Upravljanje lokacijom u višeplatformskim mobilnim aplikacijama

Pavić, Kristian

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

UPRAVLJANJE LOKACIJOM U VIŠEPLATFORMSKIM MOBILNIM APLIKACIJAMA

Diplomski rad

Kristian Pavić

Osijek, 2016.

SADRŽAJ

1. UVOD

U današnjem svijetu informatizacije i digitalizacije društva postalo je sasvim uobičajeno posjedovati pametni mobilni uređaj (engl. *smartphone*). Ono što je prije nekoliko desetljeća bilo nezamislivo, danas je dostupno skoro svima. Prva računala 50-ih godina prošlog stoljeća ispunjavala su čitave prostorije, a danas imamo uređaje koje bezbrižno nosimo u džepu te uz to imaju do nekoliko puta veću procesorsku moć obrade podataka. Prema [1], samo u 2015. godini prodano je više od 1,4 milijarde pametnih mobilnih uređaja.

Razvojem industrije pametnih mobilnih uređaja, kao i u svakoj industriji u razvoju, došlo je do pojave različitih proizvođača na tržištu. Svaki od njih ima svoje posebnosti i prednosti, s ciljem privlačenja što većeg broja kupaca. Uz sam hardver, razvile su se i različite softverske platforme. Svaka platforma temelji se na drugačijem operacijskom sustavu, što vodi do toga da je za razvoj svake aplikacije za svaku od mobilnih platformi potrebno znanje drugačijeg programskog jezika. Stoga, ako je potrebno razviti jednu mobilnu aplikaciju za najbitnije mobilne platforme, to jest one platforme s najviše korisnika, potrebno je poznavanje najmanje tri programska jezika. U pravilu to znači da je za svaku platformu potrebna jedna osoba koja dobro poznaje potreban programski jezik za razvoj mobilne aplikacije te na kraju, svaka osoba obavlja isti posao samo s određenim prilagodbama koje ovise o programskom jeziku i platformi za koju razvija aplikaciju. Pomoću hibridnih mobilnih aplikacija moguće je razviti sve tri aplikacije odjednom, od strane jedne osobe i to sa znanjem potrebnih web tehnologija.

1.1. Zadatak diplomskog rada

U teorijskom dijelu potrebno je ukratko opisati tehnologije za razvoj višeplatformskih mobilnih aplikacija. Detaljno opisati Apache Cordova tehnologiju. Opisati načine pristupa lokaciji korisnika na različitim platformama korištenjem Apache Cordova tehnologije.

Za praktični dio rada razviti višeplatformsku mobilnu aplikaciju za upravljanje sastancima koja će sadržavati funkcionalnost upravljanja lokacijom mobilnog uređaja na barem dvije platforme.

1.2. Organizacija rada

Kroz ovaj rad bit će objašnjene prednosti i nedostaci razvoja mobilnih hibridnih aplikacija. Osim teorijskih podloga potrebnih za razvoj takvih aplikacija, bit će pokazani i primjeri proizašli iz praktičnog dijela rada.

Nastavak rada organiziran je kroz sljedeća poglavlja:

- drugo poglavlje sadrži potrebne teorijske podloge za razvoj mobilne hibridne aplikacije. Sve tehnologije i programski jezici korišteni u praktičnom dijelu rada bit će objašnjeni u ovom poglavlju,
- u trećem poglavlju prikazan je razvoj hibridne mobilne aplikacije, čija je funkcija upravljanje lokacijom korisnika aplikacije,
- u četvrtom poglavlju iznesen je zaključak, koji je proizašao iz ovog rada.

Nakon zaključka dan je pregled literature primijenjene za izradu ovog rada. Također su dani kratki sažeci rada na hrvatskom i engleskom jeziku, kratak životopis autora te potrebni prilozi za bolje razumijevanja rada.

2. TEORIJSKE PODLOGE

2.1.HTML

HTML je kratica za „*Hyper Text Markup Language*“, što znači prezentacijski ili opisni jezik, HTML stoga nije klasični programski jezik kao što su to na primjer C ili JavaScript. HTML se koristi za izradu web aplikacije i to na način da se oznakama (engl. *tags*) opisuje koji elementi će se nalaziti na web stranici te što će sadržavati, kao na primjer tekst, slike i slično. Prezentacija web aplikacije, realizirana HTML oznakama, predaje se web pregledniku, koji uz pomoć danih uputa učitava strukturu web stranice u memoriju te prikazuje sadržaj web stranice korisniku na zaslonu. HTML standard je tako osmišljen da se dani opisi u svakom pregledniku prikazuju na jednak način. Kako je HTML samo opisni programski jezik, ne podržava aritmetičke niti bilo kakve druge matematičke ili logičke operacije.

Sl. 2.1. *Prikaz osnovne strukture HTML dokumenta*

Osnovna struktura HTML dokumenta prikazana je na slici 2.1.. Svaki element u HTMLu počinje početnom oznakom, koja sadrži ime oznake te završava završnom oznakom. Završna oznaka, osim imena oznake sadrži i znak „/“ (engl. *slash*), koji jasno upućuje na to da se radi o završnoj oznaci elementa, dok se između oznaka može nalaziti tekst koji želimo prikazati u pregledniku. Na slici 2.1. nalaze se sljedeći HTML elementi:

- *<html> </html>* – sve što se nalazi unutar ovih oznaka, to jest ovog HTML elementa je HTML kôd i svi ostali HTML elementi nalaze se u njemu
- *<head> </head>* – sadrži informacije o stranici koje se ne prikazuju u pregledniku

- *<body> </body>* – samo ono što se nalazi unutar *body* elementa prikazuje se u pregledniku
- *<h1> </h1>* – tekst unutar *h1* elementa u pregledniku prikazuje se kao glavni naslov. Ponašanje teksta unutar ove oznake definirana je HTML standardom i rezultira većom veličinom slova od ostalog teksta.

Osim HTML elemenata koji označavaju poseban dio u web stranici ili elemenata koji imaju posebnu ulogu, kao što na primjer element *h1* ima ulogu glavnog naslova, tako postoje i elementi kojima možemo dodijeliti posebne HTML atribute. Prema [2], atributi dodjeljuju dodatne informacije o sadržaju nekog elementa i sastoje se od imena i vrijednosti razdvojenih znakom jednakosti. Na slici 2.2. prikazan je primjer jednog HTML elementa s dodatnim atributima, koji u ovom slučaju elementu dodjeljuju jedinstvenu identifikaciju među ostalima elementima te određuju širinu elementa na 200 točkica (engl. *pixels*).

```
<p id="odlomak" width="200px">Neki tekst</p>
```

Sl. 2.2. *Primjer korištenja HTML atribut*

2.2.CSS

CSS stoji za *„Cascading Style Sheets“*. CSS je stilski jezik uz pomoć kojega opisujemo izgled HTML dokumenta i njegovih elemenata. Ključno za razumijevanje CSSa jest da CSS svaki element HTMLa „gleda“ kao posebnu jedinicu okruženu nevidljivim kvadratom. CSS omogućuje određivanje pravila koja određuju način na koji će se svaki kvadrat i sav sadržaj unutar njega prikazati u pregledniku. Na slici 2.3. prikazan je primjer sintakse CSS pravila, koje se sastoji od selektora (engl. *selector*) i deklaracije (engl. *declaration*).

Sl. 2.3. *Primjer CSS pravila*

Selektori pokazuju na koji element se pravilo odnosi. Isto pravilo može se odnositi na više elemenata ako njihova imena podijelimo zarezima. Deklaracije govore kako će se element označen selektorom stilizirati. Deklaracije se sastoje od svojstva (engl. *property*) i vrijednosti (engl. *value*), koje su odvojene dvotočkom, dok su deklaracije međusobno odvojene točkom

zarez. Na slici 2.4. pokazan je primjer primjene CSSa nad jednim elementom s više deklaracija. Svojstva deklaracije određuju što se želi promijeniti, kao na primjer font, boja ili širina elementa, dok vrijednosti određuju kako se pojedino svojstvo želi promijeniti, na primjer boju (svojstvo) sadržaja elementa promijeniti u žutu boju (vrijednost).

Sl. 2.4. *Primjer CSS deklaracije*

Prva riječ skraćenice CSS, *Cascading* (hrv. kaskadni), obilježava da je CSS kaskadni stilski jezik. U praksi to znači, ako postoji više pravila koja utječu na isti element, potrebno je znati koje će od tih pravila imati prednost nad drugima. Prema [2], sljedeća su bitna obilježja kaskade u CSS stilskom jeziku:

- pravilo zadnjeg – ako su dva ili više selektora jednaka, primijeniti će se pravila definirana zadnjim selektorom
- pravilo specifičnosti – ako postoje dva ili više pravila za isti element, primjenjuje se ono pravilo koje je specifičnije za pojedini element
- pravilo nasljeđivanja – neka svojstva elementi mogu naslijediti od drugih elemenata, iako nisu specifično njima dodijeljena. Tako na primjer, ako *font-family* svojstvo dodijelimo *body* elementu, svi elementi unutar *body* elementa naslijediti će to svojstvo (u slučaju da nemaju specifično pravilo dodijeljeno baš njima).
- iznimka – nakon svake vrijednosti deklaracije moguće je dodati ključnu riječ *!important*, koja pregledniku ukazuje na to da je neovisno o svim ostalim pravilima za dani element potrebno primijeniti pravilo s ključnom riječju *!important*.

2.3. JavaScript

JavaScript je prema [3], dinamičan, netipiziran programski jezik visoke razine. Standardiziran je sa strane ECMAScript standarda. Uz HTML i CSS, JavaScript je jedna od temeljnih tehnologija za izradu web aplikacija te je podržan sa strane svih bitnijih web preglednika. Osim tehnologije, koja je bitan dio za razvoj današnjih web aplikacija, JavaScript se zbog svoje brzine izvođenja, lake implementacije i drugih prednosti koristi i u druge svrhe. JavaScript se tako na primjer može koristiti kao jezik kojim se može pristupati i upravljati nekim od baza podataka (MongoDB) ili kao platforma za razvoj aplikacija (Adobe Integrated Runtime).

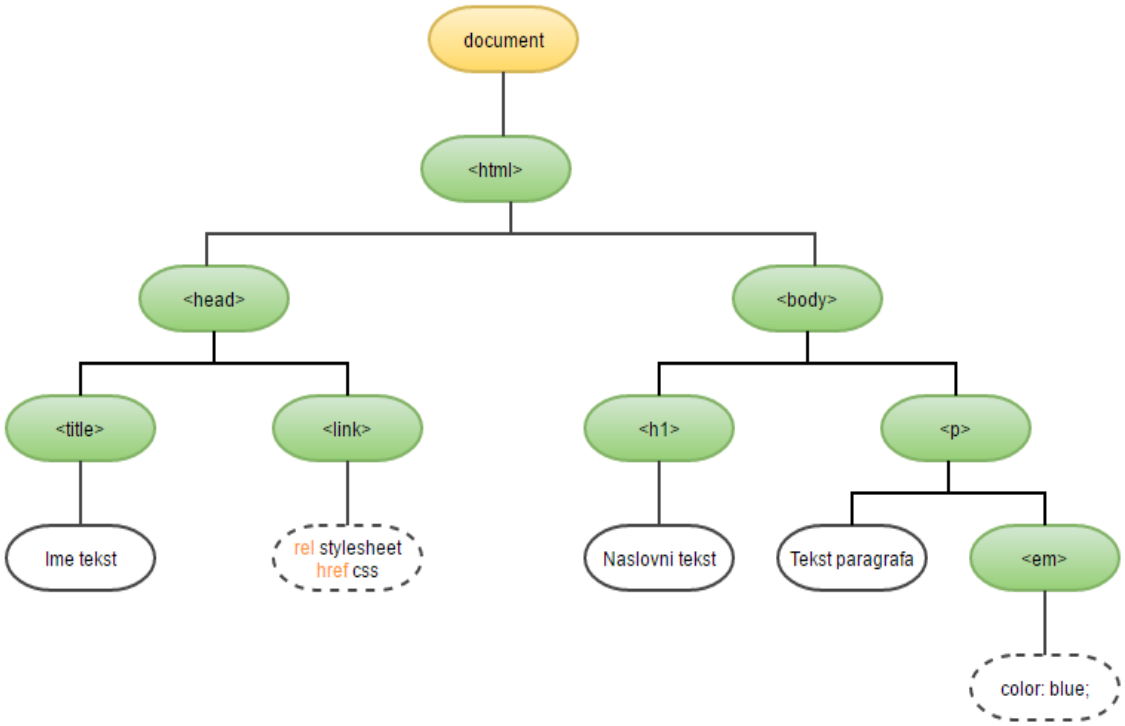
U ovom radu, JavaScript koristit će se kao programski jezik na klijentskoj strani hibridne mobilne aplikacije. To prema [4] znači da se aplikacija izvodi na osobnom računalu ili drugom uređaju korisnika (eng *client*), a ne na udaljenom poslužitelju (engl. *server*). Takav način razvoja aplikacije omogućuje jednostavnost pri izradi aplikacije, poboljšava brzinu izvršavanja (nije potrebno primati upute s udaljenog servera), olakšava testiranje aplikacije te najvažnije, omogućuje laku implementaciju na sve uređaje koji pristupaju navedenoj aplikaciji.

2.3.1. JavaScript u web programiranju

Kao što je već navedeno, JavaScript uz HTML i CSS čini najbitnije tehnologije za razvoj modernih web aplikacije. Svaka od navedenih tehnologija ima svoju svrhu, te se prema [5] može reći da je svaka tehnologija poseban sloj na svakoj web aplikaciji. Prema svrsi, slojeve web aplikacije dijelimo na:

- sadržajni sloj (engl. *content layer*) – za ovaj sloj odgovoran je HTML. U ovom sloju nalazi se sadržaj web aplikacije. Osim sadržaja, ovaj sloj web aplikaciji daje strukturu i semantička obilježja.
- prezentacijski sloj (engl. *presentation layer*) – za ovaj sloj odgovoran je CSS. Prezentacijski sloj nadograđuje sadržajni sloj pravilima koja određuju kako će se sadržaj u sadržajnom sloju prikazivati (npr. pozadine, rubovi, boje, dimenzije itd.).
- sloj ponašanja (engl. *behavior layer*) – za ovaj sloj odgovoran je JavaScript. U ovom sloju moguće je odrediti ponašanje web aplikacije dodavanjem interakcija, kao što su:
    - pristupanje, dodavanje ili brisanje bilo kojeg elementa, atributa ili teksta iz sadržajnog i prezentacijskog sloja
    - dodavanje programskih skripti koje mogu izmijeniti web aplikaciju dajući potrebne upute pregledniku
    - reagiranje na događaje na način da se određena skripta pokrene kada dođe do određenog događaja, kao na primjer pritisak tipke ili klik miša.

Kako bi se razumio način na koji JavaScript manipulira sadržajnim i prezentacijskim slojem web aplikacije, potrebno je shvatiti kako web preglednik prikazuje web aplikaciju korisniku. Preglednik tako prvo prima potrebne podatke s udaljenog poslužitelja te stvara model web aplikacije, takozvani DOM (engl. *Document Object Model*) u lokalnoj memoriji. Na vrhu modela web aplikacije nalazi se *document* objekt koji predstavlja jednu stranicu unutar cijele web aplikacije i taj objekt predstavlja cijeli dokument, to jest HTML strukturu ili sadržajni sloj pojedine stranice web aplikacije. Kao i svaki objekt, *document* objekt sadrži svojstva, metode i događaje. Na razini ispod *document* objekta nalaze se elementi koji se zovu čvorovi (engl. *nodes*), koji su također objekti. Primjer DOMa jedne jednostavne stranice unutar neke web aplikacije prikazan je na slici 2.5..



Sl. 2.5. *Primjer DOMa jednostavne web stranice*

Nakon što preglednik učita DOM, obrađuje CSS pravila te ih dodjeljuje predviđenim elementima. Tada JavaScript prevoditelj (engl. *interpreter*) u pregledniku prevodi JavaScript skripte i omogućuje njihovo izvođenje. Ovisno o želji programera, JavaScript skripte mogu u bilo kojem trenutku pristupiti DOMu te manipulirati sadržajem web aplikacije, donoseći tako željenu interakciju.

2.3.2. AngularJS

AngularJS je prema [6], strukturni *framework* za dinamične web aplikacije. Zasnovan je na JavaScript programskom jeziku. AngularJS je primarno razvijen zbog izazova koji nastaju pri razvoju jednostraničnih (engl. *single page*) web stranica. Osim za razvoj jednostraničnih web stranica, AngularJS ima puno širu primjenu te se može koristiti i za razvoj svih drugih web aplikacija te je kompatibilan i preporučen *framework* za razvoj mobilnih hibridnih aplikacija uz pomoć Apache Cordovae, za što će se i koristit u praktičnom dijelu ovog rada.

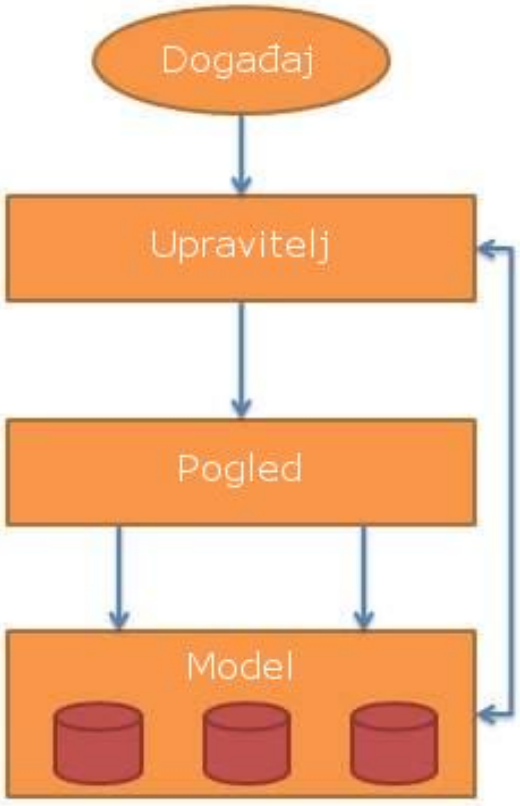
Neke od prednosti korištenja AngularJS *frameworka* su:

- razdvajanje DOM manipulacije od logike aplikacije
- razdvajanje klijentske i poslužiteljske strane unutar aplikacije. Time se omogućuje paralelni razvoj i ponovno korištenje resursa s obje strane. Realizira se uz pomoć ubrizgavanja ovisnosti (engl. *dependency injection*).
- dodjeljivanje strukture pri procesu razvoja aplikacije, od razvoja korisničkog sučelja, do pisanje poslovne logike i testiranje aplikacije
- omogućavanje testiranja aplikacije.

