičan kod, isto je dakle moguće napraviti sa mapama poput slika i zvuka, te CSS i HMTL datotekama.



Slika 5.3: Uloga **merges** mape

- **res**
  Ova mapa sadrži ikone, početne ekrane, certifikate i slične datoteke i mape koje su nativno specifične za neku platformu. Ona se u pravilu ne mijenja.

- **Dependencies**
  Ova mapa je prazna osim ako nismo npr. instalirali neki Bower paket. Tada bi se u ovoj mapi nalazile datoteke iz njega.

Pokretanje aplikacije

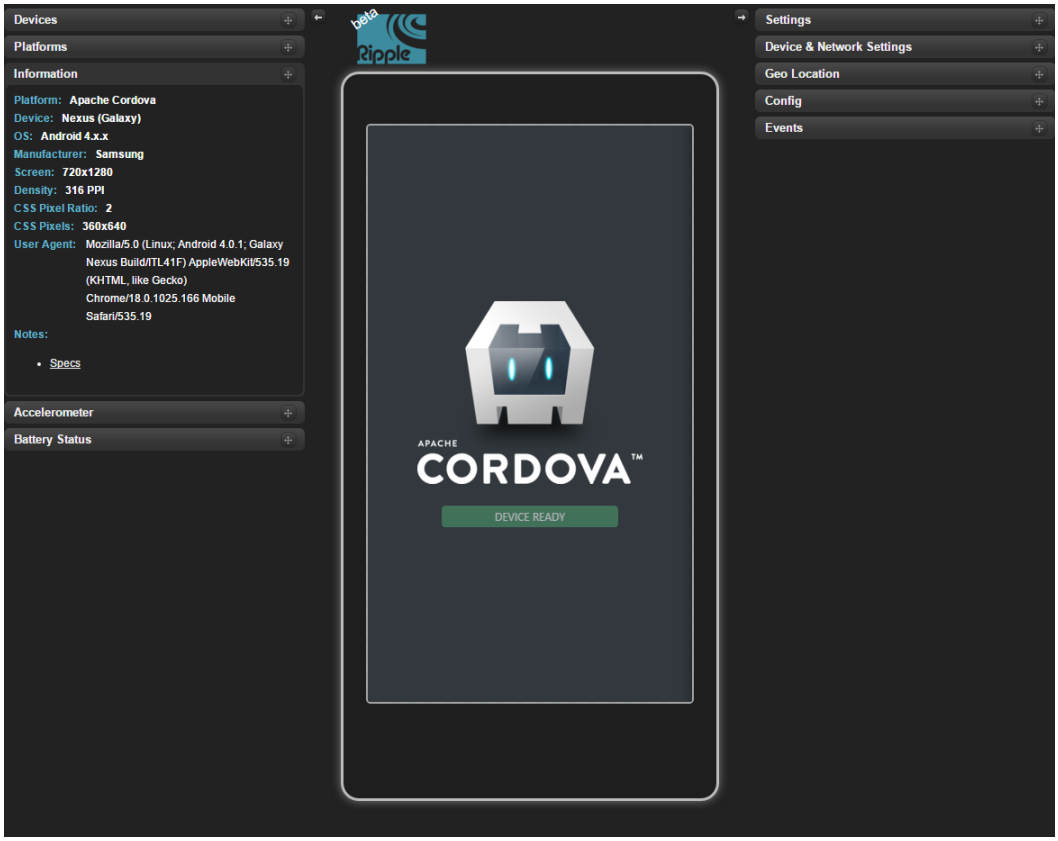
U gornjoj alatnoj traci izaberemo **Ripple-Nexus (Galaxy)** gumb kao na slici 5.4

Slika 5.4: Pokretanje aplikacije

Aplikacija će se otvoriti u Appache Ripple simulatoru i bit će prazna kao na slici 5.6 jer nismo još napisali nikakav kod. Appace Ripple besplatan je, jednostavan i brz simulator za testiranje Apache Cordova aplikacije za Android. Na lijevoj i desnoj strani Appace Ripple simulatora imamo mogućnost mijenjaja uređaja na kojem testiramo, servera, lokacije, orijentacije, interneta i slično. Pomoću njega možemo testirati HTML, CSS i dio JavaScript koda, to jest izgled aplikacije. Također ima podršku za neke plug-inove, geolokaciju i orijentaciju uređaja. Ako naša aplikacija koristi neke druge plug-inove onda ih se na ovaj način neće moći testirati, već treba testirati na uređaju ili nekom drugom emulatoru. O samim mogućnostima testiranja reći ćemo više u sljedećem poglavlju. Ako aplikacija ne koristi druge plug-inove moguće je testirati i kompletnu aplikaciju. Ipak treba imati na umu da nakon što smo potpuno napisali aplikaciju ipak je treba testirati na uređajima ili drugim emulatorima jer Appache Ripple koristi istu verziju web preglednika za testiranja.

Da bismo zaustavili testiranje aplikacije u gornjoj alatnoj traci kliknemo stop kao na slici 5.5

Slika 5.5: Pokretanje aplikacije

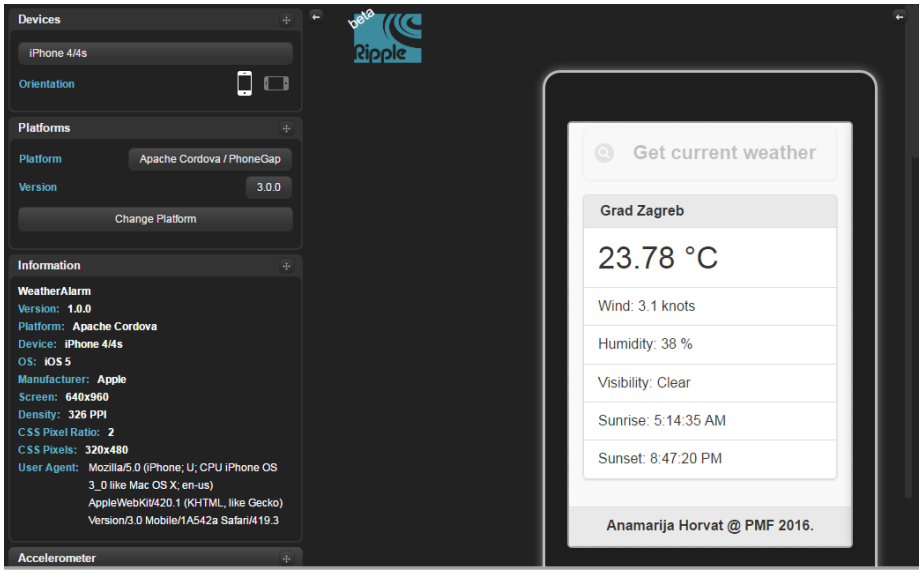


Slika 5.6: Pokretanje aplikacije

Poglavlje 6

Razvoj osnovnog dijela aplikacije

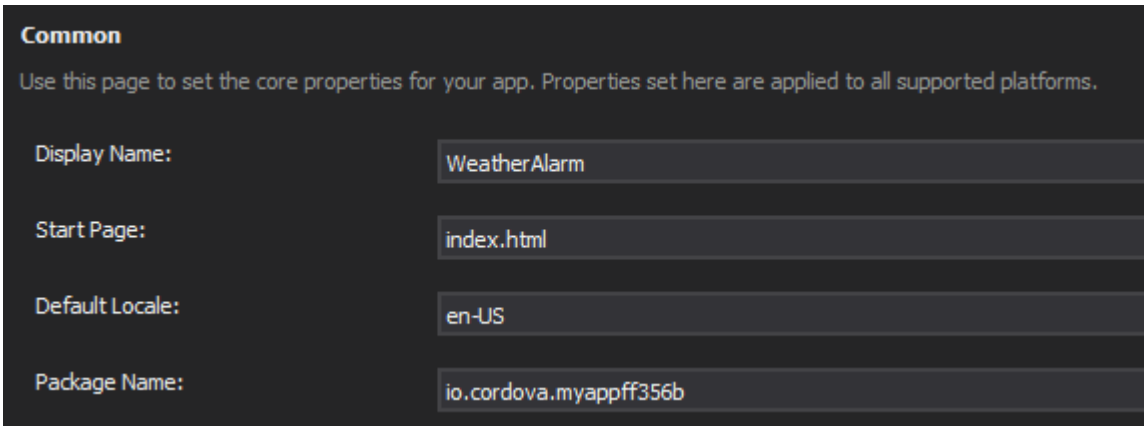
U ovom poglavlju ćemo objasniti razvoj osnovnog dijela aplikacije koju ćemo nazvati Weather Alarm tj. napraviti ćemo aplikaciju koja ima gumb na čiji klik se dohvaćaju i prikazuju podaci o trenutnoj vremenskoj prognozi na osnovi geolokacije uređaja. Ovaj dio aplikacije gotovo je identičan kao kod razvoja obične web stranice, razlikuje se samo u jednom malom dijelu, a to je da ćemo koristiti plug-in za geolokaciju. Aplikacija koju generiramo imat će na klik gumba izgled kao na slici 6.1. Primijetimo da smo ovaj puta aplikaciju testirali na IPhone-u, a ne na Androidu, naime koristeći Appache Ripple simulator može se testirati na ta dva operacijska sustava.



Slika 6.1: Izgled osnovnog dijela aplikacije

6.1 Osnovne postavke

Da bismo aplikaciji dali ime, te neke druge osnovne postavke, dvaput kliknemo na **config.xml**. Otvorit će se prozor kao na slici 6.2 gdje unesemo ime u prvo polje. Polja ima puno, no ostala se obično ispunjavaju nakon što je aplikacija napravljena, a neka su već ispunjena zadanim (defaultnim) vrijednostima.

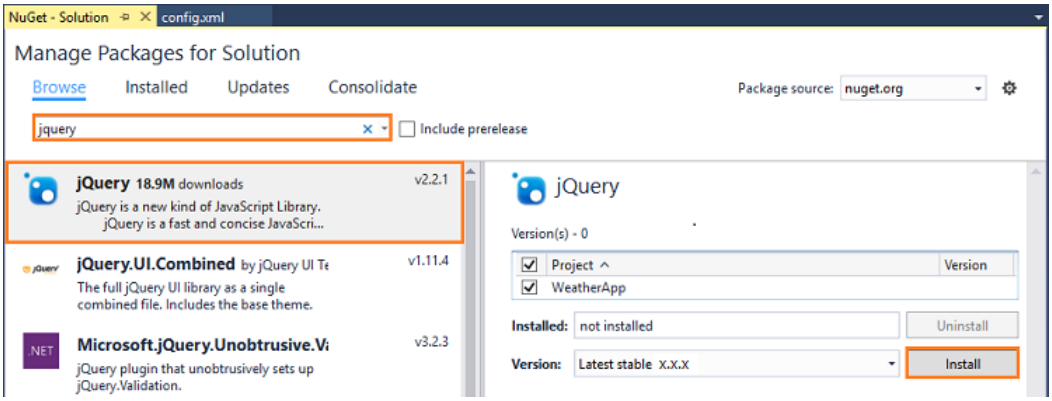


Slika 6.2: Davanje imena aplikaciji

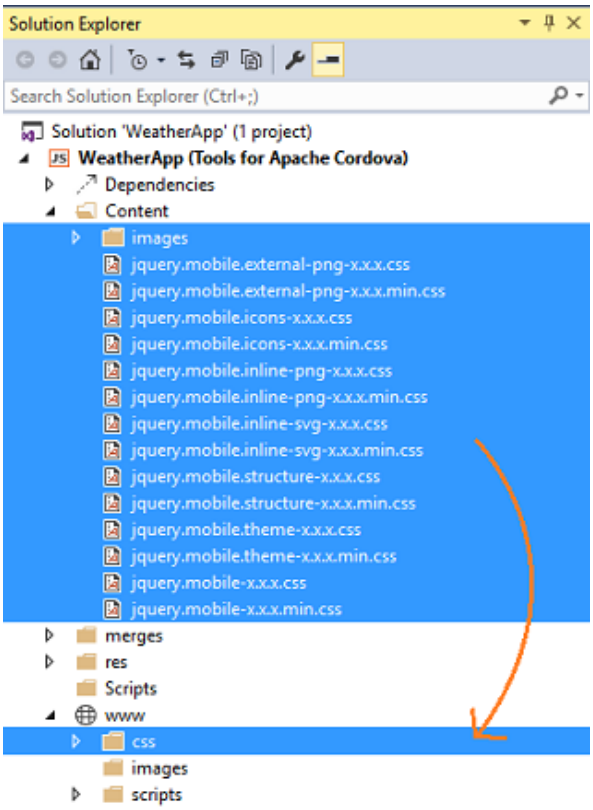
Dodavanja osnovnih alata

Kod aplikacije piše se koristeći HTML, CSS i JavaScript. S obzirom da je ovo mobilna aplikacija preporuča se koristiti JQuery Mobile jer uvelike ubrzava i olakšava količinu koda koji bismo trebali napisati bez njega za sam izgled i interkaciju s aplikacijom. Da bismo to učinili trebamo dodati JQuery i JQuery Mobile u naš projekt.

U gornjem meniju izaberemo **Tools ->NuGet Package Manager - >Manage NuGet Packages for Solution**, u novootvorenom prozoru pronađemo JQuery i instaliramo ga, kao na slici 6.3 Na isti način dodamo i JQuery Mobile u projekt. NuGet će kreirati dvije nove mape u projektu **Content** i **Scripts**, te u prvu staviti CSS datoteke, a u drugu JavaScript datoteke. To nije lokacija gdje ih želimo, sve što nama treba za razvoj aplikacije treba se nalaziti u **www** mapi, stoga ih moramo preseliti. Naprosto povučemo datoteke iz **Scripts** mape u **www - >scripts** i datoteke iz **Content** u **www - >css** kao na slici 6.4



Slika 6.3: Dodavanje JQuery-a u projekt



Slika 6.4: Stavljanje datoteka instaliranih pomoću NuGet-a na pravu lokaciju u projektu

Pomoću NuGet-a na ovaj način možemo dodati bilo koje druge razvojne okoline ili biblioteke koje nam trebaju.

