UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

KATEDRA ZA RAČUNARSTVO

**ANDROID MOBILNA APLIKACIJA ZA PLANIRANJE OBILAZAKA TURISTIČKIH ZNAMENITOSTI**

- ZAVRŠNI RAD -

**Zadatak:** Upoznati se sa postojećim mobilnim aplikacijama koje se koriste u turizmu za planiranje obilaska znamenitosti. Proučiti tehnologije za razvoj Android mobilnih aplikacija sa posebnim osvrtom na razvoj aplikacija sa podrškom za offline režim rada. U praktičnom delu rada implementirati Android mobilnu aplikaciju za planiranje obilazaka turističkih znamenitosti sa podrškom za offline režim rada.

**Mentor:** Doc. dr Aleksandar Stanimirović **Kandidat:** Jelena Ljubenović, 14357

**Komisija:**

1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Datum prijave: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Datum predaje: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Datum odbrane: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Niš, 2016

Sadržaj

[1. Uvod 4](#_Toc464630723)

[2. Primena mobilnih aplikacija u turizmu 5](#_Toc464630724)

[3. Pregled tehnologija za izradu Android mobilnih aplikacija koje podržavaju offline režim rada 12](#_Toc464630725)

[3.1 Meteor 13](#_Toc464630726)

[3.2 Cordova 18](#_Toc464630727)

[3.2.1 Cordovini plugin-ovi 19](#_Toc464630728)

[3.3 Leaflet 21](#_Toc464630729)

[3.4 Tehnologije za skladištenje podataka na mobilnom uređaju 23](#_Toc464630730)

[3.4.1 MongoDB 23](#_Toc464630731)

[3.4.2 MiniMongo 26](#_Toc464630732)

[3.4.3 SQLite baza podataka 26](#_Toc464630733)

[3.4.4 HTML5 Local Storage 28](#_Toc464630734)

[4. Mobilna aplikacija za planiranje obilazaka turističkih znamenitosti 29](#_Toc464630735)

[4.1 Arhitektura aplikacije 30](#_Toc464630736)

[4.2. Opis funkcionalnosti TripPlanner aplikacije 35](#_Toc464630737)

[5. Zaključak 43](#_Toc464630738)

[Dodatak A: Implementacija karakterističnih delova koda 44](#_Toc464630739)

[A.1 – Šabloni 44](#_Toc464630740)

[A.2 – FlowRouting 46](#_Toc464630741)

[A.3 – Publish/subscribe pattern 47](#_Toc464630742)

[A.4 – Postavljanje mape 48](#_Toc464630743)

[A.5 - Iscrtavanje ruta na mapi 49](#_Toc464630744)

[A.6 – SQLite plugin 50](#_Toc464630745)

[A.7 - Metode na serveru 50](#_Toc464630746)

[6. Reference 52](#_Toc464630747)

# 1. Uvod

Stalno napredovanje i usavršavanje u oblasti nauke i tehnologije nas je dovelo do činjenice da su mobilni telefoni postali neizostavni deo naše svakodnevice. Zbog svojih naprednih računarskih karakteristika, kao što su mogućnosti skladištenja izuzetno velike količine podataka, bežičnog povezivanja na internet i mnogih drugih svojstava, mobilne telefone često poistovećujemo sa džepnim računarima. Napredak svakako nije vidljiv samo u hardverskom već i u softverskom pogledu.

Danas se veliki akcenat stavlja na stalan razvoj savremenih mobilnih operativnih sistema kao što su Android i iOS, koji su iz dana u dan sve moćniji i napredniji. Android kao vodeći operativni sistem danas nije zastupljen samo na mobilnim telefonima, već se pojavljuje i kao operativni sistem televizora, raznih mašina i drugih kućnih aparata. Procenjuje se da je danas na Google Play Store-u za preuzimanje dostupno oko 2,4 miliona Android aplikacija. Sama brojka svedoči o ogromnom izboru aplikacija koji se pruža korisniku i koji pokriva gotovo sve oblasti, od igrica i aplikacija zabavnog karaktera, preko informativnih i edukativnih, do aplikacija poslovnog karaktera.

Posebno mesto zauzima i grupa aplikacija koja je namenjena turizmu i koja je podjednako zastupljena i tražena kao i sve ostale oblasti. Aplikacije iz ove kategorije su najčešće turistički vodiči koji treba da ukažu turistima na znamenitosti jednog mesta i ponude im sve relevantne informacije vezane za to mesto. Turistički vodiči takođe često sadrže komentare i ocene drugih korisnika što značajno doprinosi kvalitetu same aplikacije. Neke od aplikacija u oblasti turizma nude i mogućnost rezervisanja avionskih i hotelskih karti. Aplikacija TripPlanner koja će biti detaljno opisana u ovom radu spada u grupu aplikacija iz oblasti turizma i ima za cilj da olakša korisnicima planiranje obilazaka turističkih znamenitosti unutar gradova Srbije, sa idejom da se proširi i na ostale zemlje Evrope i sveta.