AngularJS koristi MVC arhitekturu pri izradi web aplikacije. MVC (engl. *Model View Controller*) je realizacija softverske arhitekture na način da se pojedini dijelovi aplikacije podijele u komponente koje imaju jasno definirani zadatak. Pojednostavljeni model MVC arhitekture prikazan na slici 2.6..

Kod AngularJSa MVC arhitektura se prema [7] realizira na sljedeći način:

- model (engl. *model*) – upravlja podacima aplikacije. Odgovara na upite od strane pogleda (tj. korisnika) i prihvaća upute sa strane upravitelja kako bi ažurirao svoje podatke.
- pogled (engl. *view*) – HTML struktura sa svojim CSS pravilima i drugim dodatcima koji se prikazuju korisniku u pregledniku. Korisnik putem pogleda može komunicirati s aplikacijom i time izazvati promjene u aplikaciji.
- upravitelj (engl. *controller*) – spojnica između modela i pogleda aplikacije. Upravitelj uz pomoć gledatelja (engl. *watchers*) i probavne petlje (engl. *digest loop*) reagira na promjene i može ažurirati iste u pogledu i modelu.



Slika 2.6. *Prikaz MVC arhitekture* [7]

Prema [8], ubrizgavanje ovisnosti je za razliku od MVC arhitekture, obrazac dizajna (engl. *desing pattern*), koji se brine o komponentama i njihovim ovisnostima. AngularJS ubrizni sustav zadužen je za stvaranje komponenti, dodjeljivanje njihovih ovisnosti te prosljeđivanje komponenti drugim komponentama prema određenom zahtjevu.

Kako bi navedena komunikacija između pojedinih dijelova MVC strukture radila, AngularJS koristi direktive (engl. *directives*) koje se kao atributi dodaju HTML elementima. Pri učitavanju DOMa, direktive ukazuju AngularJSu na koje dijelove aplikacije je potrebno paziti te na koji način. Tako na primjer *ng-app* direktiva određuje početak dijela AngularJS aplikacije, to jest njenog modula, ona se uglavnom nalazi, kao atribut, u *html* elementu.

Povezivanje podataka (engl. *data binding*) također je jedan od bitnih koncepata AngularJS *frameworka*. Pod povezivanjem podataka, se prema [9] misli na ažuriranje podatak između modela i komponenti pogleda. Način na koji AngularJS koristi povezivanje podataka omogućuje korištenje modela kao jedinstveni izvor podataka. Pogled je projekcija modela u svakom trenutku. Kada se model promijeni, pogled odražava tu promjenu te isto vrijedi i suprotno. Većina *frameowrka* slične namjene povezuje podatke samo u jednom smjeru. Takav način razmjene podataka vodi to toga, da nakon što se dogodi spajanje modela i pogleda te prikazivanje rezultata korisniku, više ne postoji automatsko ažuriranje pogleda putem modela ili

modela kroz pogled. To znači da programer mora samostalno pisati kôd koji će neprestano ažurirati podatke pogleda s podacima modela i obratno.

Pomoću servisa (engl. *services*) omogućuje se organiziranje i dijeljenje podataka kroz cijelu aplikaciju. Servisi su prema [10], zamjenjivi objekti koji su međusobno povezani ubrizgavanjem ovisnosti. Svi AngularJS servisi imaju dva bitna obilježja, a to su:

- lijeno instanciranje (engl. *lazily instantiated*) – AngularJS samo instancira određeni servis ako aplikacijska komponenta o njemu ovisi
- *Singletons* – svaka komponenta koja ovisi o nekom servisu dobiva preporuku (engl. *reference*) jedine instance koju je generirao *service factory*.

AngularJS nudi nekoliko korisnih servisa kao što je *$http* servis za komunikaciju aplikacije s resursima na internetu, ali je često, ovisno o potrebi, potrebno napraviti vlastiti. Kako bi se moglo koristiti AngularJS servisom, potrebno je servis dodati kao ovisnost za određenu komponentu (upravitelj, servis, filter ili direktivu), koja ovisi o servisu. AngularJS podsustav ubrizgavanja ovisnosti brine o ostatku.

2.4. Apache Cordova

Apache Cordova, dalje u tekstu samo Cordova, *framework* je otvorenog koda (engl. *open source*) za razvoj hibridnih mobilnih aplikacija. Cordova *frameowrk* razvija se sa strane Apache Software Foundationa ili skraćeno ASFa. ASF je prema [11], osnovan 1999. godine kao dobrotvorna organizacija financirana pomoću donacija pojedinaca i korporativnih sponzora. ASF sa svojim potpuno volonterskim članovima odbora upravlja s više od 350 vodećih projekata otvorenog koda, uključujući Apache HTTP Server, svjetski najpopularniji web poslužiteljski softver.

Prema [12], Cordova omogućuje korištenje standardnih web tehnologija – HTML5, CSS3 i JavaScript za razvoj istih. Aplikacije se izvode unutar omotača (engl. *wrappers*) usmjerenog na svaku platformu i oslanjaju se na povezivanje s aplikacijskim programskim sučeljem ili APIjem (engl. *application programming interface*), kako bi pristupile mogućnostima svakog uređaja, kao što su senzori, mrežni statusi i slično.

2.4.1. Arhitektura

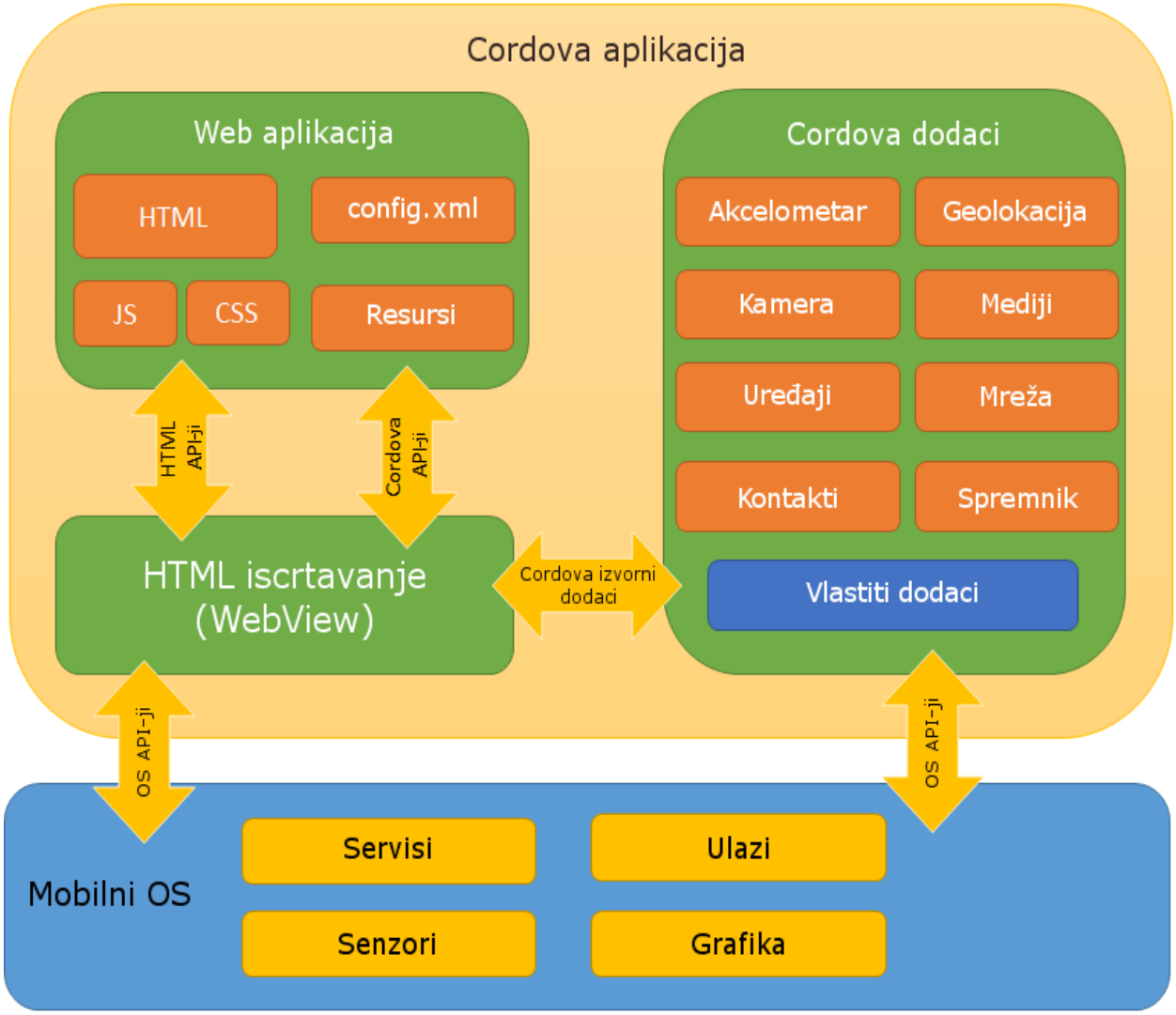
Cordova aplikacija sastoji se od nekoliko komponenti. Na slici 2.7. je prema [12], prikazan dijagram visoke razine arhitekture Cordova aplikacije. Najbitniji dijelovi u arhitekturi Cordova aplikacije su:

- web pogled (engl. *WebView*) – Cordovin omogućen web pogled može aplikaciji pružiti cjelokupno korisničko sučelje. Na nekim platformama to može biti

komponenta unutar veće hibridne aplikacije koja miješa web pogled s izvornim (engl. *nativ*) aplikacijskim komponentama.

- web aplikacija (engl. *WebApp*) – ovo je dio gdje boravi aplikacijski kod. Aplikacija je implementirana kao web stranica koja se zadano sastoji od lokalne datoteke naziva *index.html*, koja pokazuje na CSS pravila, JavaScript kôd, slike ili ostale datoteke potrebne za njeno pokretanje. Aplikacija se izvodi u web pogledu unutar izvornog omotača, koja se kao takva dijeli na trgovine aplikacija. Ova komponenta ima vrlo bitnu datoteku *config.xml*. Ta datoteka sadrži informacije o aplikaciji i određuje parametre koji utječu na njen rad, kao što su odgovori na orijentacijske smjene.
- programski dodaci (engl. *plugins*) – čine sastavni dio Cordova ekosustava. Oni pružaju sučelje za komunikaciju između Cordovae i izvornih komponenti te veze do standardnih APIa uređaja. Takav način rada omogućuje pozivanje izvornog koda iz JavaScripta. Cordova projekt sadrži skup programskih dodataka koji se nazivaju temeljni dodaci (engl. *core plugins*). Temeljni dodaci omogućuju aplikaciji pristup mogućnostima uređaja kao što su baterija, kamera ili kontakti. Osim osnovnih programskih dodataka, postoji nekoliko drugih programskih dodataka koji pružaju dodatne veze do značajki koje nisu nužno dostupne na svim platformama. Željene programske dodatake moguće je pronaći na službenim internetskim stranicama Cordovae. Cordova ne pruža podršku pri razvoju korisničkog sučelja ili arhitekture aplikacija. Cordova samo omogućuju okruženje u kojem ti programski dodaci mogu biti izvedeni.

Cordova pruža podršku razvoja mobilnih hibridnih aplikacija za nekoliko najbitnijih platformi, a to su prema [12] sljedeće: android, blackberry10, ios, ubuntu, firefox os, fire os, wp8 (Windows Phone 8) te windows platformu (8.1, 10, Phone 8.1).



Sl. 2.7. *Prikaz arhitekture Cordova aplikacije* [12]

2.5.Hibridne aplikacije

Hibridna aplikacija je mobilna aplikacija koja sadrži izolirani preglednik, web pogled tj. *WebView*, kako bi pokrenula web aplikaciju unutar izvorne aplikacije. Koristi izvorni omotač koji može komunicirati s izvornom platformom uređaja i s web pogledom. To znači da je moguće pokrenuti web stranicu na mobilnom uređaju kao izvornu aplikaciju te da ista ima pristup mogućnostima uređaja, kao što su kamera i GPS lokacije. Hibridne su aplikacije u biti male web aplikacije pokrenute u pregledničkom omotaču koji ima pristup izvornom platformskom sloju.

Alati koji omogućuju komunikaciju između web pogleda i izvorne platforme omogućuju rad hibridnih aplikacija. Ti potrebni alati nisu dio iOS, Android ili bilo koje druge mobilne platforme, nego su to drugi alati, kao što je Cordova, koja će se i koristiti u praktičnom dijelu

ovog rada. Kada se hibridna aplikacija prevodi u jezik koji računalo, to jest mobilni uređaj, razumije, web aplikacija prevodi se u izvornu aplikaciju za traženi uređaj.

Hibridne aplikacije, naspram izvornih aplikacija i mobilnih web stranica imaju nekoliko prednosti, a to su prema [13]:

- mogućnost razvoja jedne aplikacije za jednu platformu te je tada uz minimalna ulaganja moguće razviti istu aplikaciju za sve druge podržane platforme
- omogućavanje razvoja mobilnih aplikacija sa znanjem tehnologija korištenih u razvoju web stranica i web aplikacija
- pristup svim resursima i mogućnostima kao i kod izvornih aplikacija
- jednostavnost i brzina razvoja.

Hibridne aplikacije pružaju otpornu osnovu za razvoj mobilnih aplikacija, ali ipak omogućuju korištenje web platforme. Većina aplikacije može se razviti kao obična web stranica te kadgod je potrebno pristupiti izvornim APIjima. Hibridni *framework* omogućuje poveznicu (engl. *bridge*) tim APIjima pomoću JavaScripta. Aplikacija tada može otkriti dodire ekrana i ostale geste, isto kao što i web aplikacija može otkriti i reagirati na pritiske miša ili tipkovnice.

Osim velikog broja prednosti hibridnih aplikacija postoje i neki nedostaci pri korištenju i razvoju hibridnih mobilnih aplikacija, kao što su prema [13] sljedeći:

- ograničenja web pogleda – aplikacija može biti pokrenuta samo onoliko dobro koliko i sam web pogled, što znači da je izvođenje aplikacije vezano i ograničeno na kvalitetu izvođenja preglednika na pojedinoj platformi.
- pristupanje izvornim mogućnostima pomoću programskih dodataka – potreban pristup izvornim APIjima možda trenutno nije dostupan pa je u takvom slučaju potrebno prvo razviti programski dodatak koji će podržavati takav pristup
- nedostatak izvornog korisničkog sučelja – bez alata, kao što je Ionic, bilo bi potrebno razviti i posebno prilagoditi sve elemente korisničkog sučelja za pojedinu platformu i u nekim slučajevima i za različite verzije operacijskog sustava ili veličina zaslona.

2.5.1. Ionic

Kako bi se hibridnoj aplikaciji dao izvorni osjećaj korištenja aplikacije, u ovom radu korišten je Ionic. Ionic *framewrok* je prema [14], kombinacija tehnologija i usluga koje služe razvoju korisničkog sučelja hibridnih mobilnih aplikacija. To uključuje vizualne elemente kao što su gumbovi, navigacijske trake, zaglavlja i slično. Takvi elementi razvijeni su pomoću CSSa, HTMLa i JavaScripta te se oni ponašaju kao izvorni elementi, koji se koriste na svakoj platformi. Ionic je razvijen u ekosustavu koji uključuje AngularJS i Cordovau za razvoj strukture i logike

aplikacije te omogućuje pristupanje mogućnostima uređaja na kojem se aplikacija koristi. Ionic u navedenom ekosustavu omogućuje korisniku osjećaj izvorne mobilne aplikacije.

Ionic razvojni tim je u vremenu pisanja ovog diplomskog rada prešao na razvoj Ionic 2 verzije, koja se od korištene verzije u praktičnom dijelu rada, Ionica 1, točnije Ionic verzije 1.3.1 pod nazivom *El Salvador*, razlikuje po nekim bitnim značajkama. Ionic verzija 1 ili Ionic V1 za razvoj hibridne mobilne aplikacije koristi AngularJS 1.x verzije te podržava izvorno korisničko sučelje ili web pogled za iOS 7 i novije verzije iOS mobilnog operacijskog sustava te Android 4.1. i novije Android operacijske sustave. Ionic 2 za razvoj hibridne mobilne aplikacije koristi novu inačicu AngularJS *frameworka* 2.0, koja nad starijom verzijom ima osjetnu prednost u brzini izvođenja, što je vrlo bitno za razvoj hibridnih mobilnih aplikacija. Jedini bitniji nedostatak prelaska na 2.0 inačicu AngularJS *frameworka* jest ta što je sintaksa u AngularJS 2.0 inačici potpuno ili velikim dijelom drugačija te jednostavan prelazak s Ionica V1 na V2 nije sasvim jednostavan. Ionic V2, osim što podržava razvoj mobilnih hibridnih aplikacija, pomoću Cordovae dodaje podršku za razvoj progresivnih web aplikacija (engl. *Progressive Web Apps*) i Electron *frameowrka* za razvoj platformski neovisnih (engl. *cross* platform) desktop aplikacija pomoću web tehnologija, kao što je to i pri razvoju hibridnih mobilnih aplikacija. Ionic V2 pri razvoju hibridnih mobilnih aplikacija podržava iOS 8+ operacijske sustave, Android 4.4.+ operacijske sustave te Windows 10 mobilni operacijski sustav.