Sada još trebamo njih uključiti u HTML datoteku, tj. u **index.html** koji se nalazi u **www** mapi. To je HMTL dio naše aplikacije, pisanje koda je isto kao kod razvoja web stranice. Znači pod <head> dodamo

```
<link rel="stylesheet" href="css/jquery.mobile-1.4.5.min.css" />
```

a pod <body>

```
<script src="scripts/jquery-2.2.3.min.js"></script>
<script src="scripts/jquery.mobile-1.4.5.min.js"></script>
```

gdje brojevi za verzije mogu biti drugačiji ovisno o tome koje smo instalirali.

6.2 HTML i CSS

Napravit ćemo dakle jedan gumb za dohvat vremenske prognoze ovisno o trenutnoj lokaciji i načinu prikaza podataka.

HTML

Sve što ćemo ovdje navesti treba raditi u **index.html**.

Prvo treba izbrisati dio koda koji je zadan (defaultan), a to je:

```
<div class="app">
  <p id="deviceready" class="event">Connecting to Device</p>
</div>
```

Sada ćemo dodati gumb i način prikaza podataka o vremenu. U <body> napišemo sljedeći kod:

```
<div data-role="page" id="weather-page">

    <div data-role="header" class="header">
        <h1 id="app-title">Weather Alarm</h1>
    </div>

    <div role="main" class="ui-content">
        <button id="get-weather-btn" data-role="button"
        data-icon="search">Get current weather</button>

        <ul id="weather-data" data-role="listview"
```

```
        data-inset="true"
        class="ui-listview ui-listview-inset ui-corner-all
        ui-shadow
        not-displayed">
            <li data-role="list-divider" id="title"
            class="ui-li-divider ui-bar-a"></li>
            <li><span id="summary"><span
            id="temperature"></span>&deg;C</span></li>
            <li>Wind: <span id="wind"></span> knots</li>
            <li>Humidity: <span id="humidity"></span> %</li>
            <li>Visibility: <span id="visibility"></span>
            </li>
            <li>Sunrise: <span id="sunrise"></span></li>
            <li>Sunset: <span id="sunset"></span></li>
        </ul>

        <div id="error-msg" class="not-displayed">
        </div>

    </div>

    <footer data-role="footer" data-position="fixed">
        <h4>Anamarija Horvat @ PMF 2016.</h4>
    </footer>

</div>
```

Trebat će nam neka JavaScript datoteka za dohvat vremena. U projekt ćemo je dodati kasnije, a sada samo dodamo na mjesto gdje su ostale skripte

```
<script src="scripts/weather.js"></script>
```

Također od nekud moramo dobiti podatke o vremenu, za to ćemo koristiti http://openweathermap.org/api - to je API koji postoji u besplatnoj verziji i verzijama koje se plaćaju za dohvaćanje podataka o vremenskoj prognozi. Koje sve mogućnosti API ima može se pročitati na toj stranici, a kako se koristi reći ćemo kasnije. Za sada trebamo dodati tu stranicu kao sigurnu u Content Security Policy dio koda kako bismo toj stranici mogli pristupiti iz aplikacije. U

```
<meta http-equiv="Content-Security-Policy" content="default-src
'self' data: gap:
https://ssl.gstatic.com 'unsafe-eval'; style-src 'self'
```

```
'unsafe-inline'; media-src *">
```

nakon `data: gap:` treba umetnuti `http://api.openweathermap.org`

CSS

Ovaj dio samo služi za editiranje izgleda. JQuery Mobile već jako dobro oblikuje elemente sam po sebi pa ne treba pisati puno koda. Ako smo pročitali HTML vidimo da se navodi klasa koju još nismo definirali `class="not-displayed"`. Želimo s njom sakriti poruku o greški i prikaz podataka o vremesnkoj prognozi jer se trebaju prikazati tek na klik gumba. Sadržaj **index.css**-a treba zamijeniti sljedećim kodom:

```
.not-displayed {
    display: none;
}

#get-weather-btn {
    font-size: 22px;
}

#title {
    font-size: 16px;
}

#summary {
    font-size: 35px;
}

#error-msg {
    text-align: center;
    margin-top: 50%;
    font-weight: bold;
}
```

6.3 JavaScript

Sada trebamo dohvatiti podatke i ubaciti ih u HTML pomoću JavaScript-a.

index.js

Primjetit ćemo da se sav kod nalazi unutar jedne funkcije.

```
(function () {
    "use strict";
    // sav kod se nalazi ovdje.
})();
```

To je funkcija koja se pokreće kada i aplikacija i sav potreban kod se piše unutar nje tj. unutar nje također se nalazi funkcija `onDeviceReady` u koju zapravo treba pisati sav kod. Ako bismo kod pisali izvan te funkcije onda se dijelovi uređaja poput orijentacije, akceleromtera tj. bilo kakvi hardverski odnosno nativni dijelovi ne bi mogli koristiti. Treba izbrisati iz nje zadan (defaultan) kod,

```
var element = document.getElementById("deviceready");
element.innerHTML = 'Device Ready';
element.className += ' ready';
```

te dodati događaj za klik gumba

```
$('#get-weather-btn').click(getWeatherWithGeoLocation);
```

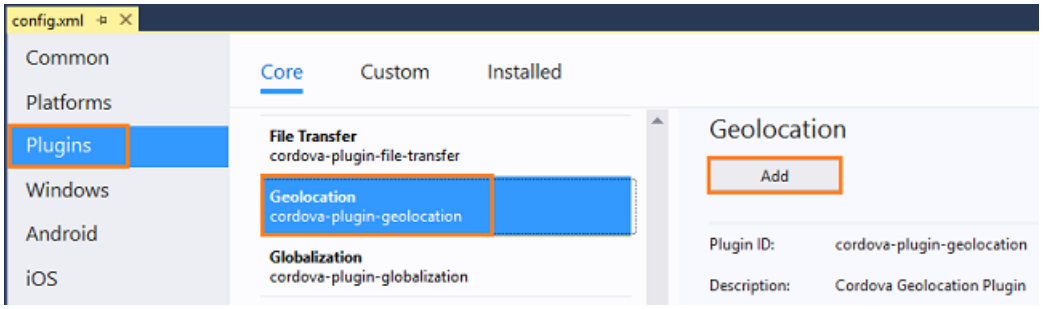
Znači na klik će se pozvati funkcija `getWeatherWithGeoLocation`, koju još nismo napisali. Napisat ćemo ju u **weather.js** datoteci koju treba prvo dodati u projekt. To učinimo tako da u Solution Explorer-u desno kliknemo na **www ->scripts** i odaberemo **Add->New JavaScript file**, nazovemo datoteku weather.js i klinemo **Add**.

weather.js

Da bismo dohvatili vrijeme već smo rekli koristit ćemo OpenWeather API, ključ se besplatno može nabaviti registracijom na adresi https://home.openweathermap.org/users/sign$_i$n. Na vrhu **weather.js** koda dodamo taj ključ,

```
var OpenWeatherAppKey = "Kljuc koji smo dobili";
```

Sada ćemo dodati plug-in za geolokaciju. Kliknemo dvaput na **config.xml** izaberemo **Plugins** pa **Core** pronađemo **Geolocation** te kliknemo **Add** kao na slici 6.5. Stvorit će se mapa **plugins** u našem projektu. Na ovaj način dodali bismo bilo koji službeni plug-in



Slika 6.5: Dodavanje službenih plug-inova

koje smo naveli u poglavlju 3. Mogućnosti i način korištenja službenih plug-inova nalazi se na web stranici Apache Cordove. Za ovaj plug-in dokumentacija se nalazi na adresi https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/index.html

Sada ćemo napisati funkciju `getWeatherWithGeoLocation`. Plug-in za geolokaciju se koristi na ovaj način:

```
navigator.geolocation.getCurrentPosition(onGetLocationSuccess,
onGetLocationError,{ enableHighAccuracy: true });
```

Prvi paramter je funkcija koje će se pozvati ako je lokacija uspješno dohvaćena, a drugi ako nije. Stoga da bi ova aplikacija radila lokacija mora biti omogućena na uređaju. Kasnije ćemo napraviti kod koji to provjerava jer je kompliciranije, a u ovom poglavlju radimo samo jednostavne stvari. Dakle ako je lokacija uspješno dohvaćena, dohvatit ćemo geografsku širinu i dužinu te poslati upit OpenWeather API-u te spremiti podatke te pozvati funkciju za prikaz istih koju još nismo napisali. Ako nije, ispisat ćemo poruku o grešci. Kod izgleda ovako:

```
function getWeatherWithGeoLocation() {

  navigator.geolocation.getCurrentPosition(onGetLocationSuccess,
  onGetLocationError,
    { enableHighAccuracy: true });

  $('#error-msg').show();
  $('#error-msg').text('Determining your current location ...');

  $('#get-weather-btn').prop('disabled', true);
}
function onGetLocationSuccess(position) {
```

```
var latitude = position.coords.latitude;
var longitude = position.coords.longitude;

var queryString =
  'http://api.openweathermap.org/data/2.5/weather?lat='
    + latitude + '&lon=' + longitude +
    '&appid=' + OpenWeatherAppKey + '&units=imperial';

$('#get-weather-btn').prop('disabled', false);

$.getJSON(queryString, function (results) {

    showWeatherData(results);

    }).fail(function (jqXHR) {
        $('#error-msg').show();
        $('#error-msg').text("Error retrieving data. " +
        jqXHR.statusText);
});

}
```

Open Weather API vraća podatke u JSON formatu. To je najčeši oblik spremanja podataka koji se bazira na atributu i vrijednosti. Što sve API vraća može se pročitati na adresi http://openweathermap.org/currentgeo. Mi ćemo napisati funkciju koja prikazuje podatke koje smo definirali u **index.html**. Podaci će se prikazati na dnu aplikacije pa ćemo dodati i kod koji stavi pogled na dno aplikacije (zbog uređaja sa malim ekranima).

```
function showWeatherData(results) {
        if (results.weather!=undefined) {

            $('#error-msg').hide();
            $('#weather-data').show();

            $('#title').text(results.name);
            $('#temperature').text(results.main.temp+"  ");
            $('#wind').text(results.wind.speed);
            $('#humidity').text(results.main.humidity);
            $('#visibility').text(results.weather[0].main);
```

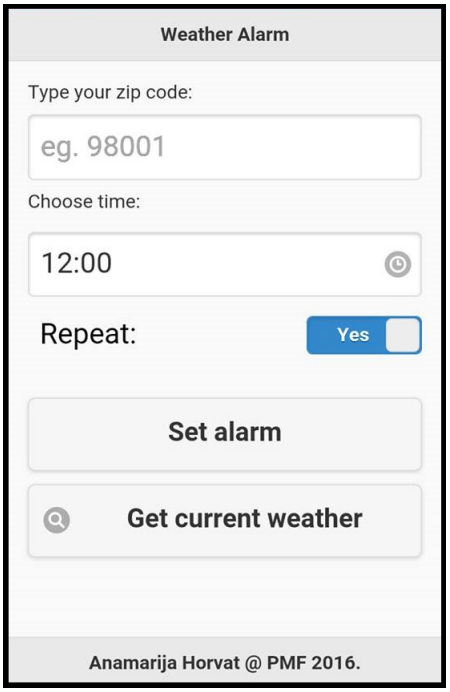
```
        var sunriseDate =
        new Date(results.sys.sunrise * 1000);
        $('#sunrise').text(sunriseDate.toLocaleTimeString());

        var sunsetDate =
        new Date(results.sys.sunset * 1000);
        $('#sunset').text(sunsetDate.toLocaleTimeString());
        $('html,body').animate({ scrollTop:
        document.body.scrollHeight }, "fast");

        } else {
            $('#weather-data').hide();
            $('#error-msg').show();
            $('#error-msg').text("Error retrieving data. ");
            $('html,body').animate({ scrollTop:
            document.body.scrollHeight }, "fast");
        }
    }
```

S ovime smo napravili prvu aplikaciju u Apache Cordovi, kao na slici 6.1 s početka poglavlja. Iz ovog poglavlja se vidi da se razvoj aplikacije ne razlikuje puno od razvoja web stranice. To je zato jer nismo pristupali nativnim dijelovima mobitela kojima inače web stranice ne mogu pristupiti. Apache Cordova je stoga odličan alat za razvoj aplikacija tog tipa jer je to zapravo web stranica zapakirana u aplikaciju, a danas će većina ljudi prije koristiti aplikaciju nego web stranicu. U sljedećem poglavlju nadogradit ćemo ovu aplikaciju tako da pristupa nativnim dijelovima mobitela.

Poglavlje 7

Razvoj napredne aplikacije

U ovom poglavlju ćemo napraviti aplikaciju koja ima mogućnost postavljanja alarma koji će na odabrano vrijeme ”probuditi” korisnika i poslati obavijest o vremenu u mjestu čiji poštanski broj smo unijeli. Aplikacija će imati i dijelove navedene u prethodnom poglavlju. Konačan izgled aplikacije bit će kao na slici 7.1



Slika 7.1: Postavke za Windows

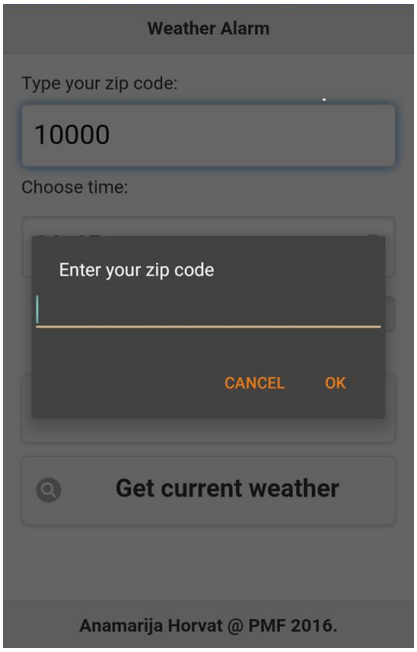
S obzirom da kodiranja ima puno nećemo pisati kod već dijelove koda koji su potrebni za razumijevanje načina izrade aplikacije u Apache Cordova. Također nećemo detaljno pisati što treba napraviti za određene stvari jer je to bilo objašnjeno u ranijim poglavljima.