U prvom delu ovog rada biće navedene postojeće mobilne aplikacije koje omogućavaju turistima lakše snalaženje u nepoznatom gradu, a ujedno su u našoj zemlji i jedne od najpopularnijih aplikacija iz ove oblasti. Zatim će u nastavku rada biti detaljno objašnjene tehnologije koje su korišćene u implementaciji TripPlanner aplikacije. Ostatak rada sadržaće detaljne opise same aplikacije, njene funkcionalnosti, kao i informacije o arhitekturi i načinu implementacije aplikacije.

# 2. Primena mobilnih aplikacija u turizmu

Kao što smo već spomenuli sa razvojom informacionih tehnologija u 21 veku, došlo je do značajnih promena u svim oblastima pa i u turizmu gde danas mnogo lakše dolazimo do infomacija koje će doprineti kvalitnijem planiranju putovanja. Gotovo je nezamislivo započeti putovanje, a da prethodno nismo pogledali neku od aplikacija koje nam mogu pomoći u odabiru zemlje, oblasti, grada pa i hotela, i ostalih znamenitosti koje nas tamo očekuju. Zato turizam predstavlja jedno od područja za primenu najnovijih mobilnih aplikacija. Sa pojavom pametnih telefona, mobilnih uređaja i tableta svako planiranje putovanja moguće je obaviti i za vreme samog boravka u nekom gradu, bez obzira na to da li smo u zemlji ili inostranstvu. Jedno od prvih pitanja sa kojim se turisti najpe susreću u nepoznatom mestu je "šta obići". Zato će se TripPlanner aplikacija fokusirati da omogući korisnicima da nesmetano planiraju i uživaju u svom putovanju, bez obzira gde se nalaze i bez obzira na to da li imaju pristup internetu ili ne, ali fokus u ovom poglavlju biće na aplikacijama iz ove oblasti koje već postoje kao gotova rešenja.

Među njima ima aplikacija nam omogućavaju da unapred isplaniramo posetu određenom gradu, odaberemo određeni hotel, ali zajedno sa odabirom smeštaja možemo unapred planirati i određene aktivnosti i obilaske. Čak i jednostavne aktivnosti poput kupovine u nekom šoping centru, odabirom određenog restorana mogu nam olakšati putovanje. Turisti unapred mogu da budu obavešteni o lokaciji na kome se nalaze objekti, radnom vremenu objekta i podacima vezanim za javni transport. Međutim znamenitosti koje planiramo da obidjemo se često mogu nalaziti na različitim krajevima grada, pa čak i u susednim gradovima. Zato se javlja potreba da se vreme dobro organizuje, kako bi što manje vremena izgubili u putovanju između određenih lokacija i odlučivanju koje atrakcije ne treba propustiti u nekom mestu.

Jedna od bitnih karakteristika svake aplikacije za turizam jeste i prostor na kome turisti mogu da komentarišu i dele iskustva o putovanju. Feedback drugih korisnika može bitno uticati na planiranje našeg putovanja. Zbog svega navedenog, mobilne tehnologije pronalaze široku primenu u organizaciji putovanja u određenom gradu. Zapravo, turizmologija posmatra mobilne sisteme kao svoj integralni deo. Mobilne aplikacije su zamenile papire, pa umesto tradicionalnog papirnog vodiča na putovanjima koristimo mape u elektronskom formatu koje imamo u našim pametnim telefonima.

U nastavku ćemo nabrojati neke od prednosti mobilnih aplikacija u turizmu u odnosu na tradicionalne načina planiranja putovanja:

1. Sve se nalazi na jednom mestu u mobilnom uređaju. Turista može da ima jedan mobilni uređaj kod sebe i da on bude vodič koji mu pomaže u odabiru određenog mesta. Danas skoro svaki telefon ima kameru tako da može da uslika znamenitosti i da ih podeli sa ostalim turistima.
2. Moguće je integrisati vodič sa mapom. Na taj način korisnici mogu čitati o znamenitostima grada u kome se nalazi, istovremeno proveravati svoju lokaciju i odlučivati kako najbrže stići do određenog cilja.
3. Pretraga je olakšana u odnosu na papirne vodiče i umesto stalnog listanja knjige i gledanja u mapu grada da bi se pronašle informacije o nekom mestu i lokaciju gde se ona nalazi na papirnoj karti, dovoljno je samo uneti pojam i pretraživač će to sam uraditi za nas.
4. Postoji mogućnost kreiranja i izmene rute ukoliko turista to zahteva.
5. Postoji mogućnost personalizacije.

Jedna od glavnih prednosti smart mobilnih telefona za turiste je mogućnost konektovanja na internet. To možemo uraditi preko WiFi konekcije ili 3G internetom mobilnog provajdera. Međutim, s obzirom da ipak nekad nismo u mogućnosti da koristimo bilo koji vid konekcije, u TripPlanner aplikaciji akcenat je stavljen i na tome da sve funkcionalnosti bude dostupne korisniku i onda kada nema pristup internetu.