Osim što je Ionic, *framework* koji se brine o korisničkom sučelju, on olakšava i samu izradu cjelokupne hibridne mobilne aplikacije i to na način da pruža mogućnost korištenja dodatnih alata. Ti dodatni alati olakšavaju korištenje i integraciju Cordovae i njenih programskih dodataka potrebnih za razvoj hibridne mobilne aplikacije. Omogućuju korištenje gotovih predložaka (engl. *templates*) i velikog broja mobilno prilagođenih ikona. Još jedan vrlo koristan alat iz Ionic obitelji jest Ionic CLI (engl. *Command Line Utility*) to jest Ionic komandna linija. Ionic CLI je primarni alat pri razvoju hibridne mobilne aplikacije pomoću Ionic *frameworka*. CLI sadrži vrlo bitne naredbe kojima se programer može služiti tijekom razvoja aplikacije. Nekim dijelom Ionic CLI koristi Cordova CLI te isti nadograđuje za lakši razvoj hibridne mobilne aplikacije. Ionic CLI potrebno je instalirati na vlastito računalo te se naredbama pristupa preko komandne linije uređaja. Neke od bitnijih naredbi su:

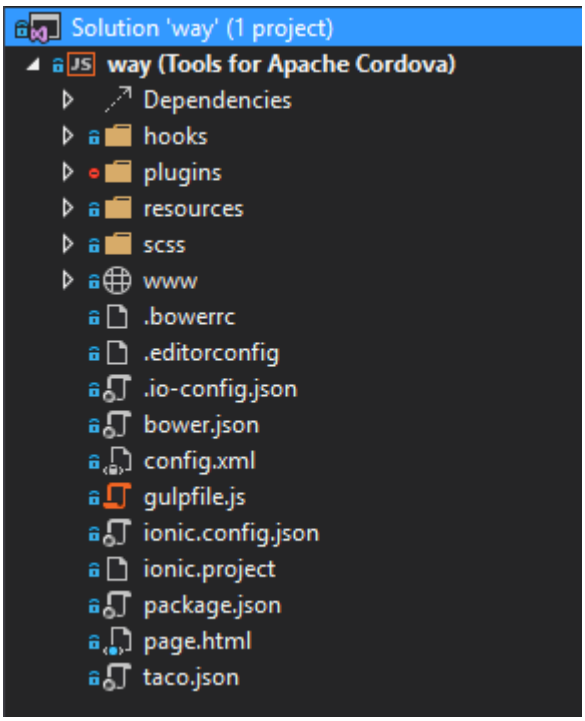
- *start* – naredba koja pokreće novi Ionic projekt sa svim potrebnim integracijama Cordovae i AngularJSa. Moguće je izabrati gotove predloške ili prazan projekt te verziju Ionic *frameworka*.
- *build* – priprema i prevodi Ionic projekt za određenu platformu

- *serve* – pokreće lokalni server u kojem pokreće hibridnu aplikaciju u pregledniku računala
- *platform* – dodaje resurse potrebne za razvoj aplikacije za određenu platformu
- *info* – daje informacije o verzijama potrebnih programskih paketa (Cordova CLIa, Ionic CLIa, Ionic *frameworka*, Nodea).

Bitno je napomenuti kako za razvoj hibridne mobilne aplikacije Ionic nije obavezan. Umjesto Ionica mogu se koristiti drugi *frameowrci* za oblikovanje korisničkog sučelja, kao što su Bootstrap ili jQuery Mobile. Također je moguće razviti hibridnu mobilnu aplikaciju bez korištenja dodatnih *frameworka*, ali takav pristup razvoju hibridne mobilne aplikacije dodaje jako puno vremena sveukupnom razvoju aplikacije te se takvim pristupom ne dobiva na kvaliteti proizvoda. Ionic s druge strane, u usporedbi s drugim navedenim *frameworcima*, aktivno podržava korištenje Cordova i AngularJS okruženja te pruža dodatnu fleksibilnost razvoju hibridne mobilne aplikacije.

2.5.2. Struktura Ionic aplikacije

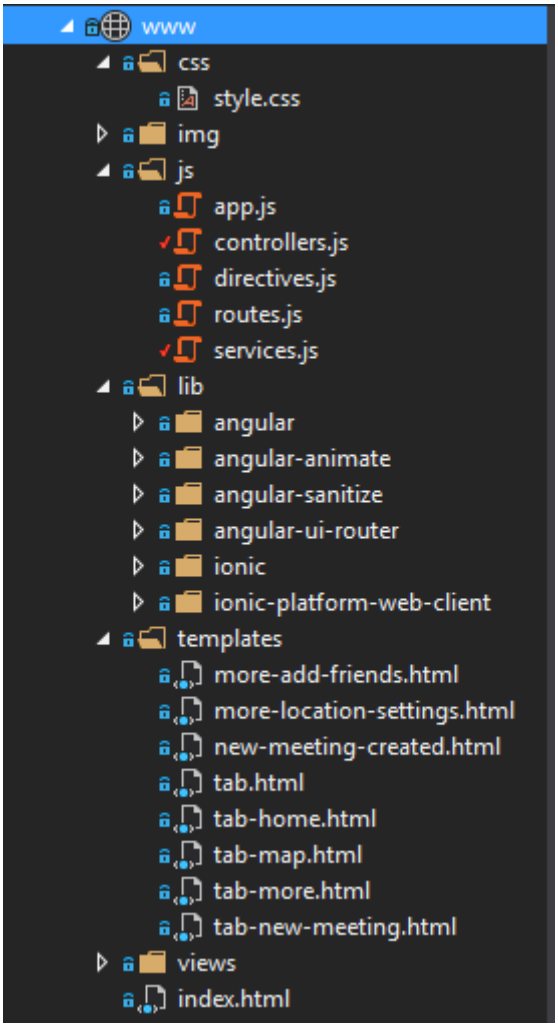
Ionic aplikacije, to jest hibridne mobilne aplikacije, napravljene su pomoću Cordovae. Iz toga proizlazi da aplikacije napravljene u Ionic okruženju koriste Cordova strukturu podataka. Prikaz jedne uobičajene strukture Ionic aplikacije pokazana je na slici 2.8..



Sl. 2.8. *Prikaz strukture Ionic aplikacije*

Neki od važnijih direktorija i datoteka prikazani slikom 2.8. su:

- *plugins* – u ovom direktoriju spremaju se programski dodaci koje programer dodaje u svoj projekt. Programski dodaci dodaju se pomoću Ionic CLIa naredbom „*ionic plugin add {ime željenog programskog dodatka}*“.
- *scss* – ovaj direktoriji sadrži SASS datoteke aplikacije. SASS (engl. *Syntactically Awesome Stylesheets*) skriptni je jezik koji nadograđuje mogućnosti CSSa. Nije ga potrebno koristiti te u praktičnom dijelu ovog rada i nije korišten, ali se pri većim projektima s kompleksnijim grafičkim sučeljem svakako preporučuje korištenje SASSa.
- *www* – u ovom direktoriju obavlja se sam razvoj hibridne mobilne aplikacije te je struktura tog direktorija slična strukturi uobičajene AngularJS aplikacije. Primjer strukture *www* direktorija prikazana je slikom 2.9..



Sl. 2.9. *Prikaz strukture www direktorija*

Unutar *www* direktorija strukturu aplikacije programer može prilagoditi svojim potrebama i preferencijama, s tim da je preporučena struktura automatski postavljena od strane Ionica. Na

slici 2.9. vidljivo je da hibridna mobilna aplikacija sadrži sve uobičajene datoteke i direktorije uobičajene web aplikacije. *www* direktoriji tako sadrži CSS datoteke sa stilskim pravilima, koji određuju izgled aplikacije, JavaScript datoteke, koje služe upravljanje resursima pomoću AngularJS *frameworka* i HTML datoteke, koje sadrže strukturu pojedine stranice aplikacije ili samo nekog određenog dijela. Unutar *lib* direktorija nalaze se AngularJS programski dodaci, koji aplikaciji dodaju dodatne mogućnosti pa tako programski dodatak *ui-router* olakšava navigaciju kroz aplikaciju, a *animate* programski dodatak olakšava dodavanje animacija.

2.5.3. Razvoj Ionic korisničkog sučelja

Kako je Ionic ujedno i CSS i JavaScript *framework*, moguće je koristiti samo jedno ili drugo te kombinaciju jednog i drugog. Ionic kao CSS *framework* programeru omogućuje korištenje gotovih CSS klasa, koje HTML elementima daju prirodni i izvorni izgled na svim mobilnim uređajima. Osim gotovih klasa, koje stoje na raspolaganju programeru, postoji i velik broj ugrađenih klasa, koje bez dodatnog utjecaja programera aplikaciji daju osnovni izvorni izgled.

Osim uobičajenog dodavanja CSS klasa HTML elementima, Ionic omogućuje prepoznavanje pojedinih platformi. Prepoznavanje na kojoj se platformi aplikacija izvodi omogućuje prilagođavanje korisničkog sučelja ovisno o platformi te i sam Ionic sa svojim ugrađenim klasama različito stilizira korisničko sučelje ovisno o platformi na kojoj se hibridna mobilna aplikacija izvodi. Ionic se tako pridržava preporučenih uputa za stiliziranje korisničkog sučelja ovisno o platformi. Na primjer, preporučeni položaj naslova kartica kod iOS platforme na vrhu je kartice centriran u sredini, dok je preporučen položaj naslova kartice za Android platformu s poravnanjem na lijevoj strani. Ionic tako prema preporučenim uputama, samostalno i automatski brine o tome na kojoj se platformi nalazi i prema tome primjenjuje stilove za uređenje korisničkog sučelja, ako programer, naravno, nije odredio drugačije. Još jedan primjer je brzina animacija između prebacivanja s kartice na karticu koja je na iOS platformi prema preporuci sporija nego ista na Android platformi. Sve to i još puno više Ionic odrađuje samostalno te programeru ostaje više vremena za razvoj logike i testiranje aplikacije. Prepoznavanje platforme Ionic vrši automatski te HTML *body* elementima, ovisno na kojoj platformi se izvodi aplikacija, dodaje *platform-ios, platform-android, platform-windowsphone* ili neku drugu *platform* klasu. Osim *platform* klasa, Ionic *body* elementima postavlja klase ovisne o verziji operacijskog sustava. Ako se, na primjer, aplikacija izvodi na Android platformi s verzijom 4.4. operacijskog sustava, Ionic, osim *platform-android* klase, dodaje i *platform-android4* i *platform-android4_4* klasu. Takav pristup omogućuje posebno stiliziranje svake platforme i pojedine verzije platforme. Moguće je koristiti bilo koju od prije navedenih klasa i prepisati ju vlastitim

stilovima. Ako se, na primjer, prikaz naslova zaglavlja želi promijeniti u velika tiskana slova, samo za Android platformu, može se koristiti CSS klasa *"platform-android .bar-header {text-transform: uppercase;}"*.

Kako je pomoću ugrađenih CSS klasa i automatskog prepoznavanja platforme moguće urediti korisničko sučelje, tako je uz pomoć JavaScripta moguće urediti i prilagoditi izgled i ponašanje korisničkog sučelja. Ionic pomoću JavaScripta posjeduje posebne HTML elemente ili direktive, koji ovisno o nazivu imaju posebnu primjenu u hibridnoj mobilnoj aplikaciji. Tako na primjer, *ionView* direktiva ima ulogu spremnika pogleda (slično kao standardni HTML *div* element). Osim uloge spremnika pogleda, *ionView* direktiva sadrži navigacijske informacije te informacije o zaglavlju. Roditelj *ionView* direktive je *ionNavView* element (kao što je *body* element roditeljski element bilo kojem *div* elementu kojeg sadrži). *ionNavView* direktiva služi kao glavni spremnik u kojem se navigacijom kroz aplikaciju izmjenjuje više *ionView* direktiva. Kada *ionView* direktiva pristupa ili izlazi iz roditeljske *ionNavView* direktive, pogled mu predaje informacije kao što su naslov, treba li prikazati tipku za nazad, treba li prikazati odgovarajući *ionNavBar*, koju animaciju za prijelaz u drugi pogled treba prikazati te u koju stranu se animacija treba obaviti. Kako bi se poboljšale performanse, pogledi su spremljeni u privremenu memoriju. Kada se navigacijom prijeđe u drugi pogled, podaci o starom pogledu ostaju u DOMu. Kada se navigacijom kroz aplikaciju dođe do pogleda koji je spremljen u privremenoj memoriji, jednostavno se traženi pogled postavlja kao aktivni, ali se ovaj put ti podaci već nalaze u DOMu pa podatke nije potrebno ponovno učitavati. Spremanje podataka o pogledu u lokalnu memoriju i broj pogleda koji će se spremiti u DOM mogu se prilagoditi prema potrebi.

3. RAZVOJ HIBRIDNE MOBILNE APLIKACIJE

Razvoj hibridne mobilne aplikacije, kao što je već navedeno, donosi mnoge prednosti nad uobičajenim razvojem mobilne aplikacije. Najveća prednost je zasigurno razvijanje programskog kôda u samo jednom programskom okruženju, a istovremeno pokrivanje više mobilnih platformi. Također, najveća prednost hibridne mobilne aplikacije dovodi do nekih nedostataka i problematika. U sklopu ovog diplomskog rada razvijena je hibridna mobilna aplikacija pomoću uobičajenih web tehnologija, kao što su HTML, CSS i JavaScript te su uz to korišteni dodatni alati kao AngularJS, Cordova i Ionic, koji su bitni za hibridni dio ovakve aplikacije. Pri razvoju je vrlo bitno shvatiti da se ne razvija obična web aplikacija, koja pomoću navedenih dodataka postaje hibridna i mobilna. Potrebno je pripaziti na arhitekturu aplikacije, obavljanje komunikacije s poslužiteljem te najviše na razvoj korisničkog sučelja. Korisničko sučelje, unatoč tome što se ono razvija web tehnologijama, mora biti u potpunosti prilagođeno korištenju na pametnim uređajima. U obzir se moraju uzeti preporučena stilska obilježja za pojedinu platformu, veličina zaslona mobilnih uređaja, koja je osjetno manja od uobičajenih veličina zaslona na kojima se izvode web aplikacije te način na koji će korisnik pristupati resursima, kao što su GPS podaci, podaci u lokalnoj memoriji i slično.

Tijekom razvoja i testiranja praktičnog dijela rada, to jest hibridne mobilne aplikacije, korišteno je nekoliko platformi i uređaja. Pri razvoju hibridne mobilne aplikacije bitno je aplikaciju, osim u pregledniku i dobrim emulatorima danih od strane Ionic platforme, testirati i na pravim fizičkim uređajima. Platforme i uređaji koji su korišteni za razvoj navedene hibridne mobilne aplikacije su sljedeći:

- iOS platforma s Iphone 5S uređajem (OS verzija 9.3.5)
- Android platforma s:
    - LGP990 uređajem (OS verzija 4.2.2)
    - Samsung S4 uređajem (OS verzija 5.0.1)
- Browser platforma s Google Chrome preglednikom (Windows OS verzija 52.0.2743.116 m).

Bitno za napomenuti je da je praktični dio ovog rada samo pokazna aplikacija, kojoj nedostaju neke značajke korisničkog sučelja te su razvijene samo najbitnije funkcionalnosti aplikacije zadane u temi ovog rada.

3.1.Načini pristupa lokaciji u hibridnim aplikacijama

Postoji nekoliko načina pristupanja podacima o trenutnoj lokaciji mobilnog uređaja putem hibridne mobilne aplikacije. Prema [15], korištenjem odgovarajućeg Cordova programskog dodatka, mogući načini pristupanja podataka o trenutnoj lokaciji uređaja korisnika su putem:

- GPS podataka
- podataka dobivenih putem mrežnih signala kao što su: IP adrese, RFID, WiFi i Bluetooth MAC adrese
- *GSM*/*CDMA* stanice mobilne mreže.

3.1.1. Pristup lokaciji putem GPS podataka

GPS (engl. *Global Positioning System*) je prema [16], mreža od oko 30 satelita u orbiti Zemlje na visini od 20000 kilometara. GPS sustav izvorno je razvijen od strane američke vlade za vojno navigiranje, ali sada svatko s GPS uređajem, bio to navigacijski, mobilni ili jednostavno samo GPS uređaj, može primati radio signale koje sateliti odašilju.

Gdje god bili na Zemlji, najmanje četiri satelita „vidljivo“ je svakog trenutka. Svaki od njih prenosi informacije o svom položaju i trenutno vrijeme u pravilnim razmacima. Te signale, koji putuju brzinom svijetlosti, primaju GPS uređaji na Zemlji, koji tada računaju udaljenost pojedinog satelita, ovisno o tome koliko je poruci bilo potrebno vremena da stigne. Kada uređaj zna koja je udaljenost najmanje tri satelita, GPS uređaj korisnika može odrediti njegovu trenutnu lokaciju koristeći metodu trilateracije (engl. *trilateration*).

3.1.2. Pristup lokaciji putem mrežnih signala

Neovisno o platformi koju korisnik mobilnog uređaja koristi, podaci o trenutnoj lokaciji automatski se u pozadini operacijskog sustava šalju poslužitelju platforme. Podaci koji se šalju mogu biti trenutna GPS lokacija, podaci o WiFi ili Bluetooth mrežama, kojima mobilni uređaj ima pristup, ali nije nužno spojen na njih, podaci o trenutnoj mobilnoj mreži, koju korisnik koristi te točna stanica koju korisnik koristi za pristupanje svojoj mobilnoj mreži. Sa svim tim podacima obavlja se „mapiranje“ prostora oko korisnika. Tako je moguće da i bez GSP podataka Google ili bilo koji drugi servisi, mogu okvirno saznati lokaciju uređaja.

Ovaj pristup dohvaćanja podataka o lokaciji uređaja je uređajima, koji nemaju ugrađeni GPS uređaj, jedina mogućnost pristupa takvim podacima. Problem kod određivanja lokacija takvim podacima je taj što se kao lokacija korisnika uređaja vraća lokacija, npr. WiFi pristupne točke, čija je točna lokacija poznata putem MAC adrese. To vodi do toga da korisnik dijeli lokaciju s uređajem na koji je spojen WiFi vezom (npr. kućnim ruterom), što ujedno znači i neprecizne rezultate. Sličan problem pojavljuje se pri dijeljenju internetske veze (engl. *HotSpot*) s uređajem

koji nema ugrađeni GPS uređaj. U takvim slučajevima, uređaj nema za trenutnu lokaciju postavljenu lokaciju uređaja koji s njim dijeli internetsku vezu, već zadnju lokaciju koja se dodijelila uređaju prije povezivanja na dijeljenu mrežu. S druge strane, uređaji koji imaju ugrađeni GPS uređaj te koriste dijeljenju vezu za komunikaciju s udaljenim poslužiteljem, mogu odrediti vlastitu lokaciju, ali s određenim zakašnjenjem. Isprva se kao trenutna lokacija pokazuje ona lokacija na kojoj je uređaj bio spojen prije spajanja na dijeljenju mrežu te se nakon spomenutog zakašnjenja lokacija ažurira na stvarnu trenutnu. Takav način rada događa se jer je ugrađenom GPS uređaju potrebno vremena, ovisno o uređaju, da nađe pravu trenutnu lokaciju, što uostalom može i ovisiti i klimatskim uvjetima (pri oblačnom vremenu GPS uređaju je potrebno više vremena ili uopće nije moguće odrediti stvarnu lokaciju uređaja). Takve iznimke pri upravljanju lokacije uređaja treba uzeti u obzir i o njima brinuti.