7.1 Razlike u HTML-u, JavaScript-u i CSS-u

Aplikacija napravljena u Apache Cordovi se zapravo pokreće u web pregledniku mobitela na kojem je instalirana. To znači da ovisno je li s aplikacijom ciljamo Android, iPhone ili Windows Phone sav HTML, JavaScript i CSS kod treba biti podržan na Google Chrome, Safari ili Internet Explorer. Za većinu stvari može se provjeriti na adresi http://www.w3schools.com gdje su određeni elementi podržani.

Dijalozi

Dosta često se u aplikacijama koriste dijalozi gdje korisnik potvrđuje unos ili dobije obavijest o radnji koju je napravio. To ćemo koristiti i ovdje, kao na slici 7.1. Prvo polje je polje za unos poštanskog broja. Na klik polja iskočit će dijalog kao na slici 7.2, to je najobičniji Prompt box JavaScript-a.



Slika 7.2: Unos poštanskog broja

Postoji službeni plug-in za dijaloge, dokumentacija se nalazi na adresi https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-dialogs/index.html. Kada god postoji službeni plug-in treba se on koristiti jer obično postoji razlog zašto je bio kreiran. U ovoj aplikaciji nije korišten jer je prilikom testiranja nađena greška. Naime slova Prompt box-a su bijela na bijeloj pozadini kod Android uređaja i uopće se ne vide. Problem je adresiran pod brojem CB-8292 na službenoj stranici Apache Cordove i može se pratiti na adresi https://issues.apache.org/jira/browse/CB-8292.

Znači prvo smo pokušali koristiti službeni plug-in nakon što je zaključeno da to ne radi sljedeći korak je bio pogledati ima li jQuery Mobile ima alternativu. jQuery Mobile se uvijek treba koristiti prije običnog JavaScripta jer je prilagođen mobitelima. Alternativa postoji, zove se popup i način korištenja se može naći na adresi http://demos.jquerymobile.com/1.4.2/popup/ui-state=dialog. U ovoj aplikaciji to nije radilo na Androidu, naime dijalog se ne bi prikazivao. Čitajući po forumima izgleda da na dosta web preglednika mobitela dijalog se ili ne prikazuje, ili se prikaže i odmah sakrije i slično. Iz tog razloga ni ovaj način nije bilo moguće umetnuti dijaloge u aplikaciju stoga se koristio JavaScript.

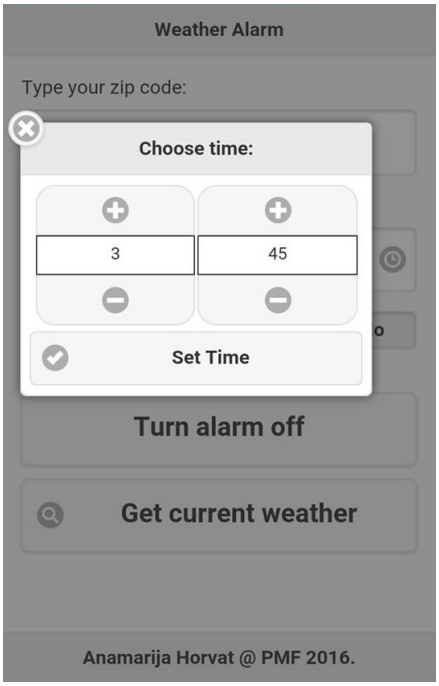
TimePicker

Sljedeće polje u aplikaciji je polje za odabir vremena takozvani TimePicker. To je input na čiji se klik treba otvoriti dijalog na kojem se može odabrati vrijeme. Postoji puno JavaScript plug-inova koji ga implementiraju. Instalirala sam par njih i koji god sam isprobala nije radio, nije se pojavljivao dijalog za odabir vremena. Čitajući onda kodove plug-inova shvatila sam da svi koriste događaj za klik umjesto događaja za tap. Naime neki mobilni web preglednici događaj na klik neće registrirati. Stoga prilikom korištenja JavaScript plug-inova treba provjeriti jesu li prilagođeni mobilnim web preglednicima.

TimePicker koji radi je dio API-ja JTSageDateBox koji se može nabaviti na adresi http://dev.jtsage.com/DateBox/builder/. Tamo treba selektirati **jQueryMobile** i **DateBox/TimeBox/DurationBox** te kliknuti **Go!**. Preuzet će se zip dokument čije CSS i JavaScript datoteke trebamo dodati u naš projekt ručno kopiranjem na prava mjesta **www** mape te ih uključiti u **index.html**. Detaljna dokumentacija API-ja može se pročitati na adresi http://dev.jtsage.com/DateBox/api/. Za potrebe ove aplikacije potreban je samo TimePicker, kod za njega u ovoj aplikaciji je;

```
<input type="text" name="timebox" id="timebox"
data-role="datebox" data-options='{"mode":"timebox",
"timeFormatOverride":"%H",
"defaultValue":"12:00", "showInitialValue":"true"}'>
```

Ovim kodom generirali smo zapravo input box na čiji se klik otvori dijalog kao na slici 7.3



Slika 7.3: TimePicker

CSS

Sljedeća stvar koja se nalazi u aplikaciji je područje gdje se bira želimo li da se alarm ponavlja ili ne. Sastoji se od labele lijevo i flipswitch gumba desno. Pozicioniranje dva elementa koji nisu fiksne dužine u isti red tako da je jedan lijevo a drugi desno radi se npr. na način da se svaki stavi u poseban , oni pak u jedan , te se **float** vrijednost jednog unutarnjeg postavi na **left** odnosno **right**. No taj postupak nije dao dobar izgled kakav je na slici 7.1. Poznajem još dva načina koji bi trebali dati takav rezultat no ni oni nisu rezultirali uspjehom. Problem je na kraju bio što je CSS od jQuery Mobile-a rušio moj. U većini slučajeva ako nam CSS ne daje rezultate kakve želimo, treba pregledati CSS od jQuery Mobile-a i nadglasati ga ako vidimo da daje vrijednosti svojstvima kojima želimo dati vrijednost. To se radi tako da se doda **!important** kod klasa ili elemenata u CSS-u koje mi pišemo a jQuery Mobile je definirao drugačije. U **index.css** treba dakle dodati:

```
#timebox{
    font-size:24px;
}
#labelrepeat{
```

```
    display:inline-block;
}
.ui-mobile label.lablsett {
    left: 0 !important;
    float: left;
    margin-right: 10px;
    margin-left: 10px;
    margin-top:0px;
    font-size:24px;
    color: #000;
}
.optContainer .ui-flipswitch {
    float: right;
    margin-right: 10px;
}
.container{
    border-width: 0;
        overflow: visible;
        overflow-x: hidden;
        padding-bottom:1em;
}
.flipswitch{
    position: absolute; right: 0;
}
.ui-field-contain, .ui-mobile fieldset.ui-field-contain {
    border-width: 0 !important;
    padding-top: 0 !important;
    margin: 1em 0 !important;
  }
```

On definira poziciju ta dva elementa i TimePicker-a tj. sljedećeg HTML koda:

```
<div class="container">
    <input type="text" name="timebox" id="timebox"
    data-role="datebox" data-options='{"mode":"timebox",
    "timeFormatOverride":"%H", "defaultValue":"12:00",
    "showInitialValue":"true"}'>
    <div data-role="fieldcontain" class="optContainer">
        <label for="repeat" class="lablsett">Repeat:</label>
        <select id="fliprepeat" data-role="flipswitch"
        class="flipswitch">
```

```
            <option value="no">No</option>
            <option value="yes">Yes</option>
        </select>
    </div>
</div>
```

Brojač vremena

Sljedeća stavka u aplikaciji je gumb na čiji klik bi se trebalo odbrojiti vrijeme do unesenog i zatim aktivirati alarm. JavaScript ima dvije funkcije za mjerenje vremena,
`setTimeout(imefunkcije, millisekunde)`
i `setInterval(imefunkcije, millisekunde)`.
No što se događa s aplikacijom kada se ugasi ili se stavi u pozadinu? Slično što se događa sa web stranicom kada ju ugasimo, brojač stane. Brojač nastavi dalje brojati vrijeme tek kada aplikacija opet postane aktivna. U slučaju web stranice brojač bi se restirao. To znači da ne možemo mjeriti vrijeme osim ako aplikacija nije aktivna cijelo vrijeme. Ne postoji trenutno nijedan plug-in koji samo broji vrijeme. Postoji plug-in koji sprječava mirovanje aplikacije, nalazi se na adresi https://github.com/katzer/cordova-plugin-background-mode. No sprječavanje mirovanja aplikacije znači da će jako brzo potrošiti bateriju mobitela i time aplikacija zapravo postaje beskorisna. Rješenje ovog problema trenutno ne postoji. Djelomično ćemo riješiti problem u sljedećem poglavlju koristeći jedan custom plug-in.

Lokalna pohrana

Aplikacija trenutno ne čuva promjene, npr. ako unesemo poštanski broj, zatvorimo aplikaciju te ponovno otvorimo to polje će biti prazno. To je zato jer se ponaša kao web stranica. Da bismo sačuvali promjene trebamo ih lokalno pohraniti i to nam omogućuje HTML 5. Način korištenja je sljedeći:

```
var storage = window.localStorage;
var value = storage.getItem(key); // dohvaca vrijednost kljuca
storage.setItem(key, value) // postavlja vrijednost kljucu
storage.removeItem(key) // brise kljuc
```

Tako npr. za dohvaćanje poštanskog broja u **index.js** treba dodati

```
var zipstore = window.localStorage.getItem("zip");
                    if (zipstore != null)
                        $('#zip-code-input').val(zipstore);
```

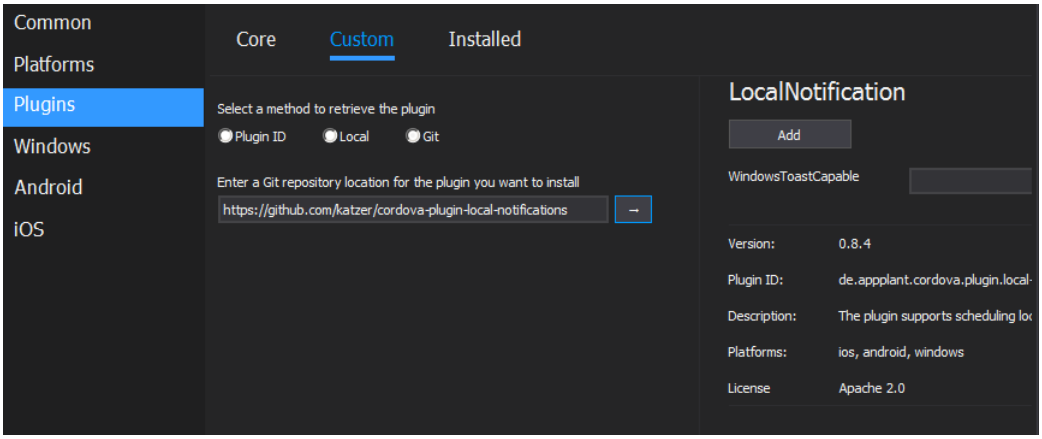
Svaku promjenu svakog polja i gumba treba dakle spremiti i dohvatiti vrijednosti u **index.js**. Polja i promjena će biti puno te ih nećemo ovdje detaljno objašnjavati jer je princip svugdje isti.

7.2 Plugins

U ovom poglavlju koristit ćemo plug-inove koje nisu službeni već su ih napravili neki ljudi i stavili ih na internet za besplatno korištenje.

Obavijesti

U prethodnom poglavlju rekli smo da ne postoji trenutno način za brojanje vremena. Postoji doduše plug-in za obavijesti, on broji vrijeme no kad istekne prikaže obavijest. Koristit ćemo njega. Nalazi se na GitHub-u, na adresi https://github.com/katzer/cordova-plugin-local-notifications. Na toj adresi su i linkovi na dokumentaciju i primjere korištenja. Da bismo ga dodali u projekt, u **config.xml** pod **Plugins** selektiramo **Custom** zatim **Git** te ispod upišemo web adresu gdje se nalazi, kliknemo strelicu pa desno **Add**, kao na slici 7.4



Slika 7.4: Dodavanje custom plug-ina u projekt

U aplikaciji imamo gumb **Set alarm**. Kada se klikne prvo trebamo izračunati vrijeme koje mora proći do aktivacije alarma. To jest alarm se treba aktivirati na vrijeme koje je uneseno u TimePicker-u. To radi sljedeći kod:

```
var trrigerTimeinms = $('#timebox').val().split(":");
```

```
        var now = new Date();
        var day = now.getDate();
        if (now.getHours() > parseInt(trrigerTimeinms[0]))
            day = day + 1;
        else if ((now.getHours() ==
        parseInt(trrigerTimeinms[0])) &&
        (now.getMinutes() > parseInt(trrigerTimeinms[1])))
            day = day + 1;
        var time = new Date(
            now.getFullYear(),
            now.getMonth(),
            day,
            parseInt(trrigerTimeinms[0]),
            parseInt(trrigerTimeinms[1]), 0
        );
```

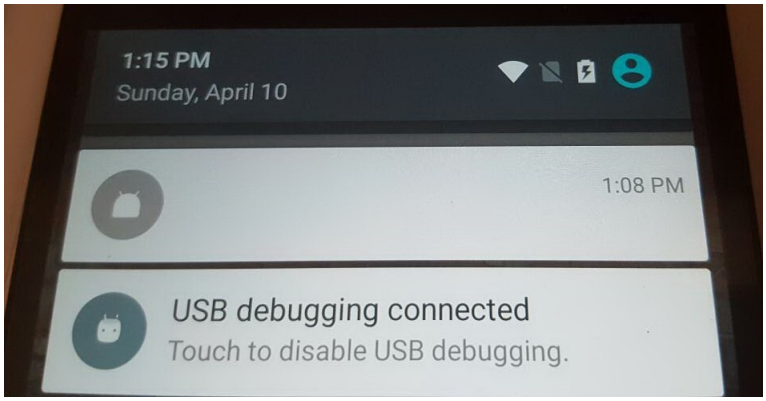
Sada treba provjeriti ponavlja li se alarm ili ne, jer će o tome ovisiti kod za obavijesti.

```
if ($('#fliprepeat').val() == "yes")
            var every = "day";
        else
            var every = 0;
```

Brojat ćemo vrijeme na način da ćemo kreirati praznu obavijest.

```
 cordova.plugins.notification.local.schedule({
            id: 1,
            firstAt: time,
            every: every,
            sound: null
        });
```

Nažalost na Androidu prazna obavijest nije skroz prazna, naime ima ikonu bez teksta kao na slici 7.5.