Kako se hibridna mobilna aplikacija izvodi kao web stranica u pregledniku, samo s dodatnim omotačima koji joj daju izvorni izgled i ponašanje, Cordova za dohvaćanje lokacije uređaja korisnika koristi HTML geolokacijski API. HTML geolokacijski API tada, osim GPS podacima uređaja, ima mogućnost pristupa i drugim APIjima za dohvaćanje lokacije korisnika. Ukoliko GPS podaci nisu dostupni, preglednik može koristiti Google Maps API. Koji dodatni API će preglednik koristiti ovisi o samoj implementaciji preglednika na izvornu platformu te je to uglavnom Google Maps API servis.

Google servis za upravljanje lokacijom, koji Cordova programski dodatak može, a i ne mora koristiti pri dohvaćanju lokacije uređaja korisnika, ovisno o pregledniku, prema [17], prima JSON objekte, koji sadrže sljedeće podatke:

- *homeMobileCountyCode* – državni pozivni broj mobilne mreže u kojoj je prijavljen uređaj
- *homeMobileNetworkCode* – mrežni pozivni broj mobilne mreže u kojoj je prijavljen uređaj
- *radioType* – vrsta korištene mreže, kao na primjer, LTE, GSM, CDMA i WCDMA
- *carrier* – ime pružatelja usluge mobilne mreže uređaja
- *considerIp* – određuje je li potrebno određivanje lokacije putem IP adrese, ako WiFi i podaci mrežnih stanica nisu dostupni
- *cellTowers* – niz objekata stanica mobilne mreže
- *wifiAccessPoints* – niz objekata WiFi pristupnih točki. Kako bi bili postavljeni podaci, moraju postojati najmanje dvije pristupne točke te se mora postaviti

macAdress parametar, koji sadrži MAC adresu (jedinstveni identifikator fizičkog uređaja) WiFi čvora.

3.1.3. Pristupanje lokaciji putem stanica mobilne mreže

Kako je već navedeno, Google servisi za upravljanje lokacijom uređaja primaju JSON objekte s određenim podacima s uređaja korisnika. Jedan od parametara tog JSON zapisa je *cellTowers* objekt, koji može sadržavati nula ili niz više objekata stanica mobilne mreže. Niz objekata *cellTowers* sasdrži sljedeće vrijednosti:

- *cellId* – obavezan parametar koji sadrži jedinstveni identifikator stanice
- *locationAreaCode* – obavezan parametar koji sadrži kôd lokalnog područja
- *mobileCountryCode* – obavezan parametar koji sadrži mobilni državni kôd
- *mobileNetworkCode* – obavezan parametar koji sadrži mobilni mrežni kôd
- *age* – broj u milisekundama za vrijeme od kada je stanica zadnji puta bila primarna
- *signalStrength* – snaga radio signala mjeren u dBm
- *timingAdvance* – vrijeme potrebno da signal dođe od stanice do uređaja.

3.2. Programski dodaci potrebni za pristup lokaciji u Apache Cordovi

Cordova programski dodatak za pristup podacima o lokaciji korisnika hibridne mobilne aplikacije naziva se *cordovaPluginGeolocation.* Prema [15], programski dodatak definira globalni *navigatorGeolocation* objekt. Bitno je primijetiti kako značajke navedenog programskog dodatka postaju dostupne tek nakon *deviceready* događaja. *deviceready* događaj, događa se kada je uređaj, nakon pokretanje hibridne mobilne aplikacije, potpuno spreman za daljnje upute. Informacije o spremnosti uređaja i ostalim podacima o uređaju Cordova dohvaća putem drugog programskog dodatka pod nazivom *cordovaPluginDevice*. U praktičnom dijelu ovog rada nije direktno korišten preporučeni Cordova način određivanja spremnosti uređaja, koji koristi JavaScript *document* objekt te „osluškuje“ uz pomoć *addEventListener* metode, je li uređaj spreman za daljnje upute. Kako se osim Cordovae koristio i Ionic *framework*, korištena je ugrađena Ionic metoda pod nazivom *ionicPlatformReady* (navedena Ionic funkcija u pozadini koristi Cordova programske dodatke), koja kada je uređaj spreman za daljnje upute obavlja zadani zadatak, u ovom slučaju je to izvršavanje funkcije koja dohvaća informacije o lokaciji uređaja putem *navigatorGeolocation* objekta, koji omogućuje *cordovaPluginGeolocation* programski dodatak.

Platforme koje podržava programski dodatak *cordovaPluginGeolocation* sljedeće su:

- Amazon Fire OS
- Android
- BlackBerry
- Firefox OS
- iOS
- Tizen
- Windows Phone 7 i 8
- Windows.

3.2.1. Metode *cordovaPluginGeolocation* programskog dodatka

Korištenjem *cordovaPluginGeolocation* programskog dodatka omogućeno je korištenje *navigatorGeolocation* metoda.

Najbitnija metoda za dohvaćanje lokacije hibridnih mobilnih aplikacija je *navigatorGeolocationGetCurrentPosition* metoda. Metoda pri uspješnom izvršavanju vraća povratnu (engl. *callback*) funkciju (koju je obavezno definirati u pozivu metode), unutar koje se nalazi *position* objekt s podacima o trenutnoj lokaciji uređaja. Ako dođe do greške pri dohvaćanju lokacije uređaja, metoda može vratiti (nije obavezno te se izvodi samo ako je metoda tako definirana) povratnu funkciju unutar koje se nalazi *positionError* objekt, koji sadrži kôd ili opis greške koja se dogodila. Kao treći parametar, *navigatorGeolocationGetCurrentPosition* metoda prima *geolocationOptions* objekt. *geolocationOptions* objekt sadrži parametre kojima je moguće prilagoditi rad *navigatorGeolocationGetCurrentPosition* metode. Parametri koje je moguće postaviti su:

- *enableHighAccuracy* (boolean) – određuje preciznost dohvaćene lokacije. Početno zadana vrijednost je *false* te to znači da će se lokacija pokušati dohvatiti pomoću mrežnih signala. Takav pristup dohvaćanja lokacije je brži i troši manje resursa, te je iz tog razloga i postavljen kao početni. Postavljanjem ove vrijednosti na *true* govori metodi da je potrebno koristiti dohvaćanje lokacije uređaja s većom preciznošću, to jest GPS metodom ili i drugim metodama ako je to potrebno.
- *timeout* (broj) – određuje najduže dozvoljeno vrijeme (u milisekundama) između poziva *navigatorGeolocationGetCurrentPosition* metode i poziva funkcije, koja se poziva kroz *navigatorGeolocationGetCurrentPosition* metodu pri uspješnom dohvaćanju lokacije uređaja. Ako povratna funkcija s podacima o lokaciji uređaja nije

pozvana u definiranom vremenu, poziva se povratna funkcija u slučaju greške s povratnim kôdom greške *PositionError.TIMEOUT*.

- *maximumAge* (broj) – određuje prihvaćanje podataka o lokaciji uređaja koji su spremljeni u privremenoj memoriji. Podaci u privremenoj memoriji ne smiju biti stariji od vrijednosti *maximumAge* parametra (u milisekundama).

Pojednostavljeno upravljanje *navigatorGeolocationGetCurrentPosition* metodom, uz prikaz povratnih funkcija i dodatnih komentara, prikazano je slikom 3.1..

```
// Provjera spremnosti uređaja
ionic.Platform.ready(function () {
    initMap();
});

// Metoda za dohvacanje lokacijskih podataka s povratnim funkcijama
function initMap() {
    navigator.geolocation.getCurrentPosition
    (onMapSuccess, onMapError, options);
}

// Povratna funkcija s dobivenim lokacijskim podacima u position objektu
var onMapSuccess = function (position) {
    Latitude = position.coords.latitude;
    Longitude = position.coords.longitude;
}

// Povratna funkcija s ispisom greške
function onMapError(error) {
    console.log('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

// Postavljanje opcija za dohvacanje lokacije
var options = { timeout: 6000, enableHighAccuracy: true };

// Pracenje promjena lokacijskih podataka
var id = navigator.geolocation.watchPosition(Updatesuccess, onMapError, options);

// Povratna funkcija s promjenjenim lokacijskim podacima
var Updatesuccess = function (position) {
    var updatedLatitude = position.coords.latitude;
    var updatedLongitude = position.coords.longitude;
}
```

Sl. 3.1. *Primjer korištenja Cordova programskih dodataka*

Metoda *navigatorGeolocationWatchPoosition* vraća trenutne podatke o lokaciji uređaja kada je otkrivena promjena lokacije. *navigatorGeolocationWatchPoosition* metoda sadrži jednake povratne funkcije te opcije kao i prethodna *navigatorGeolocationGetCurrentPosition* metoda. Zanimljiva značajka *navigatorGeolocationWatchPoosition* jest ta da se izvodi asinkrono i

automatski pri promjeni lokacijskih podataka. Osim navedenih parametara, metoda *navigatorGeolocationWatchPoosition* vraća *watch* identifikator, koji se koristi u kombinaciji s *navigatorGeolocationClearWatch* metodom kako bi se prestale pratiti promjene lokacijskih podataka uređaja korisnika.

3.2.2. Problemi pri korištenju *cordovaPluginsGeolocation* programskog dodatka

Pri izradi praktičnog dijela rada te tijekom korištenja *cordovaPluginsGeolocation* programskog dodatka primijećeni su neki problemi s obzirom na funkcionalnost programskog dodatka pri izvođenju na nekim mobilnim platformama te u web pregledniku.

Jedan od problema primijećen pri izvođenju *navigatorGeolocationGetCurrentPosition* metode je neizvođenje niti jedne povratne funkcije, to jest, neizvođenje povratne funkcije u slučaju uspješno dohvaćenih lokacijskih podataka niti izvođenje povratne funkcije koja ukazuje na grešku pri dohvaćanju lokacijskih podataka. Metoda *navigatorGeolocationGetCurrentPosition* iz nepoznatih razloga, koji također nisu dokumentirani u službenoj dokumentaciji, zastaje u tijelu metode i onemogućuje daljnje izvođenje drugih funkcija i metoda. Jedno rješenje, koje je lakše obaviti u web pregledniku, ponovno je učitavanje web stranice, dok isto nije moguće (barem ne na tako jednostavan način) pri pokretanju aplikacije kao izvorne na mobilnom uređaju. Rješenje navedenog problema prikazano je slikom 3.2.. Kako bi se izbjeglo neodazivanje metode pri dohvaćanju lokacijskih podataka, prvo se poziva ista *navigatorGeolocationGetCurrentPosition* metoda, samo bez pravih povratnih funkcija i ostalih parametara te se metoda tada poziva na uobičajeni naziv i izvodi bez greške na svim bitnijim mobilnim platformama i u web pregledniku.

```
navigator.geolocation.getCurrentPosition(function () { }, function () { }, {});
navigator.geolocation.getCurrentPosition
(onMapSuccess, onMapError, options);
```

Sl. 3.2. *Prikaz rješenja problema s Cordova programskim dodatkom*

Sljedeći problem uočen pri razvoju i testiranju praktičnog dijela rada te korištenju *cordovaPluginsGeolocation* programskog dodatka je postavljanje *enableHighAccuracy* parametra *geolocationOptions* objekta pri pozivu *navigatorGeolocationGetCurrentPosition* metode. Kako bi metoda dobro radila, potrebno je navedeni parametar postaviti na *true* vrijednost, što znači korištenje GPS metode (prema potrebi i drugih metoda) za dohvaćanje lokacijskih podataka uređaja. U suprotnom, podaci koje metoda dohvaća često su ili pogrešni ili ne dolazi do ažuriranja podataka.

3.3. Pozadinski servisi kao usluga

Pozadinski servisi kao usluga ili BaaS (engl. *Backend as a Service*), također poznat i pod imenom MBaaS (engl. *Mobile Backend as a Service*), su prema [18], način spajanja web i mobilnih aplikacija sa servisima zasnovanima u oblaku (engl. *cloud*). Umjesto korištenja međuprograma (engl. *middleware*) i samostalno razvijanje pozadinskog sustava, BaaS stvara API i SDK (engl. *Software Development Kit*), koji omogućuju komunikaciju između web i mobilne aplikacije s, na primjer, spremnikom u oblaku. Takav pristup razvijanju web ili mobilne aplikacije, programeru omogućuje fokusiranje na korisničko sučelje, bez trošenja vremena na razvoj samostalnog servisa. Usluge koje uobičajeni BaaS sustav nudi su: notifikacije (engl. *push notifications*), integracija sa socijalnim medijima (engl. *social networking integration*), lokacijske usluge (engl. *location* services) i upravljanje korisnikom (engl. *user management*).

3.3.1. Ionic platforma

Ionic, osim što nudi *framework* za razvoj hibridnih mobilnih aplikacija, koji je fokusiran na korisničko sučelje, prema [19], nudi i sljedeće BaaS usluge:

- Ionic Push – omogućuje objavljivanje notifikacija korisnicima Ionic aplikacije. Notifikacije korisniku mogu dati informacije o aplikaciji, iako korisnik u tom trenutku možda ne koristi pametni mobilni uređaj. Putem Ionic platforme moguće je stvaranje notifikacija, koje će biti poslane korisniku kada korisnik ispuni neki određeni kriteriji. Slanje notifikacija se na Android i iOS platformi obavlja putem GCM (engl. *Coogle Cloud Messaging*) i APNs (engl. *Apple Push Notification Service)* servisa. Svaki od navedenih servisa zahtijeva određeno vrijeme za njegovo postavljanje i konfiguraciju te se razlikuju prema načinu upotrebe i funkcionalnosti. Ionic Push servis djeluje kao posrednik između GCMa, APNsa i Ionic aplikacije. Servis nudi API, izvještaj notifikacija, spremnik za informacije o uređaju i dodatne mogućnosti, koje se ne mogu ostvariti koristeći APNs i GCM servise.
- Ionic Users – (korišten u izradi praktičnog dijela rada) – omogućuje registraciju i verifikaciju korisnika. Svaka aplikacija, koja pruža mogućnost stvaranja korisničkog računa i komunikaciju među korisnicima, sadrži način registracije te verifikacije korisnika. Ionic Users servis, osim APIa, koji olakšava upravljanje registracijom i verifikacijom korisnika, omogućuje i web sučelje, pomoću kojeg programer može pristupiti informacijama o pojedinom korisniku.
- Ionic Deploy – omogućuje ažuriranje aplikacije na zahtjev, bez korištenja dugotrajnih i kompliciranih procesa kroz Trgovinu Google Play usluge za Android platformu te

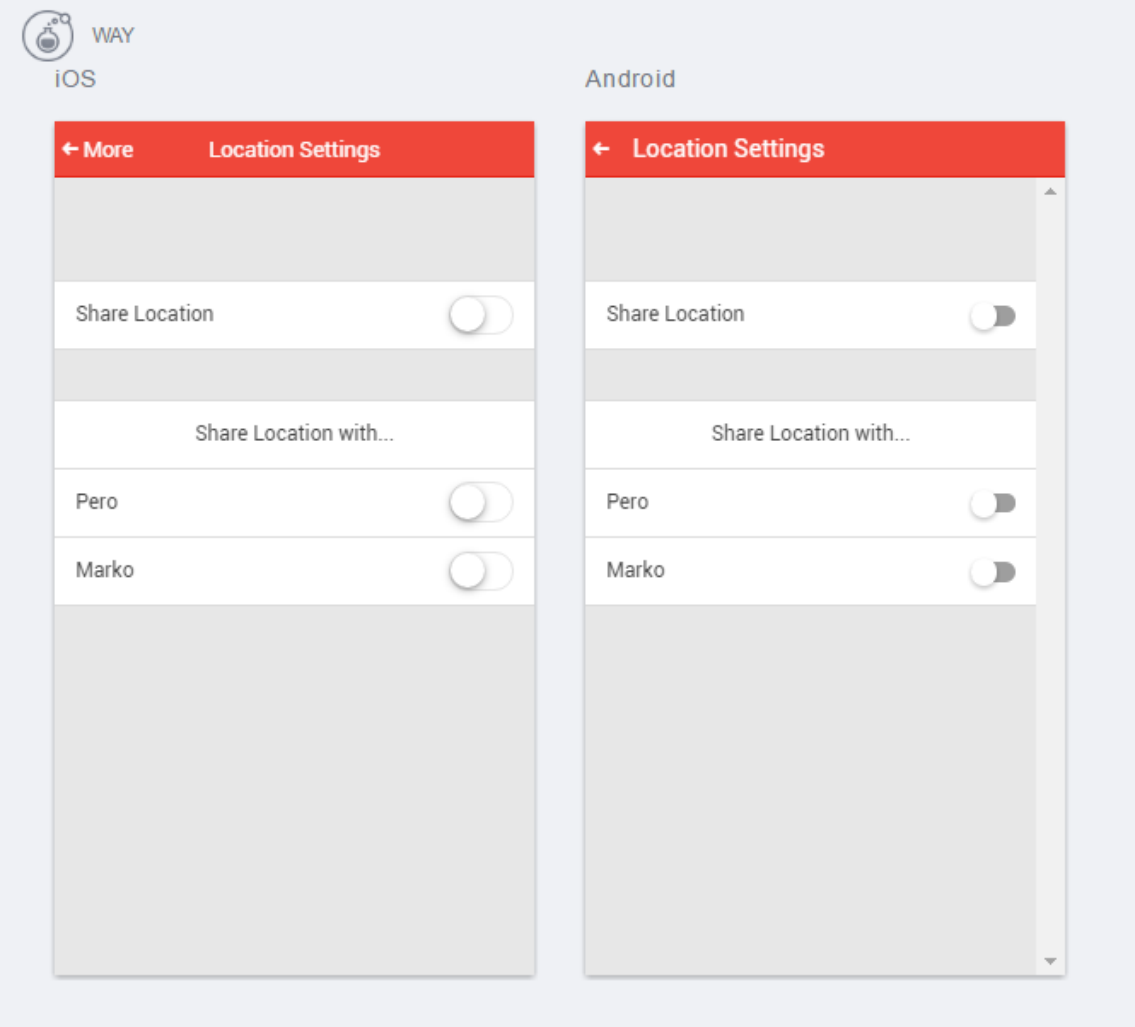
App Store usluge za iOS platformu. Ažuriranje aplikacije na zahtjev ograničeno je tako što je ažuriranje na zahtjev omogućeno samo za promjene HTML, CSS, JavaScript i multimedijskih datoteka, koje se nalaze u *www* direktoriju projekta. Ako se, na primjer, želi dodati novi programski dodatak, potrebno je koristiti uobičajeni proces za ažuriranje mobilne aplikacije. Osim ažuriranja na noviju verziju, pomoću navedenog servisa, moguće je aplikaciju vratiti na jednu od starijih verzija.