Slika 7.5: Prazna obavijest

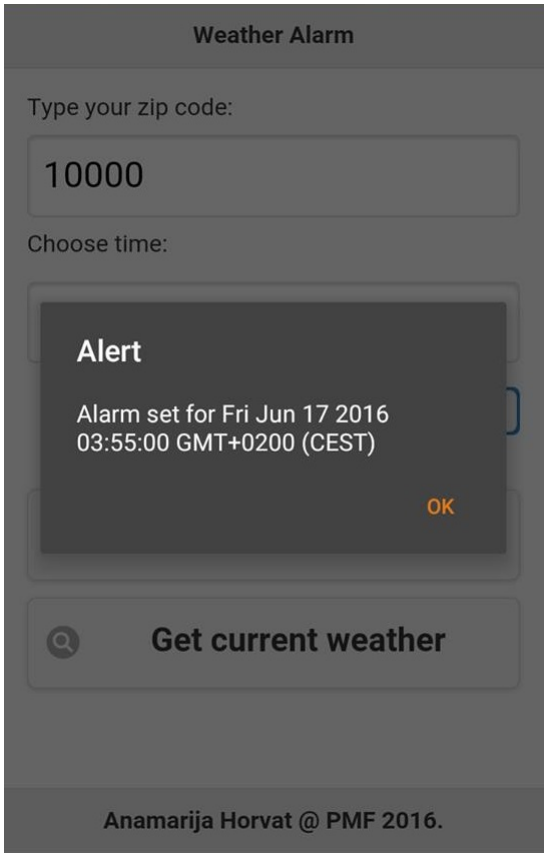
Nemoguće ju je maknuti pa ćemo to riješiti na način da ćemo ju sakriti čim se pojavi. Također kada se aktivira treba prikazati obavijest o trenutnoj vremenskoj prognozi pa ćemo pozvati funkciju koja je zadužena za to. Logično bi bilo da prazne obavijesti nema već se odmah pokaže obavijest s vremenskom prognozom. To je nemoguće napraviti jer nama prva obavijest nije zapravo obavijest već samo služi za mjerenje vremena, a tek kada smo dosegnuli vrijeme treba dohvatiti podatke o vremenskoj prognozi. Kod je dakle sljedeći

```
cordova.plugins.notification.local.on(
"trigger", function (notification) {
            if (notification.id == 1) {
                clear();
                getWeatherWithZipCode();
            }
        });
```

Sada je alarm postavljen te treba sakriti gumb za postavljanje i prikazati gumb za gašenje alarma, te također prikazati obavijest korisniku kada će se alarm aktivirati.

```
$('#setalarm').hide();
$('#deletealarm').show();
alert("Alarm set for " + time);
```

Obavijest koja se prikaže izgledat će kao na slici 7.6



Slika 7.6: Obavijest da je alarm postavljen

Sada u aplikaciji postoji gumb s tekstom **Turn alarm off**. Na klik njega treba otkazati alarm te prikazati opet gumb za postavljanje alarma. Funkciju za micanje obavijesti smo nazvali `clear`, a za otkazivanje ćemo ju nazvati `cancel` i njihov kod je sljedeći:

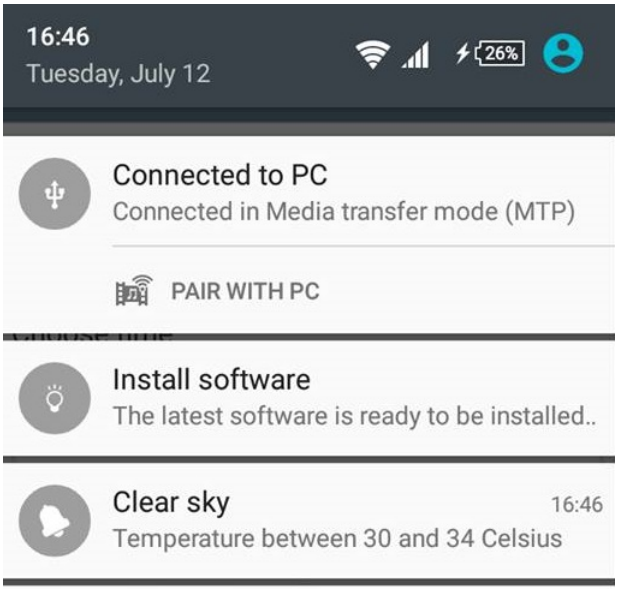
```
function clear() {
        cordova.plugins.notification.local.clearAll(function () {
        }, this);
    }

    function cancel() {
        cordova.plugins.notification.local.cancelAll(function () {
        }, this);
    }
```

Zatim treba dohvatiti vrijeme na osnovi unesenog poštanskog broja, to je slično kao u prethodnom poglavlju za dohvaćanje pomoću lokacije pa nećemo ovdje pisati kod. Bitna razlika je što nećemo rezultate prikazati u **index.html** već u obliku obavijesti, tj. pozvat ćemo funkciju `showNotification` umjesto `showWeatherData`.

```
function showNotification(results) {
        if (results.weather != undefined) {
            $('#error-msg').hide();
            var name =
            results.weather[0].description.substr(0, 1).toUpperCase()
            + results.weather[0].description.substr(1);

            cordova.plugins.notification.local.schedule({
                title: name,
                message: "Temperature between " +
                results.main.temp_min + " and " +
                results.main.temp_max + " Celsius",
                sound: "file://sounds/Kalimba.mp3"
            });
        }
        else {
            $('#weather-data').hide();
            $('#error-msg').show();
            $('#error-msg').text("Error retrieving data. ");
            $('html,body').animate(
            { scrollTop: document.body.scrollHeight }, "fast");
        }
    }
```

Primijetimo da smo koristili stazu za zvuk. Treba u **www** mapi kreirati mapu **sounds** i unutra staviti datoteku **Kalimba.mp3**.

Na aktivaciju alarma će se dakle prvo pojaviti prazna obavijest koja će odmah nestati, zatim će se pojaviti obavijest s vremenskom prognozom kao na slici 7.7 te će svirati datoteka **Kalimba.mp3**. Aplikacija je s ovim skoro gotova, treba još samo dodati neke sitne funkcionalnosti.



Slika 7.7: Obavijest o trenutnoj vremenskoj prognozi

Dohvaćanje koda zemlje

U aplikaciji koristimo poštanski broj, no poštanski brojevi su drugačiji za svaku zemlju. Trebamo na neki način saznati na koju zemlju se misli. Napravit ćemo to na način da ćemo dohvatiti kod zemlje distributera SIM kartice koja se nalazi u uređaju. Postoji plug-in za to i nalazi se na adresi https://github.com/pbakondy/cordova-plugin-sim. Treba ga dodati u projekt na isti način kao što smo dodali plug-in za obavijesti. U funkciju za klik gumba za postavljanje alarma treba dohvatiti kod zemlje

```
window.plugins.sim.getSimInfo(zipsuccess, ziperror);
```

Parametri su dvije funkcije, ako je uspio dohvatiti i ako nije. Ako je uspio, pohranit ćemo kod zemlje u lokalnu pohranu,

```
function zipsuccess(result){
        window.localStorage.setItem("countrycode",
        result.countryCode);
    }
```

ako nije ispisat ćemo poruku o grešci

```
function ziperror() {
        $('#error-msg').show();
```

```
        $('#error-msg').text("Error retrieving country
        code using us instead. ");
        $('html,body').animate({ scrollTop:
        document.body.scrollHeight }, "fast");
    }
```

U **index.js** treba postaviti inicijalnu vrijednost na us,

```
var countrycode = window.localStorage.getItem("countrycode");
if (countrycode == null)
        window.localStorage.setItem("countrycode", 'us');
```

Prilikom slanja upita za dohvaćanje vremenske prognoze pomoću poštanskog broja treba dohvatiti kod zemlje iz lokalne pohrane

```
var zipcode = $('#zip-code-input').val();
        var country = window.localStorage.getItem("countrycode");
        var queryString =
            'http://api.openweathermap.org/data/2.5/weather?zip='
            + zipcode + ',' + country + '&appid=' +
            OpenWeatherAppKey + '&units=metric';
```

Provjera je li na mobitelu omogućen pristup lokaciji

Ova aplikacija zahtijeva pristup lokaciji u jednom svom dijelu, za ostatak je potreban pristup internetu. Pristup internetu kod aplikacija je uobičajen no pristup lokaciji nije i većina ljudi nema stalno omogućenu lokaciju jer s time troše više baterije. Zbog toga kada korisnik želi dohvatiti vremensku prognozu na trenutnom mjestu treba provjeriti je li lokacija omogućena na uređaju. Postoji plug-in za to i nalazi se na adresi
https://github.com/dpa99c/cordova-diagnostic-plugin.
Svi dosadašnji plug-inovi bili su podržani za Android, iOS i Windows Phone 8 te je način korištenja bio isti kod svih. Ovaj nije, podržan je za Android, iOS i Windows Phone 10. Osoba koja ga je napravila ne namjerava ga raditi za starije verzije Windows Phone-a, a pitanje je hoće li ljudi koji su radili ostale plug-inove nadograditi plug-in da podržava novije verzije. To je glavni problem s razvojem Apache Cordova aplikacija. Platforme se stalno mijenjaju, a stvari koje koristimo se sporo prilagođavaju novim platformama ili se uopće ne prilagode i postanu neupotrebljive.

Osim što ovaj plug-in ne podržava Windows Phone 8 način upotrebe za Android i iOS se razlikuju. Stoga u aplikaciji moramo detektirati na kojoj platformi je instalirana i napisati drugačiji kod za svaku. Za to nam treba službeni plug-in Device i trebamo ga dodati u projekt. Koristi se na način:

```
if(device.platform=='Android') //napravi nesto
```

Kod Androida i iOS-a možemo provjeriti je li lokacija omogućena, no kod Android uređaja postoje 3 varijante. Lokacija s visokom preciznošću, lokacija koja koristi samo GPS i lokacija koja koristi samo internet za dohvaćanje lokacije. Ostalo u čem se plug-in razlikuje za ove dvije platforme je da kod Androida ako lokacija nije omogućena možemo prebaciti aplikaciju na postavke lokacije, a kod iOS-a to nije moguće. Stoga ćemo promijeniti funkciju `getWeatherWithGeolocation`. Njen kod ćemo prebaciti u novu funkciju `locationenabled`, te ga izbrisati i zamijeniti sljedećim kodom:

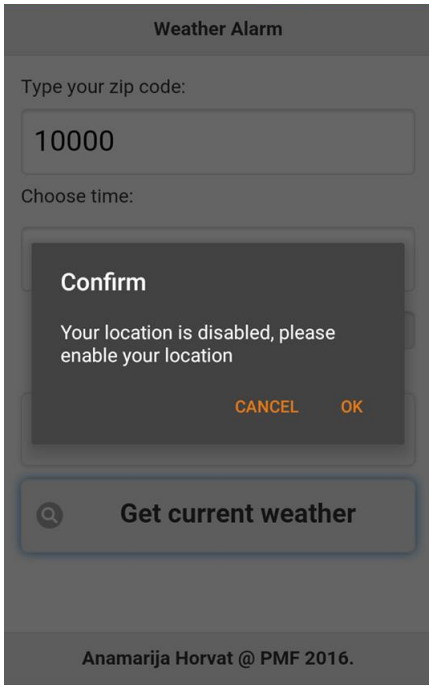
```
if (device.platform == 'Android') {
    cordova.plugins.diagnostic.isLocationEnabled(
    function (enabled)
    {
        cordova.plugins.diagnostic.getLocationMode(
        function (locationMode) {
            switch (locationMode) {
                case
                cordova.plugins.diagnostic.
                locationMode.LOCATION_OFF:
                enable();
                break;
                default: locationenabled();
            }
    }, function (error) {

    });
    }, function (error) {

    });

}
 else if (device.platform == 'iOS') {
    cordova.plugins.diagnostic.isLocationEnabled(
    function (enabled) {
        if (enabled == "disabled")
            alert("Your GPS is disabled,
            please enable it and try again");
        else
            locationenabled();
    }, function (error) {
```

```
});
        }
        else
            locationenabled();
```

Te napisati još jednu funkciju,

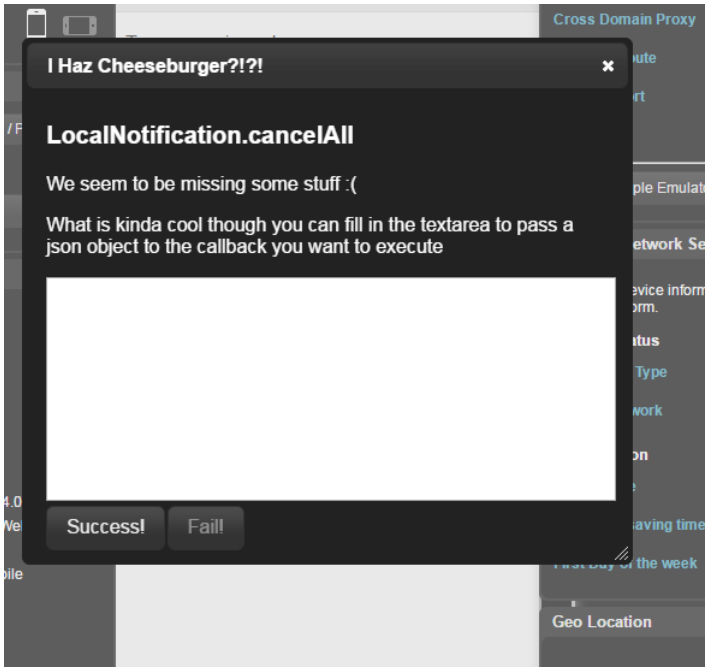
```
function enable()
    {
        var location = confirm("Your location is disabled,
        please enable your location");
        if(location==true)
            cordova.plugins.diagnostic.switchToLocationSettings();
    }
```

S ovim smo napravili provjeru je li lokacija omogućena na iOS i Android uređaju, te kod Android-a ako nije izaći će poruka kao na slici 7.8, gdje ako korisnik klikne **ok** će ga prebaciti na postavke lokacije.