- Ionic Package – omogućuje izgradnju (engl. *build*) aplikacije za distribuciju. Glavna namjena Ionic Package servisa jednostavno je slanje aplikacije drugima, izgradnja aplikacije za platforme, koje računalo programera ne podržava (npr. omogućuje izgradnju iOS aplikacije na Windows platformi) te dohvaćanje APK i IPA datoteka koje je moguće predati Trgovini Google Play za Android (APK) ili App Storeu za iOS (IPA) platformu.
- Ionic Analytics – omogućuje praćenje ponašanja korisnika Ionic aplikacije. Servis, na primjer, može pratiti koliko je aktivnih korisnika kojeg dana i ta je mogućnost ugrađena kao zadana mogućnost. Također je moguće dodati vlastite događaje (engl. *event*) koje servis može pratiti.
- Security Profiles – omogućuje grupiranje Android i iOS akreditiva u jedan profil. Tada se ti podaci mogu koristiti od strane drugih servisa.
- API – omogućuje standardne i dodatne mogućnosti, koje u nekim servisima nisu podržane. Ionic API zahtjeva verifikaciju aplikacije zasnovanu na tokenima, kojima se pristupa putem Ionic platforme. API također zahtjeva i korištenje HTTP metoda, koje omogućuje AngularJS.

Osim navedenih servisa Ionic platforma sadrži neke dodatne alate, koji olakšavaju razvoj hibridnih mobilnih aplikacija. Neki od dodatnih alata su:

- Ionic Lab – desktop aplikacija za Mac, Windows i Linux operacijske sustave. Pomoću skladnog korisničkog sučelja omogućuje jednostavno korištenje nekih od navedenih Ionic servisa. Omogućuje uz to i velik dio CLI funkcionalnosti, kao što je na primjer, pokretanje aplikacije na uređaju ili emulatoru.
- Ionic View – omogućuje jednostavno testiranje Ionic aplikacija na više uređaja, bez da je potrebno izgraditi i distribuirati aplikaciju. Moguće je Ionic aplikaciju poslati bilo kome s iOS ili Android uređajem te instaliranom Ionic View aplikacijom. Ionic View aplikacija ne podržava sve programske dodatke omogućene sa strane Cordovae

- Ionic Serve – omogućuje pokretanje Ionic aplikacije lokalno u pregledniku uređaja. Ionic Serve uz Ionic View može poslužiti kao alternativa testiranju hibridnih mobilnih aplikacija na fizičkim uređajima. Takav pristup nije preporučen, ali u danim situacijama s malo sredstava i vremena može biti vrlo dobra zamjena. Pokretanjem aplikacije u pregledniku moguće je pokrenuti i provjeru, kojom se u stvarnom vremenu može pratiti rad aplikacije. Ako se Ionic Serve usluga izvodi s dodatkom -*lab*, dobije se aplikacija koja se lokalno izvodi u pregledniku, ali ujedno simulira rad aplikacije na Android i iOS platformi. Primjer Ionic Serve usluge s –*lab* dodatkom pokazana je slikom 3.3..



Sl. 3.3. *Prikaz Ionic Serve usluge*

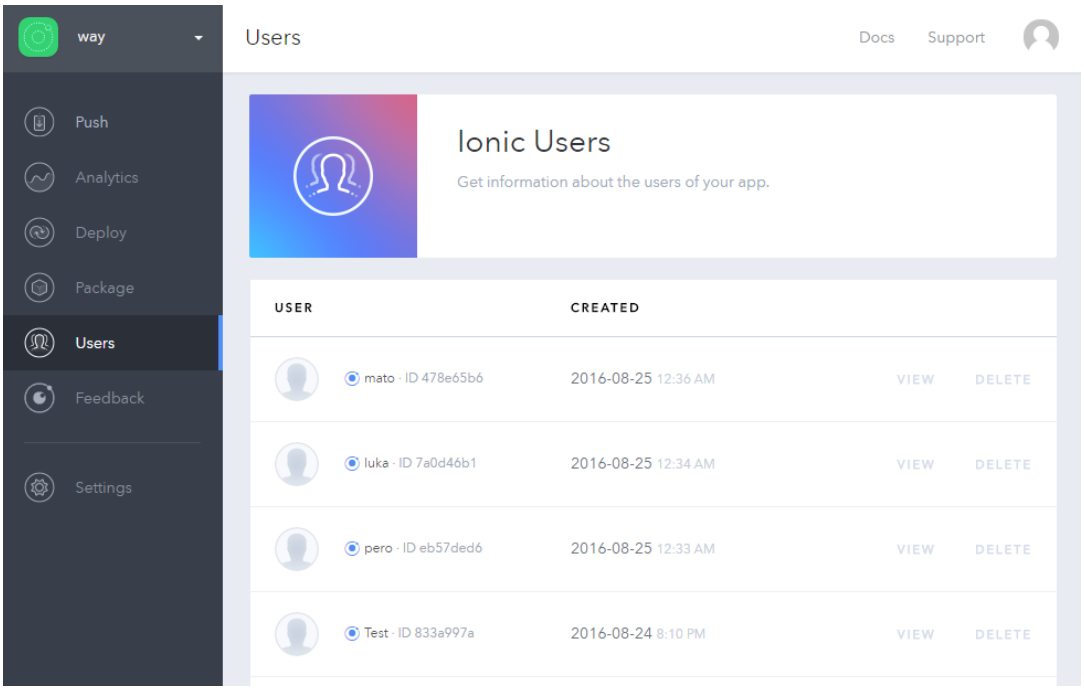
- Ionic Creator – web aplikacija koja pomoću sučelja s povlačenjem i puštanjem željenih elemenata (engl. *drag and drop*) omogućuje razvoj mobilnih aplikacija.

- Ionic Native – skup ES5/ES6/TypeScript omotača za Cordova programske dodatke, koji omogućuju dodavanje izvorne funkcionalnosti, bez direktnog korištenja Cordova programskih dodataka.

3.3.2. Upravljanje korisnicima

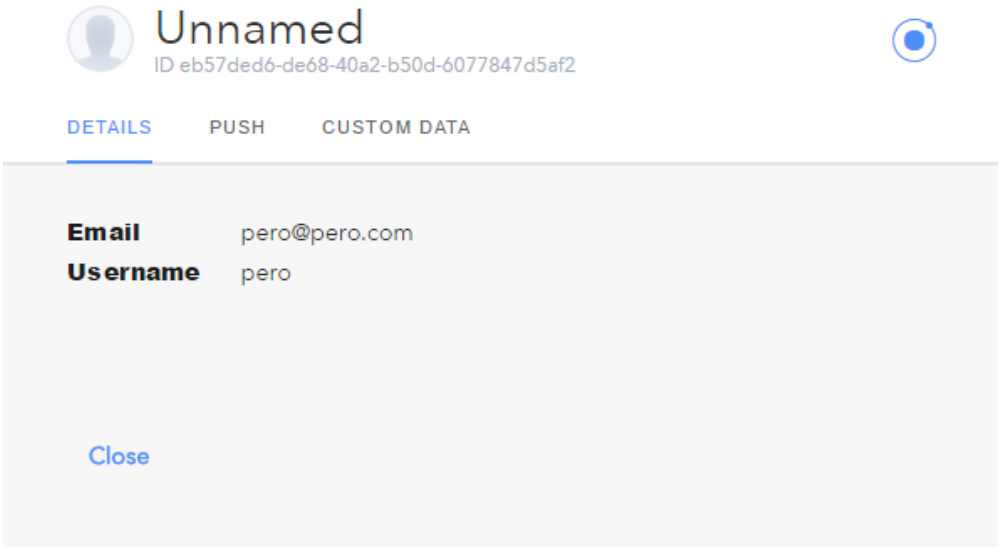
U praktičnom dijelu rada, za upravljanje korisnicima i njihovim lokacijskim i ostalim podacima, korišten je spomenuti Ionic Users servis. Osim ugrađenih metoda Ionic Users servisa, korišten je Ionic Users API za kompleksnije pristupanje i razmjenu podataka među korisnicima. Ionic Users servis korišten je kao servis koji brine o registraciji novih i prijavi starih korisnika, dodavanju novih prijatelja (drugih korisnika) na listu prijatelja te ažuriranja i dijeljenja lokacijskih podataka između korisnika aplikacije. Format zapisa podataka Ionic Users servisa u obliku je JSON objekata.

Prijavom korisnika podaci korisnika prikazuju se na web sučelju Ionic platforme. Ondje je moguć uvid u trenutne podatke svih korisnika aplikacije, kao što je i prikazano slikom 3.4..



Sl. 3.4. *Prikaz pregleda korisnika Ionic Users platforme*

Prvi dio spremljenih podataka su *details* podaci korisnika, koji osim imena i e-maila korisnika sadrže jedinstveni identifikator, *id*. *details* podaci. Osim što je njima pristup moguć i APIjem, imaju definirane ugrađene metode kojima je moguće ažurirati *details* podatke korisnika. Primjer *details* podataka s Ionic platforme pokazan je slikom 3.5..



Sl. 3.5. *Prikaz details podataka s Ionic Users platforme*

Drugi dio podataka naziva se *custom* podacima te su ti podaci u ovom slučaju ispunjeni postavljenom listom prijatelja korisnika, trenutnom lokacijom uređaja korisnika te drugim podacima. *custom* podatke određuje programer pri razvoju te je pri ažuriranju i dodavanju novih *custom* podataka obavezno korištenje APIa. Njegovo prilagođavanje ovisno je o potrebi. Primjer *custom* podataka prikazan je slikom 3.6..

Sl. 3.6. *Prikaz custom podataka s Ionic Users servisa*

Pri razvoju praktičnog dijela rada korišten je Ionic Users servis primarno zbog svoje jednostavnosti. Jedini veći nedostatak nemogućnost je direktnog ažuriranja i dodavanja *custom* podataka. Jedini način za njihovo ažuriranje pomoću je APIa i taj način nije idealan, ali trenutno, Ionic Users servis ne pruža alternativu. Pomoću APIa moguće je postaviti *custom* podatke korisnika, ali nije moguće vršiti ažuriranje liste prijatelja ili drugih podataka jer navedeni API umjesto da samo ažurira novo pridošle *custom* podatke, podatke koje postavljamo putem APIa, zamjenjuje s novo predanim *custom* podacima. Dobra alternativa Ionic Users servisu je Firebase, koji je razvijan sa strane Googlea. Firebase omogućuje više fleksibilnosti nad upravljanjem

podataka korisnika te pojednostavljuje upravljanje velikom količinom korisnika, što u pravilu vodi k aplikaciji koja troši manje resursa (mobilnih podataka i baterije).

Tijek ažuriranja *custom* podatka APIjem je takav da je prvo potrebno dohvatiti trenutne *custom* podatke korisnika. Nakon toga dohvaćene je podatke potrebno lokalno ažurirati te ažurirane podatke APIjem postaviti kao novo zadane *custom* podatke. Primjer novog postavljanja *custom* podataka uz pomoć Ionic Users APIa prikazan je na slici 3.7..

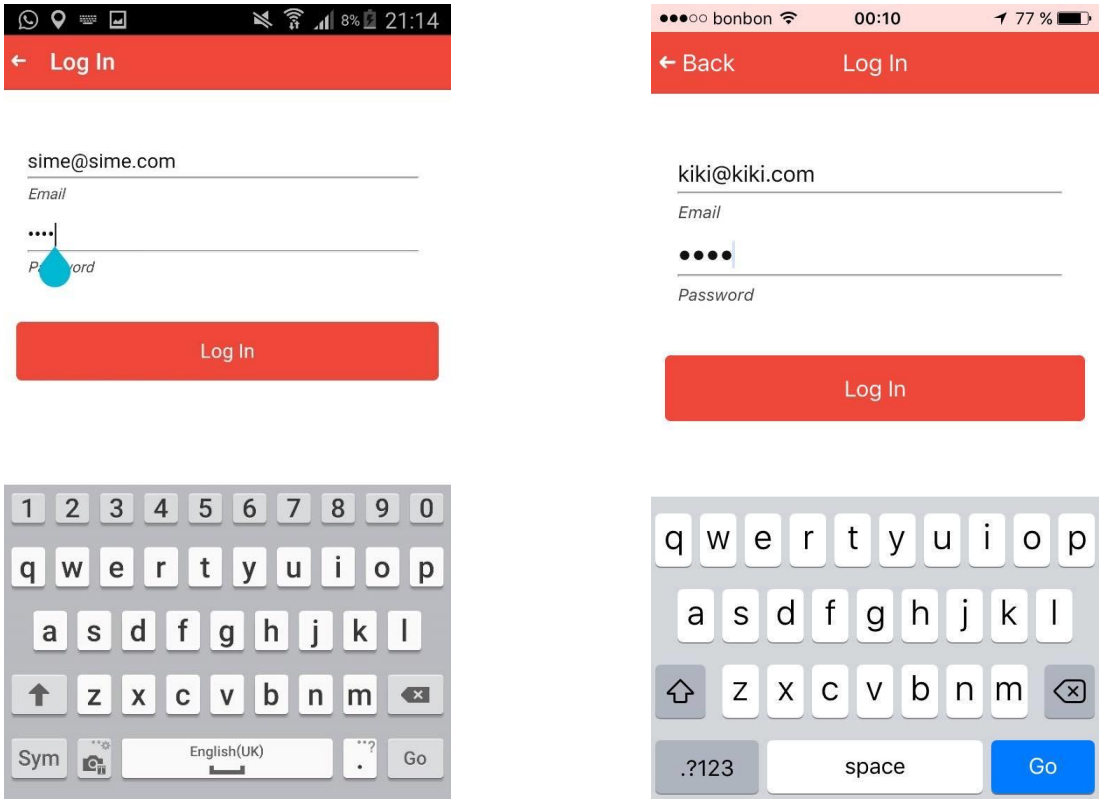
```
var token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJhOTUxMzQ4Yi1hNmNhLTQ0OTctYjllMC1jMDk0ZmY3OTUxMjUifQ.

$http({
    method: 'PUT',
    // API zahtjev
    url: "https://api.ionic.io/users/" + newFriend.uuid + "/custom",
    // Podaci koje postalvjamo kao nove custom podatke
    data: addCustom,
    // Poadci potrebni u zaglavlju
    headers: {
        'Authorization': 'Bearer ' + token
    },
})
    // Povratne funkcije
    .then(function successCallback(response) {
        console.log("Podaci ažurirani!");
    }, function errorCallback(error) {
        console.log("Nešto nije uredu!")
    });
```

Sl. 3.7. *Prikaz postavljanja custom podataka putem Ionic APIa*

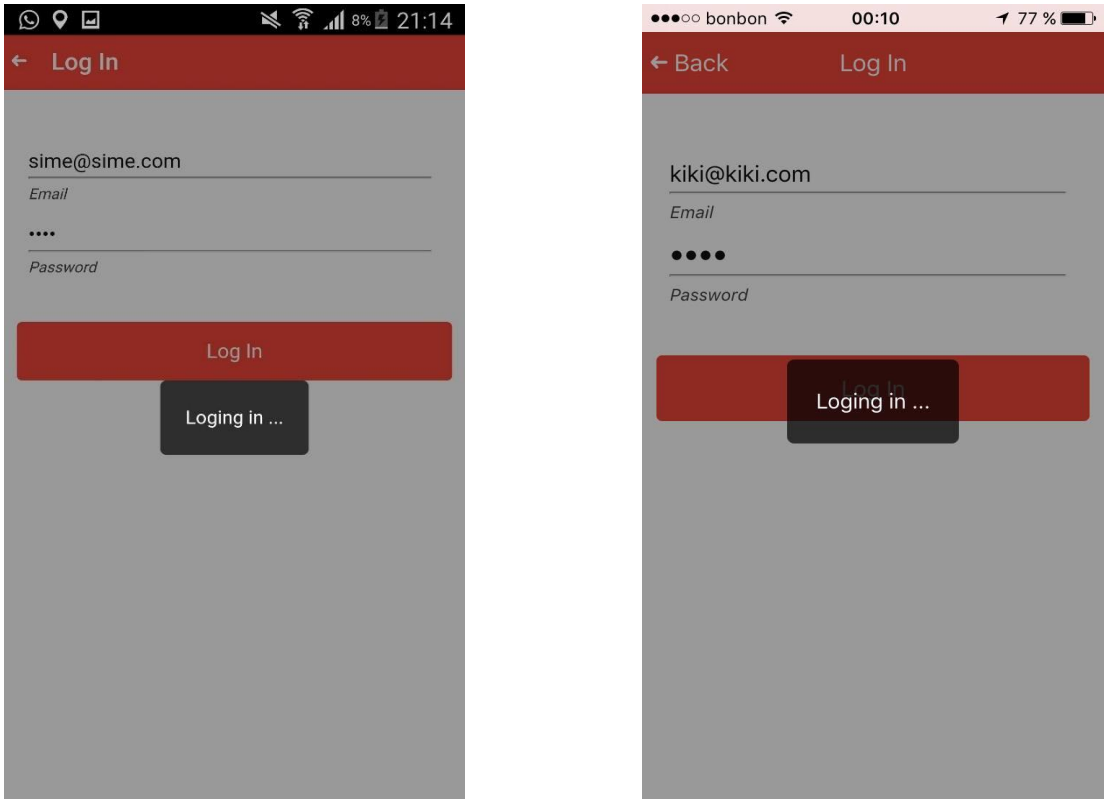
Pri pokretanju hibridne mobilne aplikacije, razvijene za praktični dio rada, korisnik prvo bira želi li se prijaviti pomoću postojećeg računa ili stvoriti novi račun. Aplikacijski kôd jednak je za sve platforme te nije dodatno modificiran. Slika 3.8. s primjerima praktičnog dijela rada pokazuje razlike između platformi i korisničkih sučelja, koje Ionic *framework* automatski dodjeljuje ovisno o platformi. Kako izgled korisničkog sučelja ovisi o zadanom pregledniku mobilnog uređaja, hibridna mobilna aplikacija, bez utjecaja programera, na različitim uređajima može izgledati drugačije. Najveće razlike se uglavnom mogu primijetiti kod različitih verzija Android OSa. Primjer prijave korisnika na Android i iOS platformi prikazan je slikom 3.8.. Na slici je vidljivo da se radi o istom korisničkom sučelju, koje je samo prilagođeno ovisno o stilskim uputama za pojedinu platformu. Hibridna mobilna aplikacija pokrenuta na Android platformi se tako prema slici 3.8. od aplikacije na iOS platformi razlikuje prema izgledu tipkovnice (izgled tipkovnice razlikuje se ovisno o pojedinoj verziji Android ili iOS verzije), izgledu unosa teksta te izgledu zaglavlja. Kao već spomenuto, zaglavlja su uređena prema preporučenim stilskim obilježjima za pojedinu platformu. Tako se kod Android platforme naslov zaglavlja nalazi na lijevoj strani pored tipke za povratak, dok se kod iOS zaglavlja naslov

zaglavlja nalazi u sredini te se tipka za nazad uz tekstualni dodatak (*Back*) također nalazi na lijevoj strani.