Slika 7.8: Dijalog koji se pojavi ako lokacija nije omogućena kod Android uređaja

S ovime smo dovršili izradu aplikacije. Ako smo je pokušali testirati na Apache Ripple emulatoru aplikacija nije radila već su se pojavljivale poruke slične kao na slici 7.9. To je zato što se plug-inovi na njemu ne mogu testirati. U sljedećem poglavlju ćemo reći kako i na koje sve načine možemo testirati aplikaciju.



Slika 7.9: Testiranje plug-inova na Apache Ripple simulatoru

Poglavlje 8

Testiranje aplikacije

Aplikaciju koju smo napravili već smo testirali na Apache Ripple simulatoru. Vidjeli smo da se plug-inovi tamo ne mogu testirati stoga ju moramo testirati na drugim simulatorima i na uređajima. Neki plug-inovi se uopće ne mogu testirati na simulatorima, no to se ne može se saznati prije pokušaja. Sada ćemo objasniti kako se testira aplikacija za različite platforme.

8.1 Testiranje aplikacije na iOS-u

Aplikacije za iOS mogu se testirati pomoću Apache Ripple simulatora što je objašnjeno kod izrade aplikacije, na iOS simulatoru što zahtijeva Mac ili Mac u oblaku i mobilnom uređaju. Zbog financijskih mogućnosti aplikaciju nisam mogla testirati na druga dva načina no reći ću nešto o njima. Za oba je potrebno prvo instalirati sav potreban dodatni software na Mac-u kao što je objašnjeno u poglavlju 4.2. Ako to još nismo učinili trebamo učiniti sada.

iOS simulator

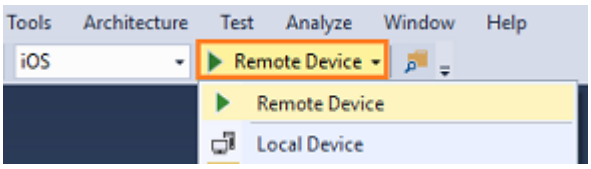
Treba prvo pokrenuti Remote Agent. Kako se to radi može se pročitati na adresi http://taco.visualstudio.com/en-us/docs/ios-guide/remoteAgent. Testiranje na simulatoru slično je kao kod Windows Phone-a, samo pod **Solution Platforms** izaberemo **iOS** i zatim desno neki simulator poput na primjer **Simulator-iPhone 5** i pritisnemo zelenu strelicu za pokretanje, no bitna razlika je da će se simulator prikazati na Mac-u a ne na Windows računalu s kojeg smo pokrenuli aplikaciju.

Provisioning profile

Kako bismo mogli testirati aplikaciju na uređaju treba nam ”provisioning profile”. Postoje tri mogućnosti kako ga možemo napraviti: koristeći Apple Id, Apple Developer račun ili koristeći od nekog drugog koji ima timski ”provisioning profile”. Apple Developer račun se plaća 99 dolara godišnje. Kako kreirati ”provisioning profile” može se saznati na adresi http://taco.visualstudio.com/en-us/docs/ios-guide/create-a-provisioning-profile.

Pokretanje aplikacije na uređaju spojenom na Mac

Mobilni uređaj mora biti uključen i spojen na Mac, te Remote agent mora biti pokrenut na Mac-u. U Visual Studiju, u projektu gdje je aplikacija, u gornjoj alatnoj traci izaberemo **iOS** pa desno **Remote Device** te kliknemo zelenu strelicu kao na slici 8.1



Slika 8.1: Pokretanje aplikacije na uređaju spojenom na Mac

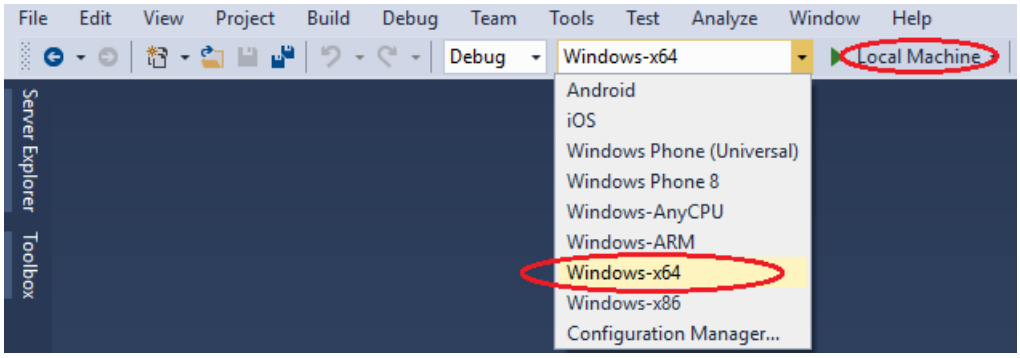
Pokretanje aplikacije na uređaju spojenom na Windows računalo

Za ovaj način potreban je Apple Developer račun. Često se koristi u tvrtkama gdje postoji samo jedan Mac za testiranje a aplikacije se razvijaju na Windows računalima. O ovom načinu testiranja može se pročitati na adresi http://taco.visualstudio.com/en-us/docs/ios-guide/local-deploy.

8.2 Testiranje aplikacije na Windows i Windows Phone

Windows

U aplikaciji treba otvoriti config.xml, izabrati **Windows** te zatim selektirati **Windows 8.1** ili **Windows 10**, ovisno o ciljnoj platformi. Nakon toga u gornjoj alatnoj traci na popisu **Solution Platforms** treba izabrati neku Windows platformu poput **Windows-x64**. Vidi sliku 8.2 Može se izabrati i **Windows-Any CPU** no to se ne preporuča. Desno od **Solutions Platforms** pod ”Run” izaberemo **Local Machine** ili **Simulator** ovisno želimo li testirati

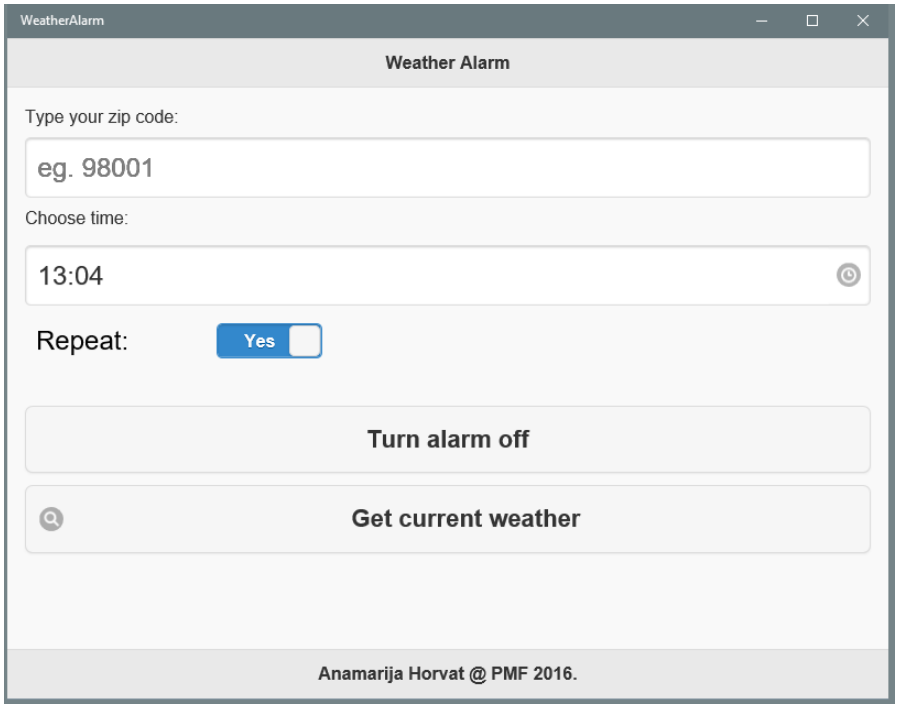


Slika 8.2: Postavke za Windows

na računalu na kojem radimo ili na simulatoru. Kliknemo F5 ili zelenu strelicu za pokretanje. Ako smo izabrali računalo otvorit će nam se poruka i u njoj link za omogućivanje ”development mode”, nakon što smo ga omogućili pristisnemo F5 još jednom. Nakon toga prikazat će se naša aplikacija, ovisno o tome što smo izabrali ili na računalu ili na simulatoru, kao što je prikazano na slici 8.3

Osnovni dio aplikacije znači dohvat vremenske prognoze u trenutnoj lokaciji radi, naravno ako nam je lokacija omogućena na računalu. Zanimljivo je što makar je uključena dobit ćemo prvo upit o davanju dozvole aplikaciji kao na slici 8.4 Ono što ne radi su dijalozi. Ispada da IE ne podržava iskočne poruke JavaScripta pa ćemo ipak morati dodati dialog plugin. S obzirom da taj plugin ne radi za Android, samo ćemo posebno napisati kod za ostale platforme a za Android ostaviti isti. Treba prvo promijeniti kod za unošenje poštanskog broja. Ovo funkcionira drukčije od standardnog Prompt boxa. Da bismo dohvatili unešenu vrijednost mora se koristiti neka callback funkcija, ima polje više (naslov), te se ručno unose imena gumba. Kod izgleda ovako:

```
function onPrompt(results) {
    var zip = results.input1;
    if (results.buttonIndex == 2) {
        $('#zip-code-input').val(zip);
        if (window.localStorage.getItem("togglebutton")
        == "false") {
            cancel();
            setalarm();
        }
    }
}
```

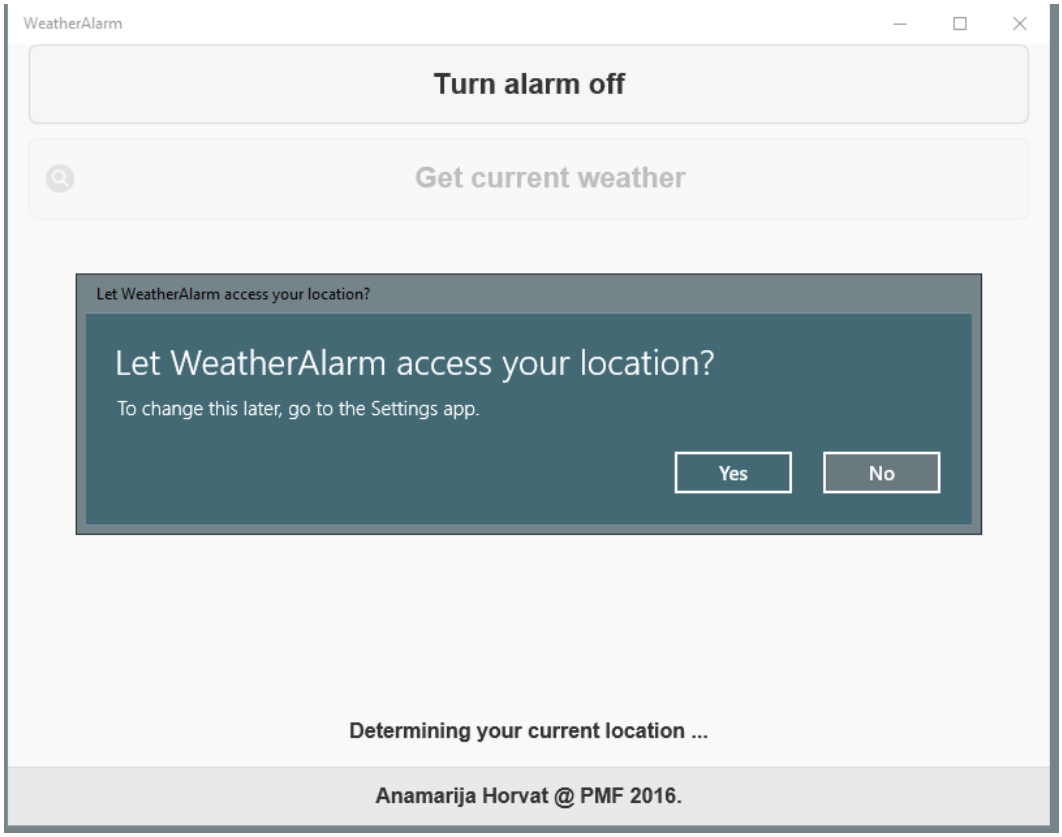


Slika 8.3: Windows aplikacije

```
function popup() {
    $('#error-msg').hide();
    if (device.platform != "Android") {
    navigator.notification.prompt(
    'Enter your zip code',  // poruka
    onPrompt,                  // callback
    'Please fill',             // naslov
    ['Cancel', 'OK']              // nazivi gumba
    );
    }
    else { //ovdje ostaje kod koji smo napisali za Android
```

Također u funkciji za postavljanje alarma moramo promijeniti Alert box na sljedeći način:

```
if (device.platform != "Android") {
        navigator.notification.alert(
        'Alarm set for'+time,  // poruka
        alertDismissed,           // callback
```



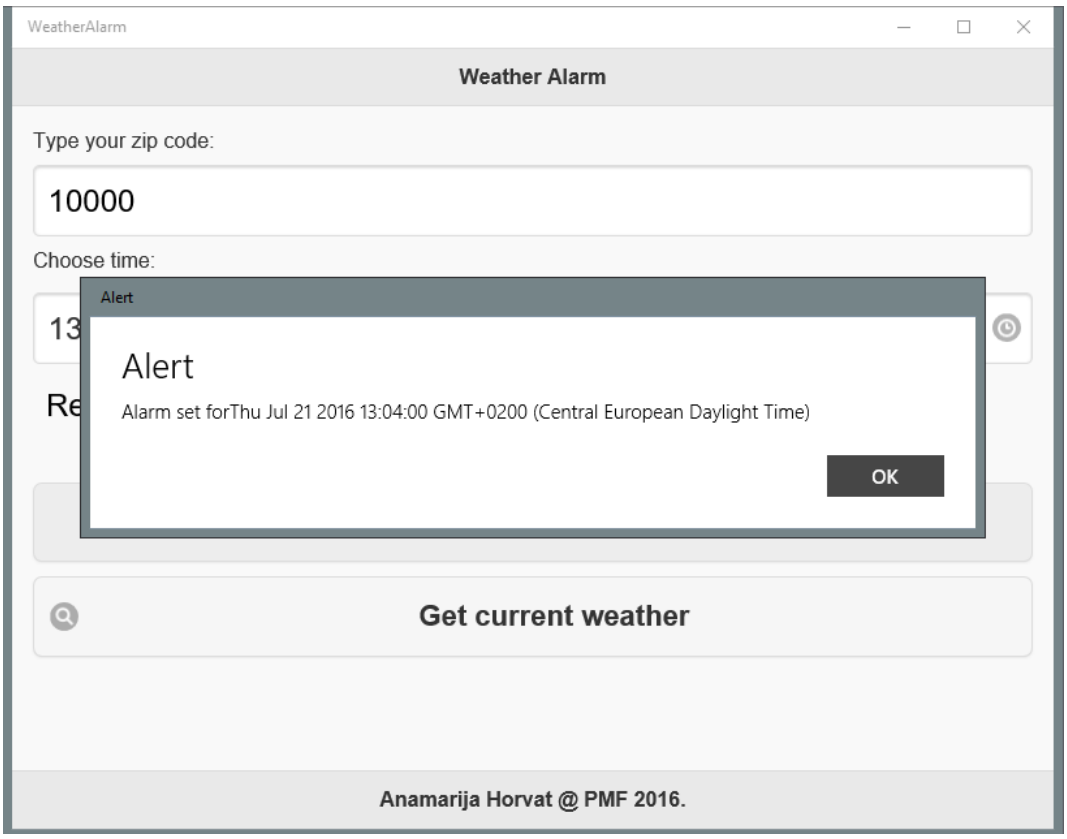
Slika 8.4: Privatnost Windows aplikacije

```
        'Alert',                    // naslov
        'OK'                          // naziv gumba
    );
        }
        else
            alert("Alarm set for " + time);

    }
```

Ovdje se isto mora koristiti callback, to polje se ne može ostaviti prazno. No s obzirom da se ništa ne treba desiti to će biti prazna funkcija.

```
function alertDismissed() {
        // ne radi nista
    }
```

Poruka o alarmu će izgledati kao na slici 8.5. Rubna boja je boja koja je postavljena na računalu, u ovom slučaju sivo-plava. Na nekom drugom računalu rubna boja bi bila možda drugačije boje.Neke pluginove naravno ne možemo testirati na Windows aplikaciji, jer Windowsi nemaju npr. SIM karticu. No osnovni dio aplikacije radi. To pokazuje da s Apache Cordovom također možemo razvijati Windows aplikacije.



Slika 8.5: Dijalozi na Windowsu

Windows Phone

Ako smo već testirali aplikaciju na Windows-u kao što je objašnjeno u prethodnom poglavlju i instalirali sve potrebno iz poglavlja 4.1, sve što trebamo napraviti za testiranje na emulatoru je pod **Solution Platorms** izabrati **Windows Phone 8** ili **Windows Phone (Universal)** te pod ”run” neki emulator na primjer **Mobile Emulator 10.x.xxxxx.x WVGA 512MB** i kliknuti zelenu strelicu. Nakon toga aplikacija će se prikazati na emulatoru. Plu-

gin za dohvat broja zemlje ne radi, emulatori nemaju SIM kartice stoga bi to trebalo testirati na uređaju. Ja nemam mogućnosti za taj način testiranja no reći ću nešto o njemu.

Za pokretanje na uređaju treba napraviti još neke stvari, razlikuju se ovisno o tome je li ciljna platforma Windows Phone 10 ili Windows Phone 8. Reći ćemo nešto o oba postupka.

Za Windows Phone 8 prvo trebamo nabaviti licencu koju smo već dobili prilikom instalacije Visual Studija no nju treba produljivati svakih 30 dana. Da bismo to učinili, u gornjem meniju kliknemo **Project** zatim **Store** pa **Acquire Developer License**. Sada trebamo registrirati mobilni uređaj na kojem ćemo testirati aplikaciju. Za to nam je potreban Microsoft račun ili Microsoft developer račun. Developerski račun je potreban ako želimo aplikaciju staviti u Windows store. Ovaj račun se plaća, a cijena ovisi o zemlji u kojoj se otvara račun. Trenutačna cijena za Hrvatsku iznosi 107 kuna, a račun se može otvoriti na adresi https://developer.microsoft.com/en-us/windows/programs/join. Tek nakon što smo otvorili račun možemo registrirati uređaj na kojem želimo testirati aplikaciju. Račun dozvoljava registriranje 3 uređaja no ako želimo testirati aplikaciju na više uređaja možemo jednostavno odregistrirati jedan i registrirati neki drugi. Kako se registrira uređaj može se pročitati na adresi https://msdn.microsoft.com/en-us/library/windows/apps/dn614128. S time smo omogućili testiranje aplikacije na Windows Phone 8.1. mobilnom uređaju. Sada ćemo reći kako isto omogućiti na Windows Phone 10.

Postupak za Windows Phone 10 je mnogo jednostavniji, potrebno je samo promijeniti neke postavke u mobilnom uređaju. Kako se to radi može se pročiti na adresi https://msdn.microsoft.com/en-us/windows/uwp/get-started/enable-your-device-for-development.

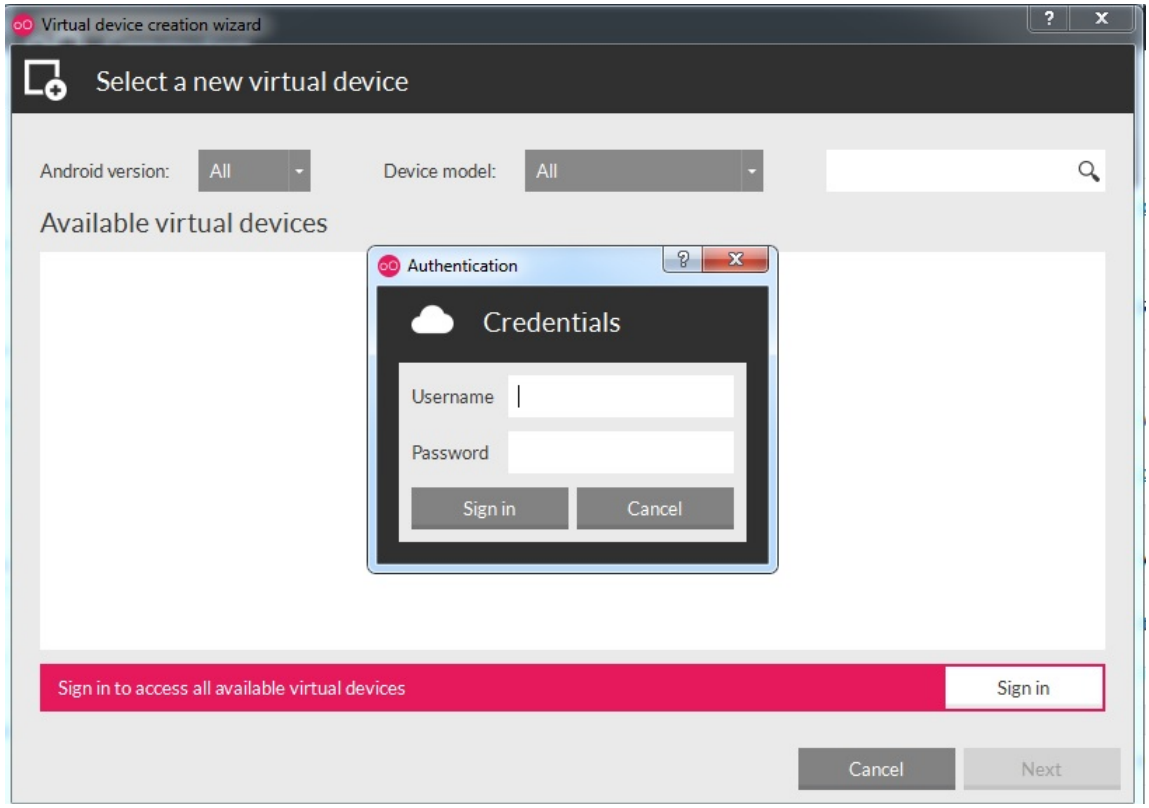
8.3 Testiranje aplikacije na Androidu

Testiranje android aplikacija puno je lakši postupak nego za iOS te postoji više mogućnosti, Apache Ripple Simulator, Visual Studio Emulator, Google Android Emulator, Genymotion emulator te na uređajima. Apache Ripple smo već vidjeli, Visual Studio emulator se pokreće na isti način samo umjesto **Ripple-Nexus(Galaxy)** odaberemo na primjer **VS Emulator 5" KitKat (4.4) XHDPI Phone** ili neki drugi koji počinje sa "VS". Sada ćemo objasniti ostala tri načina.

Genymotion emulator

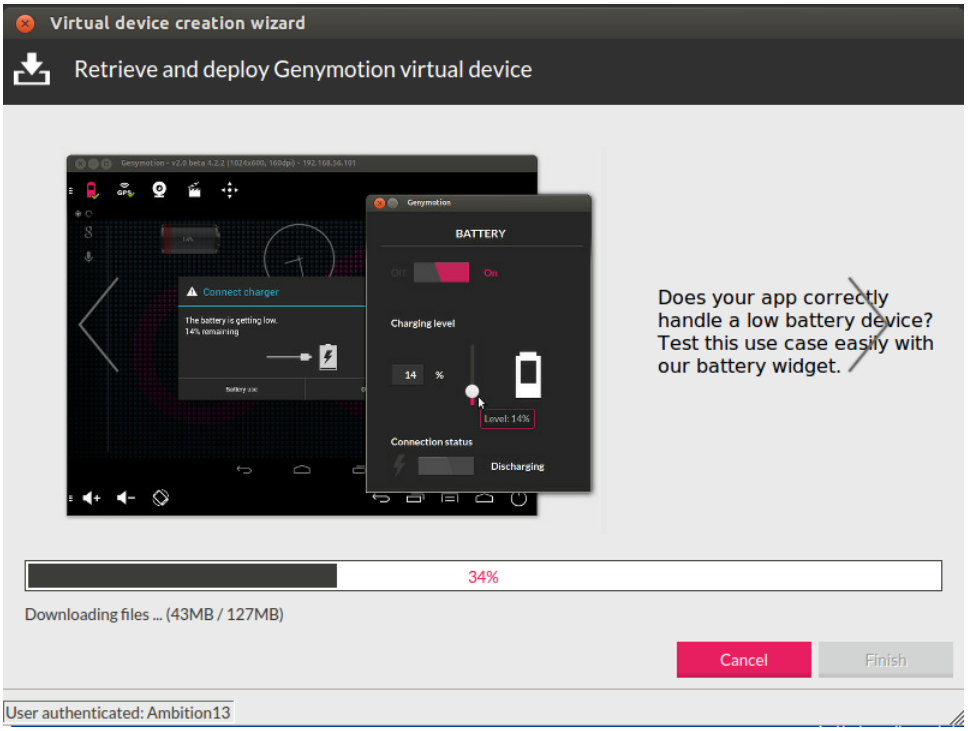
On je popularna alternativa ostalim Android emulatorima jer ima bolje performanse. Trebamo prvo kreirati račun na adresi https://www.genymotion.com/account/create/ te se onda besplatno može nabaviti na https://www.genymotion.com/pricing-and-licensing/. Nakon što smo instalirali Genymotion, pokrenemo ga, pojavit će se poruka da nemamo još nijedan virtualni uređaj i dijalog će nas pitati želimo li kreirati novi, kliknemo **Yes**. Pojavit će se prozor **Select a new virtual device** sa zacrvenjenom porukom na dnu i gumbom **Sign**

in, kliknemo ga i izaći će prozor gdje popunimo podatke o računu, kao na slici 8.6.