Sl. 3.8. *Usporedni prikaz Android (lijevo) i iOS (desno) platforme pri prijavi korisnika*

Na slici 3.9. prikazan je postupak nakon unosa korisničkih podataka od strane korisnika. Pomoću Ionic servisa *$ionicLoading* dodan je izvorni zaslon za učitavanje, koji se uvelike ne razlikuje ovisno o platformi. U ovom koraku vrši se provjera i usporedba unesenih podataka korisnika s korisničkim podacima koji se nalaze na Ionic Users platformi. Ako se uneseni podaci podudaraju s postojećim podacima na Ionic servisu, prijava će biti uspješna, u suprotnom neće doći do prijave. Za provjeru uspješnosti prijave korisnika ili registraciju korisnika korištene su Ionic ugrađene metode. Pojednostavljen prikaz korištenja ugrađenih Ionic metoda za registraciju i prijavu korisnika prikazan je na slici Sl 3.10..



Sl. 3.9. *Usporedni prikaz Android (lijevo) i iOS (desno) platforme tijekom provjere korisnika*

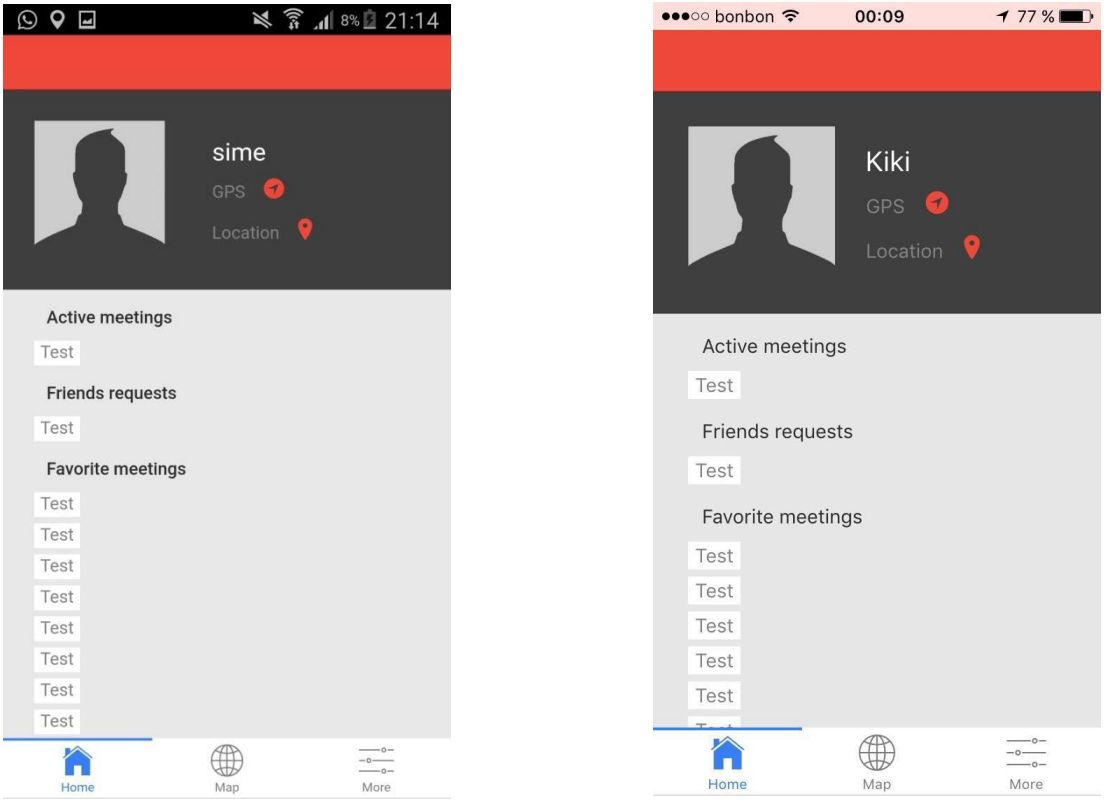
```
// Pozivanje doSignup funkcije s user objektom, koji sadrži podatke koje je
// korisnik pri registraciji ispunio.

doSignup(user)
// Poziv povratne funkcije pri uspiješnom izvođenju
    .then(function (user) {
        // Dohvacanje trenutnog korisnika
        currentUser();
        // Postavljanje dodatnih podataka pri registraciji
        setData();
        // Odlazak na profil korisnika
        $state.go('tab.home');
        // Poziv povratne funkcije pri neuspijehu izvođenja
    }, function (err) {
        console.log(er);
    }
);

doSignup = function(user) {
    // Poziv Ionic Users metode s korisničkim podacima
    Ionic.Auth.signup(user)
    .then(function () {
        // Ako dođe do uspiješne registracije, prijavi istog korisnika
        doLogin(user)
        .then(function () {
            // Prazna povratna funkcija
        }, function () {
            // Prazna povratna funkcija
        });
    }, function () {
        // Poziv povratne funkcije pri neuspijehu izvođenja
    }
)};
```

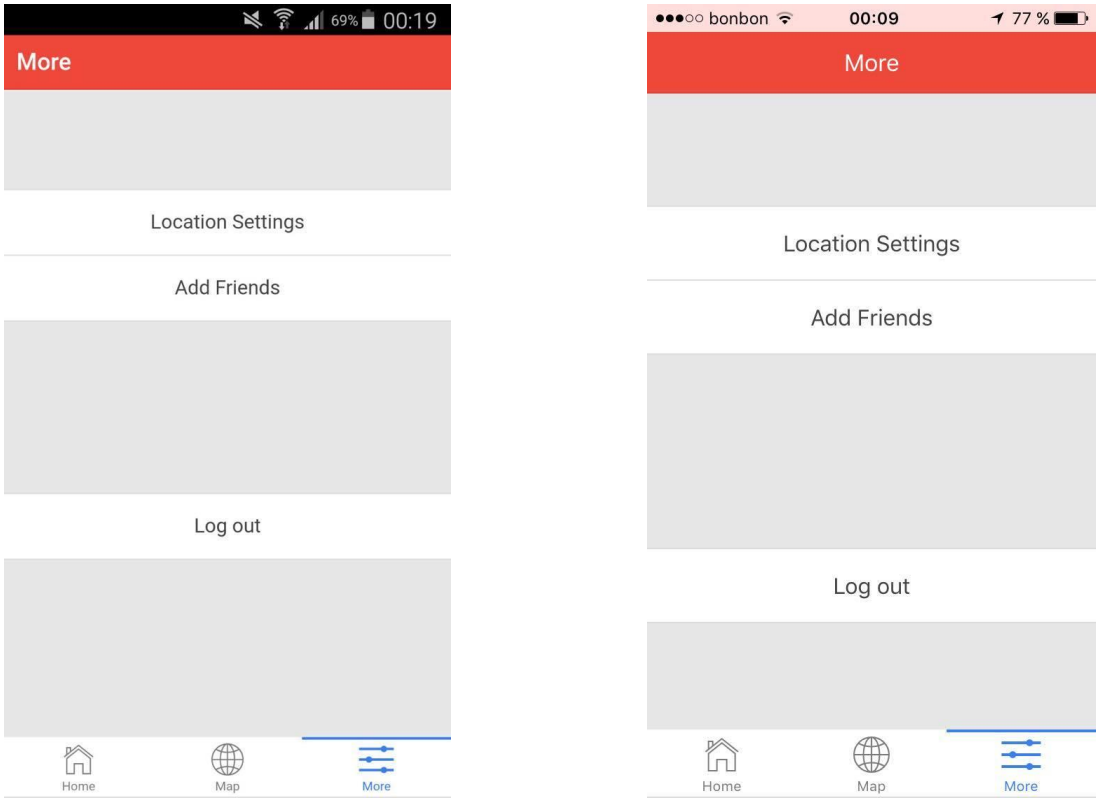
Sl. 3.10. *Prikaz korištenja Ionic Users ugrađenih metoda*

Nakon uspješne prijave, korisnik dolazi do glavnog tijela aplikacije, koje se sastoji od 3 kartice među kojima korisnik može navigirati. Korisniku se nakon prijave prvo prikazuje korisnički profil, prikazan slikom 3.11.

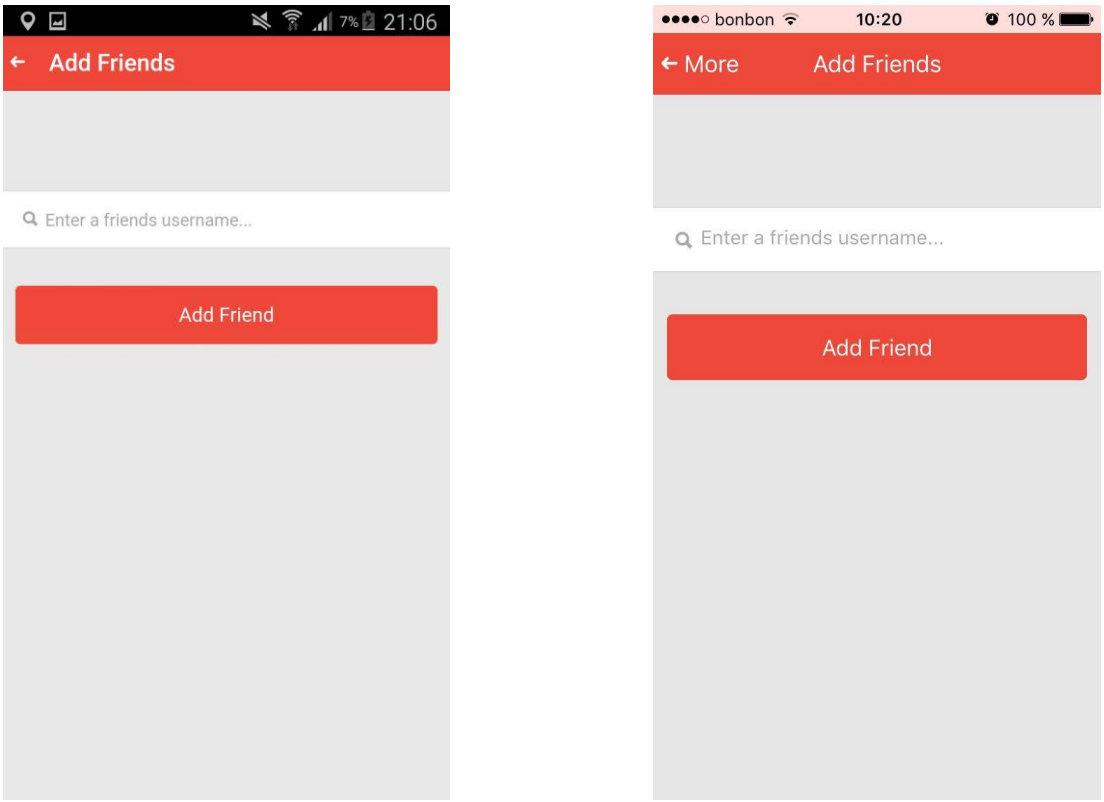


Sl. 3.11. *Prikaz korisničkog profila korisnika na Android (lijevo) i iOS (desno) platformi*

Još jedna mogućnost koju pruža aplikacija je dodavanje prijatelja, to jest drugih korisnika, koji će, pomoću poslije opisanog načina upravljanja lokacije korisnika, biti prikazani na karti korisnika koji ih je dodao. Za dodavanje prijatelja potrebno je poznavanje korisničkog imena korisničkog računa, koji se želi dodati. Kako bi korisnik mogao dodati prijatelje potrebno je navigacijskom trakom, pokazanom na slici 3.11., izabrati „*More*“ karticu, koja će korisniku pokazati dodatne mogućnosti. Na slici 3.12. pokazan je izgled navedene „*More*“ kartice. Prema slici korisnik ima tri mogućnosti: postavljanje lokacijskih postavki (ova značajka nije u funkciji), dodavanje prijatelja te odjavu s korisničkog računa. Pritiskom na gumb za odjavu se putem Ionic Users ugrađenih metoda korisnik odjavljuje te dolazi na početni zaslon, koji mu omogućuje ponovno prijavljivanje ili novu registraciju. Pritiskom na gumb za dodavanje prijatelja korisniku se prikazuje nova kartica prikazana slikom 3.13.. Kada korisnik unese korisničko ime korisničkog računa, koji želi dodati, potrebno je unos potvrditi te se gumbom u zaglavlju kartice korisnik vraća na „*More*“ karticu s glavnom navigacijskom trakom.



Sl. 3.12. *Prikaz „More“ kartice na Android (lijevo) i iOS (desno) platformi*

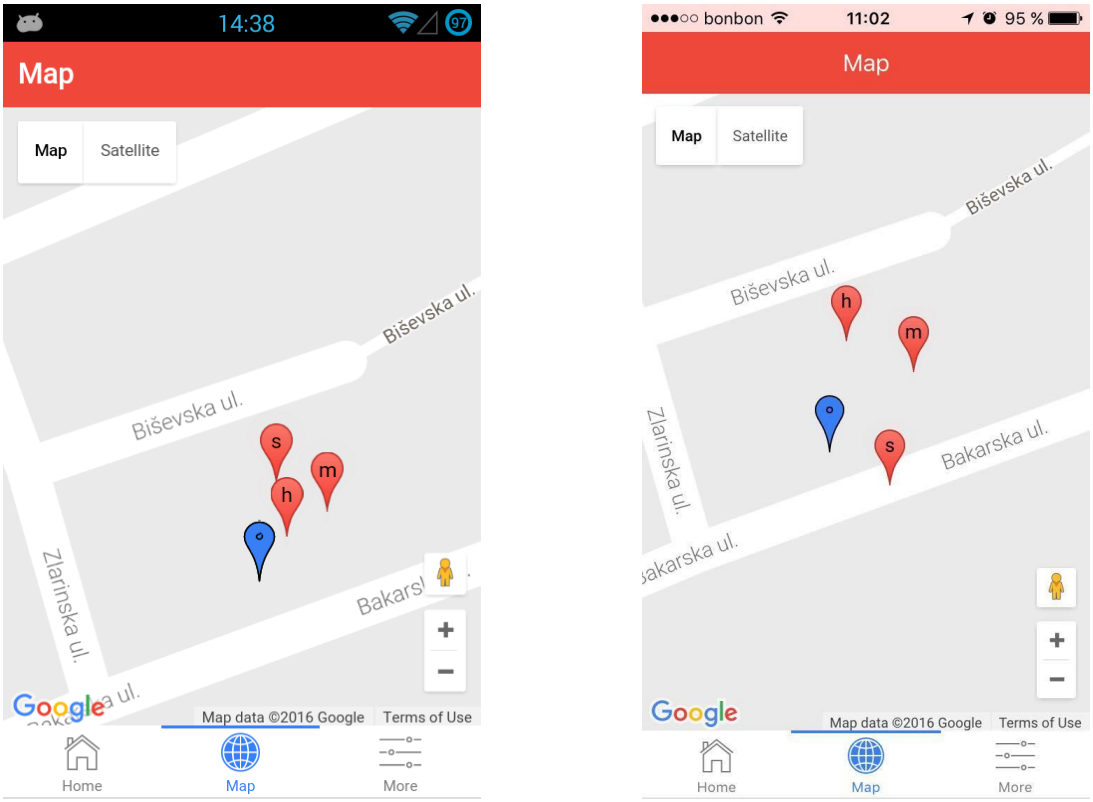


Sl. 3.13. *Prikaz kartice za dodavanje prijatelja na Android (lijevo) i iOS (desno) platformi*

Kako je i prethodno navedeno, najbitnije i najuočljivije razlike između hibridne mobilne aplikacije pokrenute na Android i iOS platformi jesu razlike u korisničkom sučelju, koje su prema slikama 3.12. i 3.13. jednake razlikama koje su prethodno već opisane.

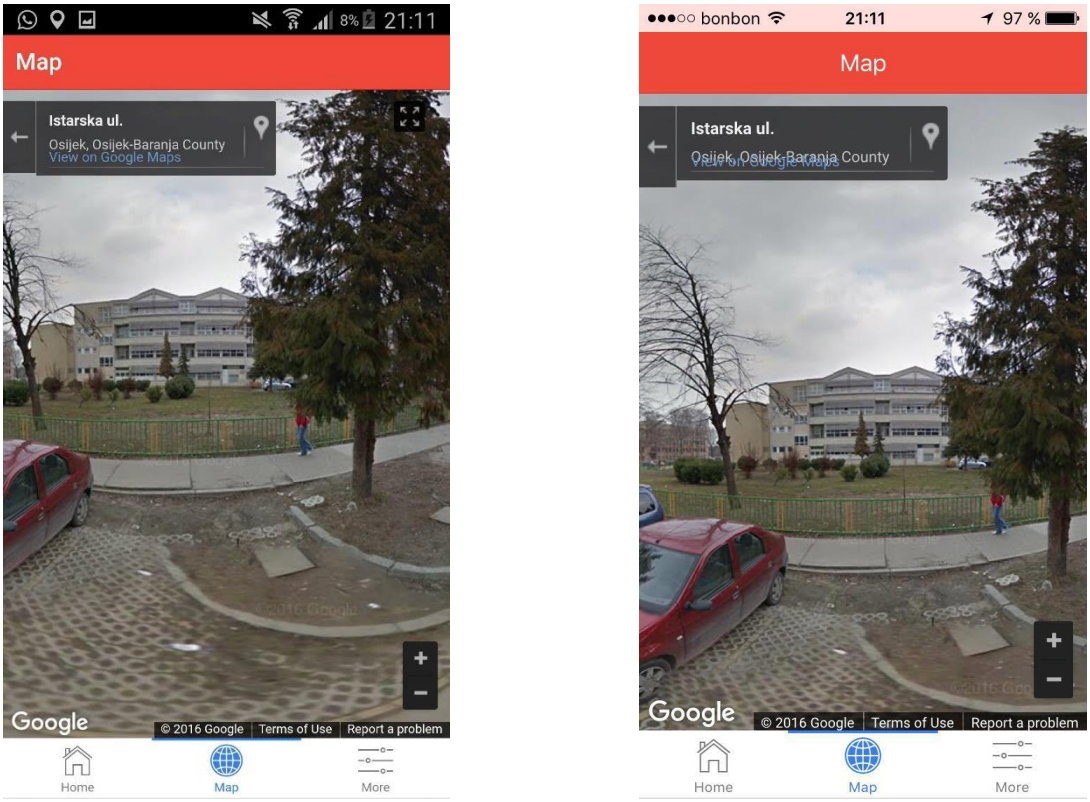
3.3.3. Upravljanje lokacijom korisnika

Upravljanje lokacijom korisnika izrađeno je pomoću Ionic Users APIa. Pri korištenju aplikacije, ona samostalno pomoću *navigatorGeolocationWatchPoosition* metode, objašnjene u odjeljku 3.2.1., ažurira lokacijske podatke samog korisnika. Tada se ažurirani podaci pomoću Ionic Users ugrađenih metoda spremaju kao novi lokacijski podaci na Ionic Users platformu. Nakon toga se isti ažurirani lokacijski podaci korisnika, pomoću Ionic Users APIa, prosljeđuju svim prijateljima na Ionic Users platformi, tako da svi korisnici koji su međusobno prijatelji imaju ažurirane lokacijske podatke. Osim *navigatorGeolocationWatchPoosition* metode, korišten je AngularJS *$interval* servis, koji omogućuje ponavljanje istog bloka kôda u određenim vremenskim razmacima. *$interval* servis korišten je kako bi aplikacija samostalno i dinamički u određenim vremenskim razmacima ažurirala markere (omogućenih sa strane Google Maps APIa) prijatelja prikazanih na karti najnovijim lokacijskim podacima. Primjer opisanog upravljanja lokacijom korisnika u hibridnoj mobilnoj aplikaciji prikazan je slikom 3.14..