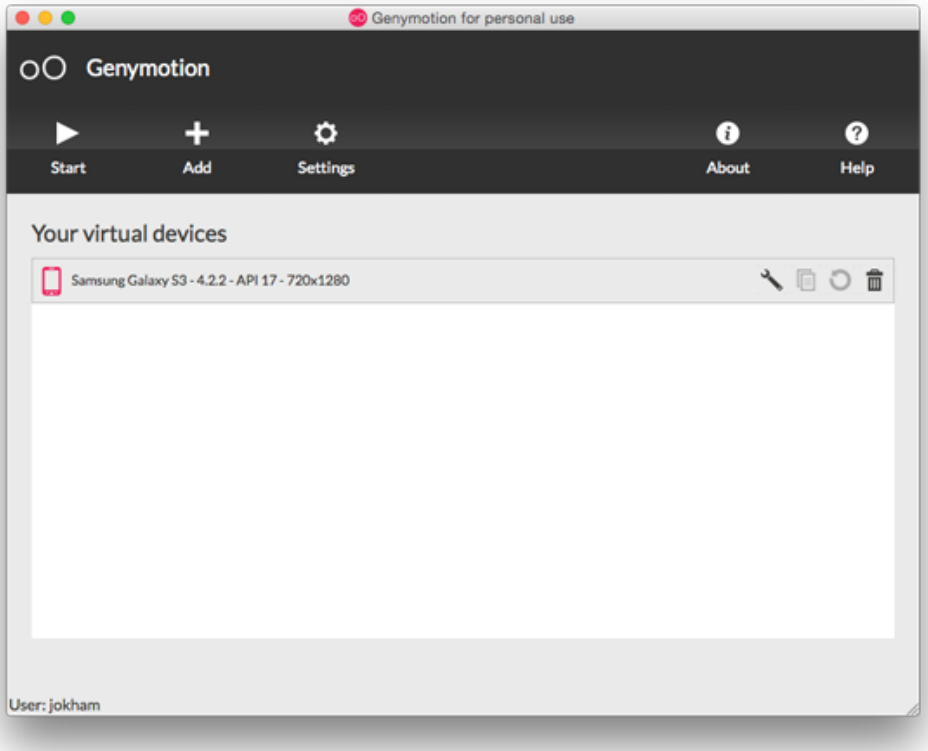


Slika 8.6: Prijava u Genymotion

Nakon što smo unijeli korisničko ime i lozinku te kliknuli **Sign in** pojavit će se lista virtualnih uređaja ispod **Available virtual devices**, selektiramo neki uređaj na kojem želimo testirati i kliknemo **Next**. Prikazat će se prozor **Create a new virtual device** koji će imati prazno polje ispod **Virtual device name**, u njega upišemo ime koje želimo za taj virtualni uređaj i kliknemo **Next**. Nakon toga otvorit će se prozor sa progress barom o preuzimanju virtualnog uređaja, kao na slici 8.7. To može malo potrajati, nakon što progress bar dođe do 100% zacrvenjet će se gumb **Finish**, kliknemo ga. Zatvorit će se taj prozor i otvoriti početni prozor Genymotiona samo što će sada imati uređaj koji smo upravo kreirali na popisu kao na slici 8.8. Selektiramo ga i kliknemo gornju lijevu strelicu za pokretanje. Uskoro bi se trebao pojaviti emulator na računalu.



Slika 8.7: Preuzimanje virtualnog uređaja u Genymotion-u



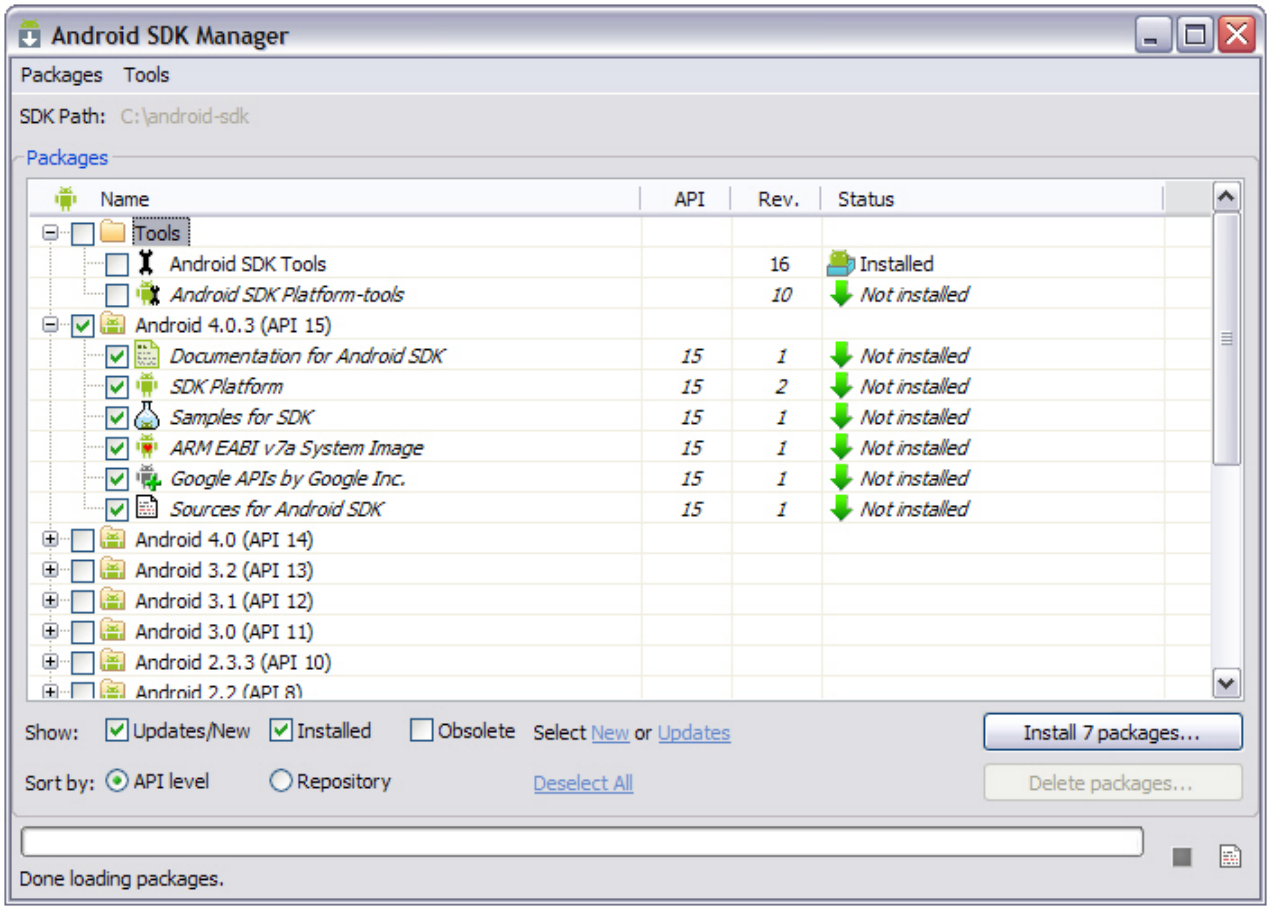
Slika 8.8: Pokretanje Genymotion emulatora

Sada u Visual Studiju s otvorenim projektom gdje je naša aplikacija, na mjestu gdje smo prije birali vrstu emulatora izaberemo **Device** i kliknemo zelenu strelicu za pokretanje. Aplikacija će se uskoro instalirati i prikazati na Genymotion emulatoru.

Google Android Emulator

Ovaj emulator je dio Android SDK pa ne zahtijeva dodatnu instalaciju jer se instalira zajedno sa Visual Studijom, no trebamo instalirati ”system images”.

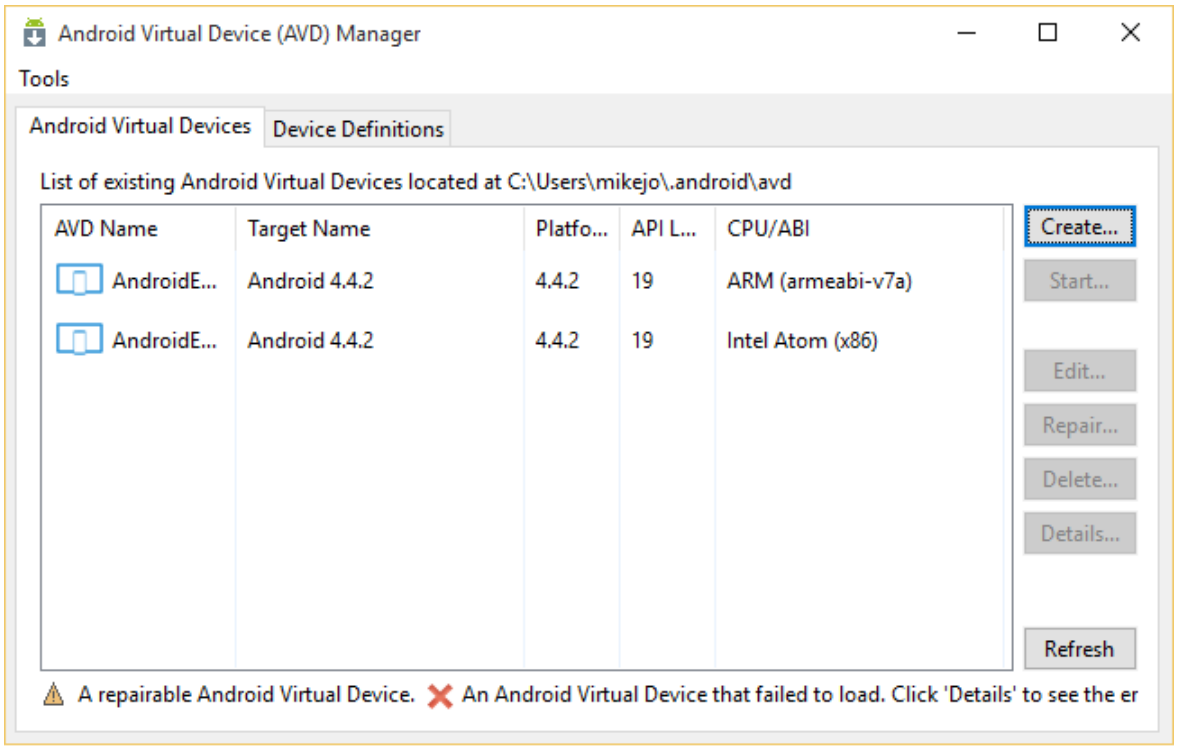
To učinimo tako da kliknemo **SDK Manager.exe** koji se nalazi u `C:\ProgramFiles(x86)\Android\android-sdk`. Otvoriti će se prozor kao na slici 8.9.



Slika 8.9: Android SDK Manager

U desnom donjem kutu pojavit će se gumb **Install x packages...** gdje je x neki broj, no mi želimo selektirati u gornjem popisu sve dokumente čije ime završava sa ”System Image”. Nakon što smo to učinili kliknemo gumb za instaliranje. Ovo može poduže potrajati no nakon što je dovršeno imamo mogućnost testiranja aplikacije na velikom broju uređaja i različitim ciljanim API-jima.

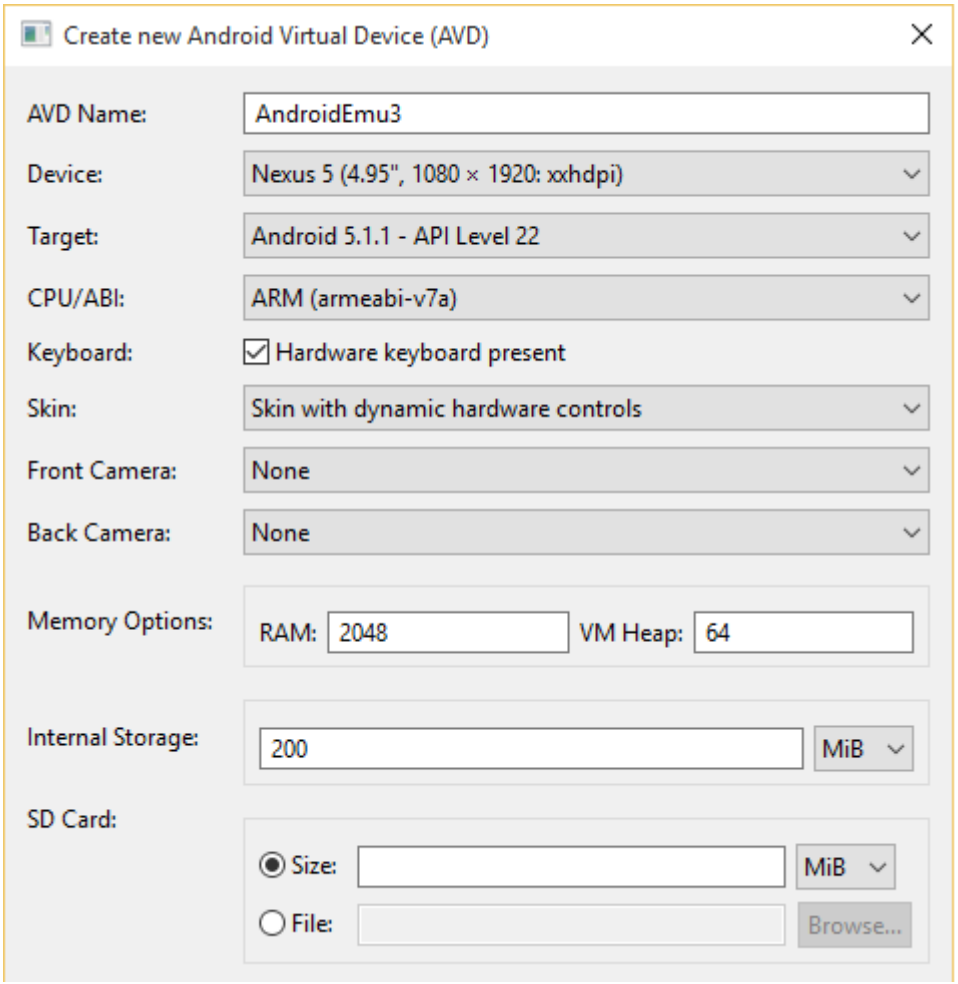
Da bismo pokrenuli emulator, u istoj datoteci gdje se nalazi SDK Manager, kliknemo **AVD Manager.exe**. Prikazat će se prozor kao na slici 8.10 ,kliknemo **Create** za kreiranje novog virtualnog uređaja.



Slika 8.10: Android AVD Manager

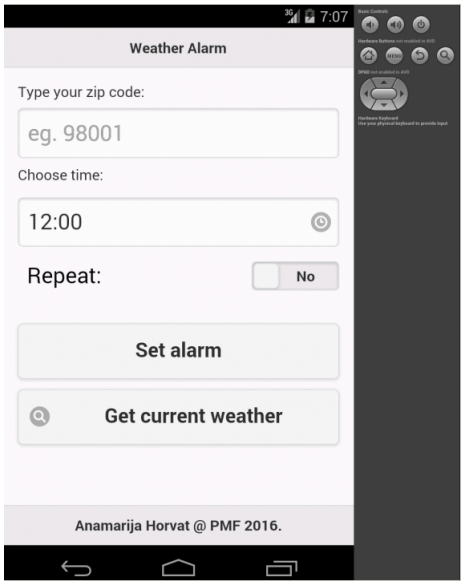
Otvorit će se prozor gdje trebamo ispuniti svojstva uređaja, pod **AVD Name** stavimo bilo ime koje želimo za taj uređaj. Pod **Device** izaberemo uređaj na kojem želimo testirati, a ispod toga pod **Target** ciljani API, neki API-ji neće biti dostupni na nekim uređajima pa ako nema API-ja kojeg želimo odaberemo drugi uređaj. Zatim pod **CPU/ABI** izaberemo procesor, no Intelovi procesori neće raditi na računalu koje nema Intelov procesor, stoga moramo obratiti pažnju na taj dio. Ostale podatke ispunimo kako želimo i trebamo, no pod

RAM se ne preporuča staviti veliki broj jer se emulator možda neće pokrenuti ili će jako sporo raditi. Nakon što smo sve ispunili to će otprilike izgledati kao na slici 8.11



Slika 8.11: Kreiranje novog virtualnog uređaja

Kliknemo **OK** na kraju tog prozora, i na sljedećem koji se otvori. Sada će se pojaviti upravo kreirani uređaj pod **Android Virtual Devices** kao na slici 8.10. Selektiramo ga i kliknemo desno **Start**. Uskoro bi se trebao prikazati emulator na računalu. Sada u Visul Studiju gdje smo pokretali ostale emulatore za testiranje aplikacije izaberemo **Google Android Emulator** i aplikacija će se instalirati i prikazati na emulatoru kao na slici 8.12



Slika 8.12: Android emulator

Uspješno sam testirala sve plug-inove osim dohvata broja zemlje i detekcije platforme. Te dvije stvari nisu radile na emulatoru i zato je potrebno testirati i na uređaju.

Android uređaji

Prvo je potrebno instalirati driver za uređaj na kojem želimo testirati aplikaciju. Ako želimo testirati na Google Nexus uređaju onda trebamo instalirati Goolge USB driver, može se naći na https://developer.android.com/studio/run/win-usb.htmltop, a za ostale uređaje trebamo instalirati odgovarajući OEM driver, OEM driveri se mogu naći na https://developer.android.com/studio/run/oem-usb.html.

Nakon što smo instalirali driver moramo na uređaju omogućiti developer mode. Na Android-u 4.2 ili višem on je defaultno skriven, da bismo ga prikazali otvorimo **Settings->About Phone** i tapnemo 7 puta po **Build Number**. Sada kada odemo nazad na **Settings** pojavit će se **Developer options**, uključimo ih. Na starijim Android-ima samo odemo na **Settings->Developer options** i uključimo ih. U Visual Studiju gdje smo do sada birali emulatore izaberemo **Device** i pokrenemo aplikaciju. Na mobilnom uređaju pojavit će se poruka **Allow USB Debugging**, kliknemo **Yes** i aplikacija će se instalirati i prikazati na uređaju. Aplikaciju sam testirala na uređaju Sony Xperia Z1 Compact, Android verzije 5.1.1. Aplikacije izgleda kao što je prikazano na slikama iz prethodnog poglavlja i rade sve funkcionalnosti.

Poglavlje 9

Pakiranje aplikacije za objavu

Pakiranje iOS i Windows aplikacije

Objavljivanje ovih aplikacija se plaća. Za iOS nisam u mogućnosti ni izgraditi aplikaciju jer nemam Macintosh računalo pa neću pisati o pakiranju aplikacije. O tome možete pročitati na adresi
http://taco.visualstudio.com/en-us/docs/tutorial-package-publish-readme/package-the-ios-version-of-your-app.

Za Windows aplikacije je slična stvar. Možemo ju izgraditi i testirati na simulatorima, no ne možemo ju testirati na uređajima. To znači da nismo testirali aplikaciju kako treba pa također nećemo pisati kako se pakira. To se može pročitati na adresi
http://taco.visualstudio.com/en-us/docs/tutorial-package-publish-readme/package-the-windows-version-of-your-app.

Pakiranje Android aplikacije

Trebamo u **config.xml** pod **Common** ispuniti stvari koje smo prije ostavili praznima. Bitno je da pod **Domain Access** dodamo **https://query.yahooapis.com** jer dohvaćanje vremena pristupa toj adresi.

Zatim treba pod **Android** tabom ispuniti polja koje smatramo potrebnima. Na primjer ako aplikacija ne radi na nekim verzijama Androida možemo ih maknuti sa liste tako da definiramo minimalnu i maksimalnu verziju Android SDK.

Sada trebamo generirati ”keystore”. To je dokument koji sadrži neke osnovne podatke o nama i set ključeva. Potreban je za stavljanje aplikacije u Google Store jer veže aplikaciju za jedinstveni ključ. Ako ćemo htjeti stavljati druge verzije te aplikacije u Google Store morat ćemo koristiti taj isti ključ. To sprječava da netko drugi ima mogućnost izdati neku lažnu aplikaciju pod našom, naime bez ključa je to nemoguće napraviti. Da bismo ga kreirali trebamo otvoriti **Command Prompt** kao administrator te navigirati do **bin** mape

određene verzije Jave koju imamo instaliranu na računalu npr. `C:\ProgramFiles(x86)\Java\jdk1.7.0_55\bin`. Zatim trebamo upisati sljedeću naredbu:

```
keytool -genkey -v -keystore c:\my-release-key.keystore -alias johnS
-keyalg RSA -keysize 2048 -validity 10000
```

s time da zamijenimo **my-release-key** s nazivom koje bi htjeli da ima datoteka gdje će se nalaziti ključ, a **johnS** s imenom koje je povezano s nama odnosno autorom. Nakon toga dobit ćemo upite da unesemo lozinku i seriju pitanja o nama, tj sljedeće:

```
Enter keystore password: pwd123
Re-enter new password: pwd123
What is your first and last name?
[Unknown]= John Smith
What is the name of your organizational unit?
[Unknown]= ABC
What is the name of your organization?
[Unknown]= XYZ
What is the name of your of your City or Locality?
[Unknown]= Redmond
What is the name of your State or Province?
[Unknown]= WA
What is the two-letter country code for this unit?
[Unknown]= US
Is CN=John Smith, OU=ABC, O=XYZ, L=Redmond, ST=WA, C=US correct??
[no]=  y
```

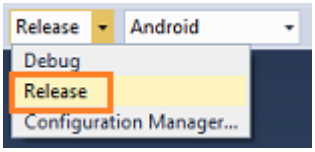
Na ta pitanja treba odgovoriti točno i iskreno jer te podatke za taj ključ više nećemo moći mijenjati. Datoteka **my-release-key.keystore** će se kreirati na C disku i u njoj će se nalaziti ključ koji vrijedi 10000 dana.

Sada trebamo povezati taj ključ s našom aplikacijom. Da bismo to učinili otvorimo **build.json** datoteku našeg projekta i popunimo s podacima o ključu. Za ovaj primjer to je sljedeće:

```
{
 "android": {
     "release": {
         "keystore":"c:\\my-release-key.keystore",
         "storePassword":"pwd123",
         "alias":"johnS",
         "password":"pwd123",
         "keystoreType":""
       }
```

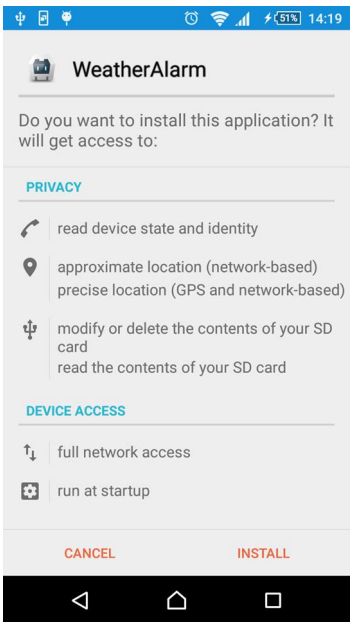
```
    }
}
```

Na kraju još moramo izgraditi aplikaciju. U gornjoj alatnoj traci izaberemo **Android** i **Release** kao na slici 9.1, izaberemo neki emulator ili uređaj, no ne Appache Ripple emulator, i kliknemo zelenu strelicu. To je kreiralo datoteku **android-release.apk** koja se nalazi u



Slika 9.1: Generiranje release verzije aplikacije

`bin/Android/Release/` mapi projekta. To je datoteka koju treba staviti u Google Store. Mi je nećemo stavljati jer se plaća 25 dolara, no kako se to radi može se pročitati na adresi https://support.google.com/googleplay/android-developer/answer/113469?hl=en. Makar ova aplikacije nije dostupna putem dućana možemo je ručno instalirati na mobitel klikom na tu datoteku. Prilikom instalacije dobit ćemo upit kao na slici 9.2.



Slika 9.2: Dozvole aplikacije

To su dozvole koje aplikacija treba za rad, naime plug-inovi koriste nativne dijelove uređaja, a za njihovo korištenje potrebna je dozvola. Prva dozvola potrebna je za čitanje SIM kartice pomoću koje dohvaćamo kod zemlje. Druga je potrebna za dohvat vremena. Treća za pohranjivanje podataka kojima smo spremili promjene unešene od korisnika. Četvrta i peta dozvola je potrebna zbog alarma. Klikom na **install** instalirali smo finalnu verziju na uređaj s čime je dovršen ovaj diplomski rad.

Bibliografija

[1] *Apache Cordova documentation*, 2016, https://cordova.apache.org/docs/en/latest/, posjećena 2016-07-19.

[2] *cordova-plugin-dialogs*, 2016, https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-dialogs/index.html, posjećena 2016-07-19.

[3] *Genymotion User Guide*, 2016, https://docs.genymotion.com/pdf/PDF_User_Guide/Genymotion-2.6.0-User-Guide.pdf, posjećena 2016-05-12.

[4] *HTML 5 in mobile devices*, 2016, https://en.wikipedia.org/wiki/HTML5_in_mobile_devices, posjećena 2016-03-29.

[5] *OpenWeatherMap documentation*, 2016, http://openweathermap.org/current, posjećena 2016-07-19.

[6] *Personal digital assistant*, 2016, https://en.wikipedia.org/wiki/Personal_digital_assistant, posjećena 2016-03-01.

[7] *Source-to-source compiler*, 2016, https://en.wikipedia.org/wiki/Source-to-source_compiler, posjećena 2016-03-29.

[8] *Visual Studio Tools for Apache Cordova*, 2016, http://taco.visualstudio.com/en-us/, posjećena 2016-07-19.

[9] Dave Alden, *Cordova diagnostic plugin*, 2016, https://www.npmjs.com/package/cordova.plugins.diagnostic#cordova-diagnostic-plugin, posjećena 2016-07-19.

[10] Peter Bakondy, *cordova-plugin-sim*, 2016, https://github.com/pbakondy/cordova-plugin-sim, posjećena 2016-07-19.

[11] Robert Chipperfield, *An introduction to cross-platform mobile development technologies*, 2012, http://www.codeproject.com/Articles/388811/An-introduction-to-cross-platform-mobile-developme, posjećena 2016-03-29.

[12] Sebastián Katzer, *cordova-plugin-background-mode*, 2016, https://github.com/katzer/cordova-plugin-background-mode, posjećena 2016-07-19.

[13] Taylor Martin, *The evolution of the smarthphone*, 2014, http://pocketnow.com/2014/07/28/the-evolution-of-the-smartphone, posjećena 2016-03-24.

[14] J.T. Sage, *JTSage Datebox*, 2016, http://dev.jtsage.com/DateBox/, posjećena 2016-07-19.

[15] John M. Wargo, *Apache Cordova 3 Programming*, Addison-Wesley Professional, 2013.

Sažetak

U današnjem svijetu se za obavljanje različitih poslova sve češće koriste mobiteli. Pri tome se koriste mobilne verzije web stranica ili mobilne aplikacije. Kako bi aplikacije bile dostupne što većem broju korisnika one trebaju podržavati više platformi poput iOS-a, Androida i Windows Phone-a što zahtijeva dodatne materijalne i ljudske resurse te je dovelo do potrebe za softverom koji omogućuje razvoj višeplatformskih aplikacija. Jedan takav softver, Apache Cordova obradit ćemo u ovom diplomskom radu.

Summary

Today most people use cellphones on daily basis for doing various tasks. Mobile web pages and applications are among the most used. How we could make applications available to a larger group of people they need to support various platforms like iOS, Android and Windows Phone. Making such applications requires an increase in human and financial resources, which created the need for the invention of software that enables developement of cross-platform aplications. One such software is Apache Cordova and we will tell more about it in this thesis.

Životopis

Rođena sam 24. srpnja 1988. godine u Zagrebu gdje sam nakon završetka Osnovne škole Izidora Kršnjavog 2003. godine upisala X. gimnaziju "Ivan Supek", prirodoslovno-matematički smjer. Po završetku gimnazije 2007. godine upisujem Prediplomski sveučilišni studij matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Nakon preddiplomskog, 2013. na istom fakultetu upisujem Diplomski sveučilišni studij Računarstvo i matematika.