Sl. 3.14. *Primjer prikaza markera korisnika i njegovih prijatelja na karti na Android (lijevo) i iOS (desno) platformi*

Razmjena ažuriranih lokacijskih podataka među korisnicima aplikacije se putem korisničkog sučelja i pomoću Google Maps API servisa prikazuju korisniku. Google Maps API servis, osim karte, omogućuje i dodjeljivanje markera s određenim geolokacijskim podacima. Pomoću lokacijskih podataka se svakom korisniku na listi prijatelja dodaje poseban marker, čija se lokacija u realnom vremenu ažurira te prijateljski korisnik, osim dinamičke promjene svog markera (označen plavom bojom) na karti, može pratiti i promjene svih drugih markera prijateljskih korisnika (označenih crvenom bojom), kako je i prikazano na slici 3.14.. Osim prikazivanja vlastite i lokacije prijateljskih korisnika, korisnik ima mogućnost korištenja Google Street View usluge. Za korištenje usluge potrebno je žuti marker (u donjem desnom kutu) povući prema željenom mjestu na karti. Primjer navedene usluge prikazan je slikom 3.15..



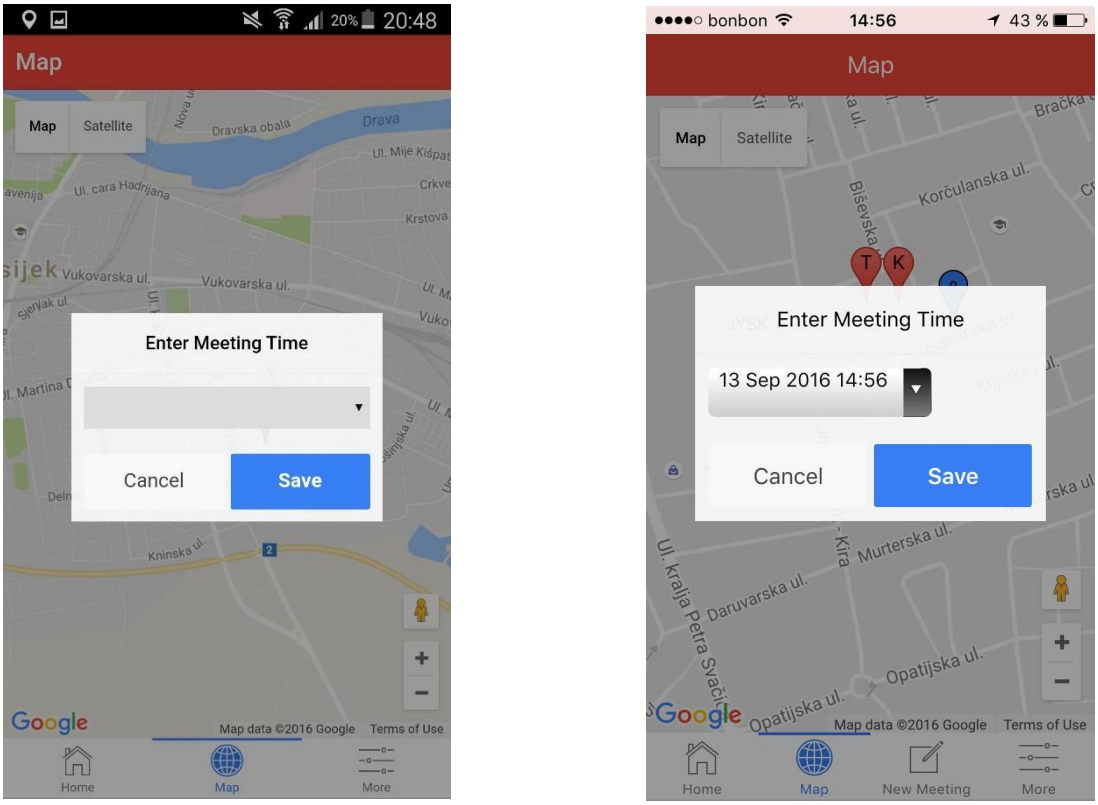
Sl. 3.15. *Prikaz korištenja Google Street View usluge na Android (lijevo) i iOS (desno) platformi*

JavaScript kao programski jezik ne omogućuje višenitno izvođenje, a hibridna aplikacija u ovom slučaju ima više metoda (*navigatorGeolocationWatchPoosition, $interval* i *$http* metode), koje se za razliku od ostalih funkcionalnosti, kao što je navigiranje kroz aplikaciju, izvode samostalno i neovisno o ostalima. Vrlo lako bi se dalo zaključiti da se one izvode na nekoj od sporednih niti, a da se glavni dio aplikacije odvija na glavnoj niti (kao što je uobičajeno ponašanje na Android platformi), što nije točno. Nasuprot višenitnom izvođenju, JavaScript

koristi spremnik za pozivanje funkcija i metoda u kojem se funkcije izvode redoslijedom kojim su pristupile u spremnik za čekanje. *navigatorGeolocationWatchPoosition* i *$interval* metode, s druge strane, jesu asinkrone metode, ali to u slučaju JavaScripta ne znači izvođenje na drugoj niti. Asinkrone metode kod JavaScripta pri pozivu iz spremnika za čekanje bivaju pozivane te kao na primjer *navigatorGeolocationWatchPoosition* metode, čekaju na odgovor udaljenog poslužitelja. U tom razdoblju čekanja na odgovor udaljenog poslužitelja, asinkrone metode omogućuju daljnje izvođenje funkcija iz spremnika za čekanje te kada asinkrona metoda dobije odgovor od udaljenog poslužitelja ponovo postaje prva sljedeća funkcija koja će se izvoditi. Na taj način ubrzava se rad cjelokupne aplikacije jer odgovori poslužitelja u nekim slučajevima mogu trajati vrlo dugo te u tom razdoblju aplikacija ne bi imala mogućnost odaziva na upute dane od strane korisnika korisničkim sučeljem.

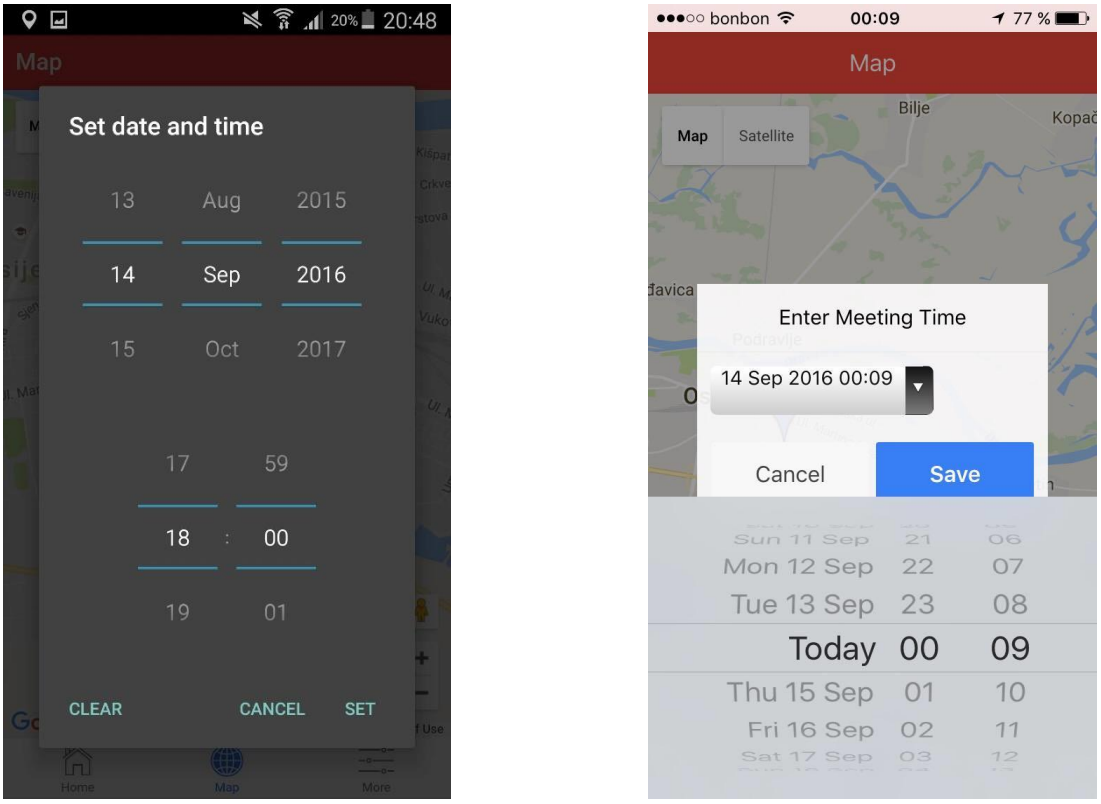
3.3.4. Upravljanje sastancima

Osim upravljanja korisničkim računima i lokacijskim podacima korisnika, razvijena hibridna mobilna aplikacija ima mogućnost upravljanja sastancima. Upravljanje sastancima, kao i ostala razmjena podataka, realizirana je pomoću Ionic Users ugrađenih metoda i APIa. Upravljanje sastancima realizirano je slično kao upravljanje lokacijskim podacima korisnika.



Sl. 3.16. *Prikaz dodavanja sastanka na Android (lijevo) i iOS (desno) platformi*

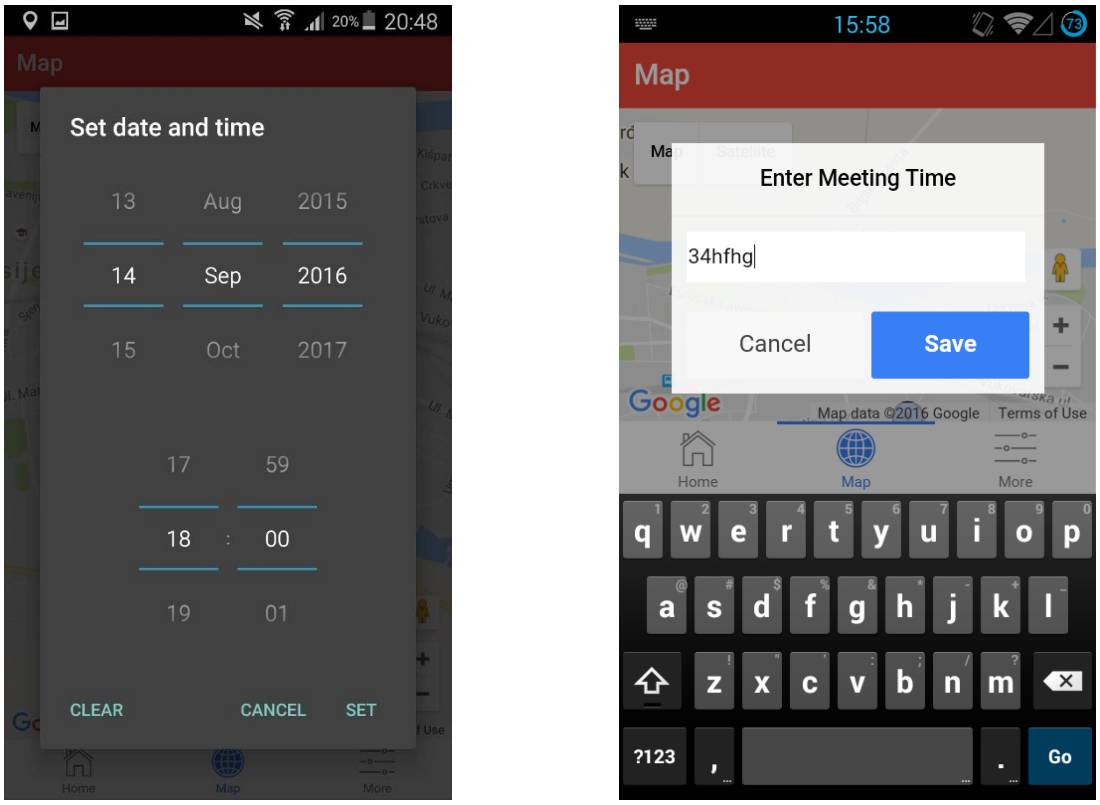
Svaki korisnik dužim pritiskom na zaslon ima mogućnost postavljanja novog sastanka te unos datuma i vremena održavanja sastanka. Primjer unosa podataka za novi sastanak prikazan je slikama 3.16. i 3.17.. Kako izgled hibridne mobilne aplikacije najviše ovisi o izvorno postavljenom pregledniku i njegovoj verziji (ako programer nije definirao drugačije), prema slici 3.17., jasno je vidljiva razlika između Android OSa, čiji je postavljeni preglednik uglavnom Google Chrome i iOS OSa, čiji je postavljeni preglednik Safari.



Sl. 3.17. *Prikaz postavljanje datuma i vremena za novi sastanak na Android (lijevo) i iOS (desno) platformi*

Osim razlika među različitim platformama, moguće su i razlike među pojedinim verzijama OSa. Značajnije promjene češće su vidljive kod različitih verzija Android OSa, koji svakom novom verzijom postavljaju nova stilska obilježja za korisničko sučelje. Prema slici 3.18. vidljive su razlike između Android uređaja s OS verzijom 5.0.1 (lijevo) i Android uređaja s OS verzijom 4.2.2 (desno). Uređaj s OS verzijom 4.2.2 (desno) uopće ne podržava način unosa datuma i vremena kao što to radi uređaj s OS verzijom 5.0.1. Umjesto toga, uređaj s verzijom OSa 4.2.2 korisniku jednostavno daje mogućnost unosa bilo kakvog teksta, što će aplikaciji poslije onemogućiti pravilni prikaz datuma i vremena održavanja sastanka. U takvim slučajevima moguće je programski dodati rješenje za navedeni OS. Navedena greška nije se

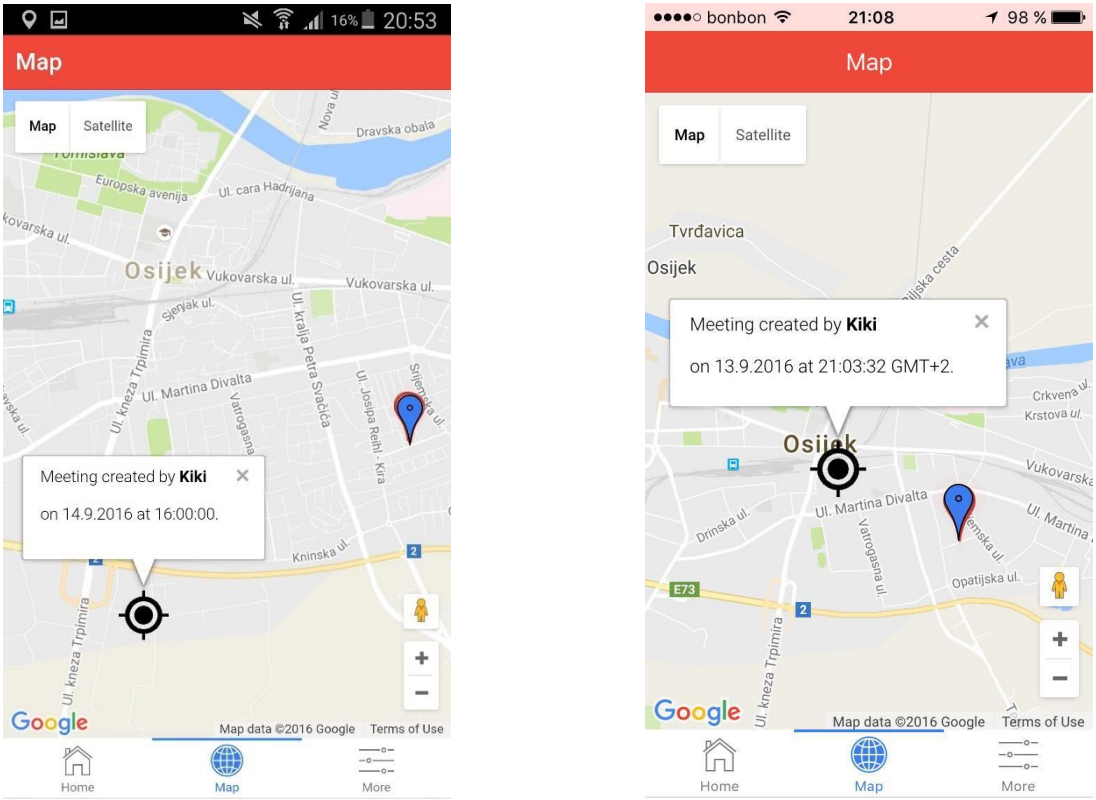
dogodila zbog Cordovae ili Ionica, nego zbog verzije preglednika uređaja, koji ne podržava unos tipa *date-time*.



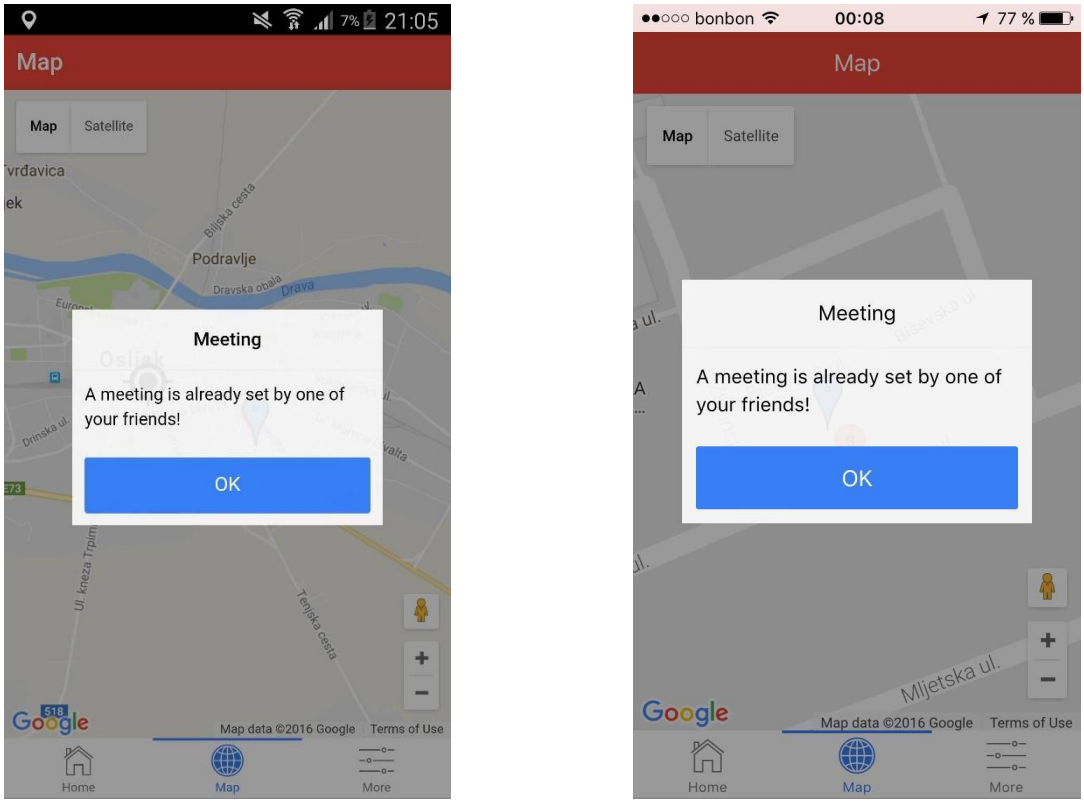
Sl. 3.18. *Prikaz unosa podataka za sastanak na dva uređaja s Android OSom, ali različitim verzijama OSa*

Kada korisnik ispuni potrebne podatke, pomoću Google Maps APIa dinamički se dodaje novi marker, koji obilježava mjesto sastanka. Dodirom na marker korisnik može saznati više podataka o sastanku. U tom se trenutku otvara prozor s informacijama o sastanku: tko ga je kreirao i kada se treba održati sastanak na lokaciji, na kojoj se nalazi marker. Podaci o novom markeru putem Ionic Users APIa proslijeđeni su svim prijateljima trenutnog korisnika. Primjer dodanog markera za sastanke i prikaz dodatnih informacija prikazan je slikom 3.19..

Ako korisnik, koji trenutno već sudjeluje u sastanku, koji je kreirao jedan od njegovih prijatelja, želi kreirati novi sastanak, to neće biti moguće. U programskom dijelu hibridne mobilne aplikacije definirano je da svaki korisnik može sudjelovati samo u jednom sastanku. Korisnik koji već sudjeluje u jednom sastanku i želi kreirati novi sastanak dobit će poruku upozoranja. Primjer poruke s upozorenjem prikazan je na slici 3.20..



Sl. 3.19. *Prikaz dodanog markera s dodanim informacijama na Android (lijevo) i iOS (desno) platformi*



Sl. 3.20. *Prikaz nemogućnosti dodavanja novog sastanka na Android (lijevo) i iOS (desno) platformi*

3.4. Izgradnja kôda za više platformi

Izgradnja kôda podrazumijeva izgradnju izvorne datoteke koja se može pokrenuti na određenoj platformi. Izgradnja kôda za testiranje u emulatorima ili privatnim uređajima automatizirana je od strane Cordovae. Pri objavljivanju mobilne hibridne aplikacije na za to predviđenim trgovinama potrebni su dodatni koraci, koji će biti detaljnije opisani u odjeljcima ovog potpoglavlja. Kako je u ovom radu pokriven razvoj hibridne mobilne aplikacije za Android i iOS platforme, detaljniji postupak izgradnje kôda u izvornu datoteku biti će objašnjen samo za navedene platforme.

Prije postupka izgradnje kôda potrebno je pripremiti i okruženje na kojem će se razvijati aplikacija te je u tu svrhu potrebno instalirati razne programske pakete. Upute za pripremanje okruženja za razvoj hibridne mobilne aplikacije neće biti opisane u ovom radu. Daljnje upute mogu se naći na službenim stranicama Cordovae.

Za izgradnju kôda u izvornu datoteku, koja se neće koristiti na službenim trgovinama aplikacijama i koja se može koristiti u privatne svrhe ili u slučaju testiranja aplikacije, potrebno je koristiti Cordova CLI naredbe u komandnoj liniji u direktoriju projekta. Sam projekt moguće je tako započeti, osim s Cordova CLI naredbama, i Ionic CLI naredbama koje su samo omotači oko izvornih Cordova CLI naredbi te su dijelom korištene u izradi mobilne hibridne aplikacije. Naredbe koje je potrebno koristiti pri razvoju hibridne mobilne aplikacije su sljedeće:

- *ionic start todo blank* – izvršenjem navedene naredbe automatski je izgrađen projekt sa osnovnom potrebnom strukturom i za početak potrebnim Cordova programskim dodacima.
- *ionic platform ad „ime platforme“* – pomoću navedene naredbe moguće je dodati platforme za koje se hibridna mobilna aplikacija želi razvijati te je, ovisno o platformi, naredbi potrebno dodati ime platforme. Pri razvoju hibridne mobilne aplikacije proizašle iz ovog rada navedenoj naredbi dodana je *ios* platforma te zatim *android* platforma.
- *ionic build „ime platforme“* – pomoću navedene naredbe dolazi do automatske izgradnje izvorne datoteke koja se može pokrenuti na za to predviđenoj platformi. Kao i u prethodnoj naredbi, potrebno je koristiti ime platforme za koju se želi izgraditi kôd u izvornu datoteku (u ovom slučaju *ios* i *android*).

Pri testiranju su vrlo korisne naredbe koje hibridnu mobilnu aplikaciju mogu direktno pokrenuti u emulatoru ili na uređaju spojenom na računalo, a to su:

- *ionic emulate „ime platforme“* – izvršenjem navedene naredbe mobilna se hibridna aplikacija automatski pokreće na emulatoru
- *ionic run „ime platforme“* – pomoću navedene naredbe mobilna se aplikacija automatski pokreće na uređaju koji je spojen na računalo.

Pri razvoju hibridne mobilne aplikacije za određenu platformu potrebno je zadovoljiti neke od uvjeta za pojedinu platformu, a to su za tri najbitnije mobilne platforme sljedeći uvjeti:

- Android – za razvoj hibridne mobilne aplikacije za Android platformu ne postoje posebni uvjeti koji bi ograničili ili onemogućili razvoj aplikacije. Svi potrebni alati i programski dodaci besplatno su dostupni svima te se izgrađene aplikacije mogu direktno testirati na svakom android uređaju. Za objavljivanje aplikacije na trgovini aplikacijama potrebno je jednokratno platiti razvojni certifikat (25$).
- Microsoft – za razvoj hibridne mobilne aplikacije za Microsoft platformu potrebno je razvoj hibridne mobilne aplikacije vršiti na računalu s Windows OS. Ostali potrebni alati i programski dodaci također su besplatni. Pri testiranju i pokretanju izgrađene aplikacije na nekom od Microsoft uređaja potrebno je posjedovati Microsoftov razvojni korisnički račun, koji je potrebno plaćati na godišnjoj bazi (oko 19$).
- iOS – za razvoj hibridne mobilne aplikacije za iOS platformu postoji više uvjeta, od kojih neki nisu čak ni dostupni svima. Tako je, na primjer, potrebno posjedovati računalo s MAC OSom, a to su u pravilu samo računala razvijana od strane Applea, kao što su Macbook Pro ili Air te su ujedno i vrlo skupa računala. Osim računala, potrebno je posjedovati Xcode alat, koji je u većini slučajeva besplatan kupnjom Apple računala. Uz navedene stavke moguće je izgraditi kôd za izvornu datoteku, ali nije moguće aplikaciju pokrenuti na uređaju koji podržava iOS platformu. Za pokretanje aplikacije na uređaju s iOS platformom i objavljivanje aplikacije na trgovini aplikacijama, slično kao i kod Microsoft platforme, potreban je razvojni certifikat. Za iOS platformu certifikat se plaća na godišnjoj bazi (99$).

3.4.1. Izgradnja kôda pomoću Cordova tehnologije

U ovom odjeljku biti će detaljnije (osnovni postupak objašnjen je na početku ovog potpoglavlja) objašnjen postupak izgradnje kôda za Android platformu pomoću Cordova tehnologije. Android platforma izabrana je kao pokazni primjer zbog vlastitog poznavanja rada s Android platformom.

Izgradnja kôda za Android platformu iz kôda hibridne mobilne aplikacije vrši se pomoću već navedene CLI naredbe *„ionic biuld android“* ili ekvivalentom *„cordova build android“*. Rezultat navedene naredbe je Android izvorna datoteka s .apk dodatkom te Android projekt s

nekim od uobičajenih izvornih datoteka s .java dodatkom, koje sadrži i svaki izvorni Android projekt. Bitno je shvatiti kako Cordova ne prevodi aplikacijski kôd napisan pomoću web tehnologija na druge programske jezike ovisno o platformi, već samo aplikaciju razvijenu pomoću web tehnologija pokreće u izvornom web pogledu određene platforme. Za Android platformu pokretanje je aplikacije razvijene web tehnologijama definiran u *MainActivity.java* datoteci. *MainActivity* razred (engl. *class*), koji se nalazi unutar *MainActivity* datoteke, glavna je aktivnost u svakoj Android aplikaciji te se izvodi prva. Pomoću *MainActivity* razreda pokreću se ostale aktivnosti koje kod izvornih Android aplikacija predstavljaju kartice koje se prikazuju korisniku. Pomoću Cordovae automatski se generira *Mainactivity.java* datoteka koja sadrži samo uputu za pokretanje web pogleda. *MainActivity* tako je i jedina aktivnost koju Cordova generira. Na slici 3.21. prikazan je primjer *MainActivity.java* datoteke koja sadrži uputu za pokretanje web pogleda i time pokreće aplikaciju razvijenu web tehnologijama. Metoda *loadUrl* pokreće izvorni web pogled i u njemu pokreće hibridni mobilnu aplikaciju razvijenu web tehnologijama.

```
package com.ionicframework.way703894;

import android.os.Bundle;
import org.apache.cordova.*;

public class MainActivity extends CordovaActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // Set by <content src="index.html" /> in config.xml
        loadUrl(launchUrl);
    }
}
```

Sl. 3.21. *Prikaz generirane MainActivity.java datoteke*

3.4.2. Potpisivanje i objava aplikacije na trgovini aplikacijama

Objavljivanje hibridne mobilne aplikacije u trgovini aplikacija ne razlikuje se od objavljivanja bilo koje izvorne aplikacije. Jedina razlika od izvornog objavljivanja aplikacije izvođenje je *„cordova build --release ime-platforme“* Cordova CLI naredbe u komandnoj liniji, koja izgrađuje izvornu datoteku posebno određenu za objavljivanje na trgovini aplikacijama. Nakon toga potrebno je koristiti uobičajeni proces objavljivanja aplikacije za određenu platformu.

4. ZAKLJUČAK

U sklopu ovog rada obrađeno je upravljanje lokacijskim podacima na višeplatformskim mobilnim aplikacijama. Kako je cilj razvoja neke mobilne aplikacije veliki broj korisnika, potrebno je pokriti što više značajnih platformi te time i veći broj korisnika. Uobičajenim načinom razvoja izvornih aplikacija potrebno je poznavanje više programskih jezika i tehnologija te time i više sredstava.

Pomoću Cordova tehnologije razvijena je hibridna mobilna aplikacija, koja dinamički upravlja lokacijskim podacima korisnika. Prednost hibridnih mobilnih aplikacija nad izvornima uporaba je istog programskog kôda, zasnovanog na uglavnom jednostavnijim web tehnologijama, kako bi se razvila aplikacija koja radi kao izvorna na više platformi.

Osim Cordova tehnologije, važan dio u izradi višeplatformske aplikacije imao je Ionic *framework* te korištena Ionic platforma, putem koje se vršilo upravljanje lokacijskim i ostalim podacima korisnika aplikacije. Pri razvoju višeplatformske aplikacije, vrlo je važno prilagoditi korisničko sučelje na način da se korisniku uvijek da osjećaj izvorne aplikacije.

Zaključno, razvoj višeplatformskih aplikacija za jednostavnije potrebe svakako može biti ravnopravna zamjena izvornim aplikacijama. S druge strane, pri kompleksnijim zahtjevima, izvorne aplikacije zbog svoje brzine izvođenja i pristupa svim izvornim mogućnostima, koje pruža mobilni uređaj, mogu u nekim situacijama biti i jedino kvalitetno rješenje.

LITERATURA

[1] Z. Epstein, Apple vs. the world, (http://bgr.com/2016/02/01/annual-smartphone-shipments-2015-apple-samsung/), pristup ostvaren 05.06.2016

[2] J. Duckett, HTML & CSS Designe and Build Websites, John Wiley & Sons, Inc., Indianapolis, 2011

[3] MDN, About JavaScript, (https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript), pristup ostvaren 01.09.2016.

[4] K. Williamson, Learning AngularJS, O'Reilly Media, Inc., Sebastopol, 2015

[5] J. Duckett, JAVASCRIPT & JQUERY interactive front-end web development, John Wiley & Sons, Inc., Indianapolis, 2014

[6] AngularJS, Developer Guide, (https://docs.angularjs.org/guide/introduction), pristup ostvaren 29.08.2016

[7] Tutorialspoint, AngularJS – MVC Arhitecture, (http://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm), pristup ostvaren 01.09.2016

[8] AngularJS, Dependency Injection, (https://docs.angularjs.org/guide/di), pristup ostvaren 02.09.2016

[9] AngularJS, Data Binding, (https://docs.angularjs.org/guide/databinding), pristup ostvaren 01.09.2016

[10] AngularJS, Services, (https://docs.angularjs.org/guide/services), pristup ostvaren 02.09.2016

[11] The Apache Software Foundation, (http://www.apache.org/foundation), pristup ostvaren 01.09.2016

[12] Apache Cordova, Introduction, (https://cordova.apache.org/docs/en/latest/guide/overview/index.html), pristup ostvaren 13.06.2016

[13] Wilken, Ionic in Action, Hybrid Mobile Apps with Ionic and AngularJS, Manning Publications Co., Shelter Island, 2016

[14] Ionic framework, Overview, (http://ionicframework.com/docs/overview/), pristup ostvaren 03.09.2016

[15] Cordova, crodova-plugin-geolocation, (https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/), pristup ostvaren 03.09.2016

[16] Physics, How does GPS work, (http://www.physics.org/article-questions.asp?id=55), pristup ostvaren 03.09.2016

[17] Google Developers, Google Maps APIs, (https://developers.google.com/maps/documentation/geolocation/intro), pristup ostvaren 03.09.2016

[18] Business News Daily, What is BaaS (Backend as a Service), (http://www.businessnewsdaily.com/4992-what-is-baas.html), pristup ostvaren 05.09.2016

[19] Ionic, Ionic Cloud Services, (http://docs.ionic.io/services/), pristup ostvaren 05.09.2016

SAŽETAK

Osnovni cilj rada objasniti je višeplatformske mobilne aplikacije, razlike među pojedinim platformama na kojima se izvode te napraviti usporedbu s izvornim mobilnim aplikacijama.

U teorijskom dijelu rada opisani su programski jezici, programski dodaci i alati potrebni za razvoj jedne hibridne mobilne aplikacije. Osim toga, objašnjeni su načini dohvaćanja lokacijskih podataka pomoću mobilnih uređaja. Detaljnije su objašnjene Cordova i Ionic tehnologije korištene u izradi praktičnog dijela ovog rada.

U praktičnom dijelu rada izrađena je višeplatformska hibridna mobilna aplikacija te je ukazano na prednosti i nedostatke takve aplikacije. Također je detaljnije pojašnjen način rada razvijene hibridne mobilne aplikacije i upravljanje lokacijskim podacima.

Ključne riječi

lokacijskih podaci, izvorni kôd, višeplatformska mobilna aplikacija, hibridne mobilne aplikacije, JavaScript, Cordova, Ionic

ABSTRACT

The main objective of this paper is to explain cross-mobile applications and the differences between platforms they run on and make a comparison with the native mobile applications.

In the theoretical part of this paper, programming languages, software accessories and tools necessary for the development of a hybrid mobile application are described. In addition, ways for retrieving location data using mobile devices are explained. The Cordova and Ionic technologies used in the practical part of this work are explained in more detail.

In the practical part of the work, a cross-platform hybrid mobile application was developed and the advantages and disadvantages of such applications were pointed out. Also, the way how the developed hybrid mobile applications works and the management of location data are explained in more detail.

Keywords

location data, GPS, native code, multi-platform mobile applications, hybrid mobile applications, JavaScript, Cordova, Ionic

ŽIVOTOPIS

Kristian Pavić rođen je 21.10.1992. godine u Celju. Nakon uspješno završene opće gimnazije „Isidor Kršnjavi“ u Našicama, upisuje se na Elektrotehnički fakultet u Osijeku. Diplomski studij procesnog računarstva upisuje 2014. godine. Student se dobro služi aktualnim web tehnologijama. Osim engleskog jezika, dobro se služi i njemačkim jezikom u govoru i pismu te ima položen B kategoriju vozačke dozvole.

X

Kristian Pavić

PRILOZI

Prilog 1. Programski kod na CDu