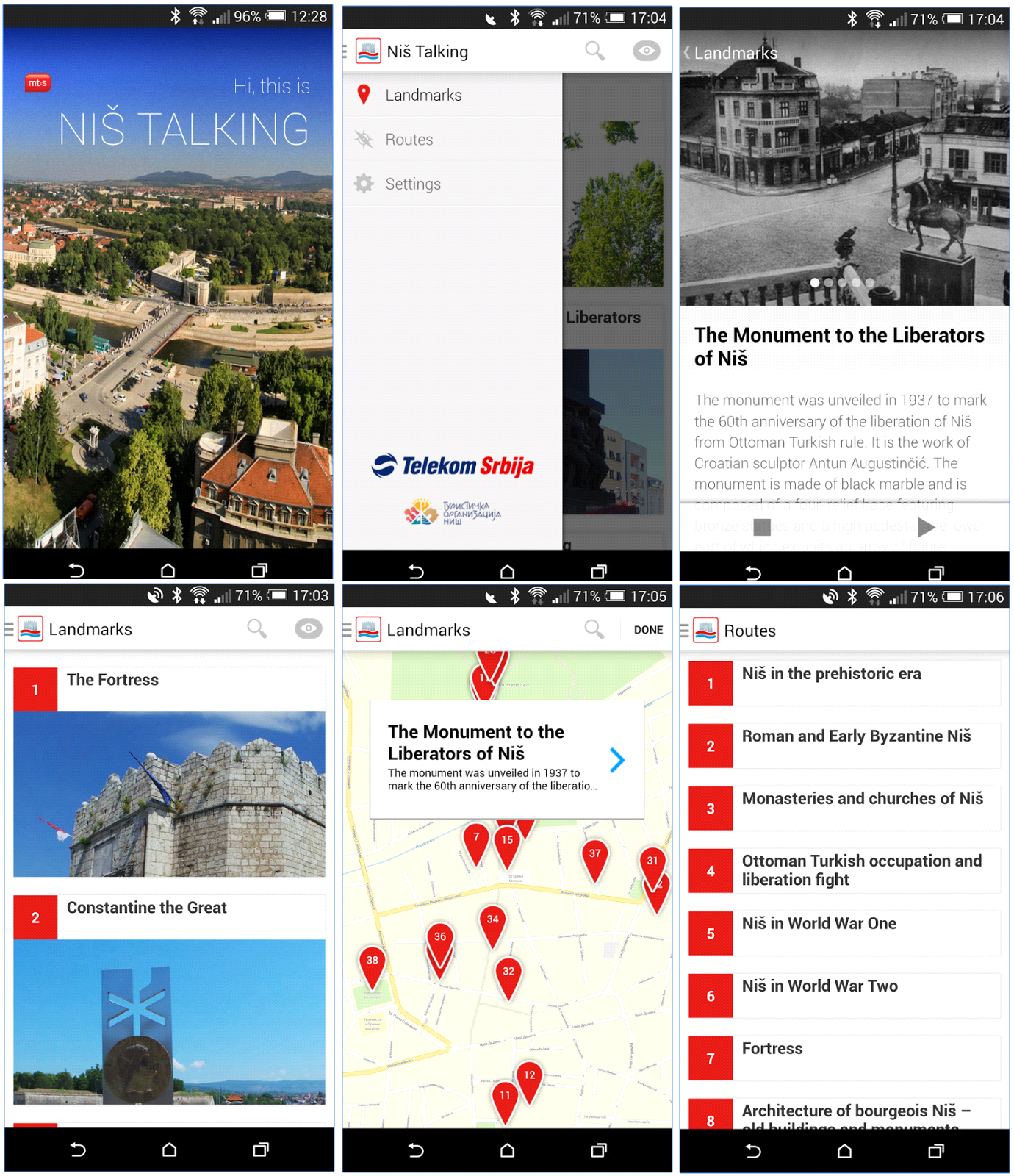
Razvijanjem mobilnih tehnologija, turisti postaju sofisticiraniji i zahtevniji. Oni zahtevaju visokokvalitetnu informaciju koja može zadovoljiti njihove potrebe. Povećana upotreba smart telefona od strane putnika donosi mnogo koristi i samim turističkim agencijama. Nije potrebno dodatno štampanje novih flajera i posete info pultova koje su važan marketinški alat u turizmu.

Na današnjem tržištu se mogu pronaći brojni primeri Android aplikacija koje se koriste u turizmu. Gotovo svaki veći grad u svetu ima svoju aplikaciju koja olakšava snalaženje i pronalazak željenih objekata. U ovom poglavlju ćemo razmotriti aplikacije iz ove oblasti koje su radjene za srpsko tržiste, a poznate su pod nazivom:

* Niš Talking
* Turistički vodič sa mapama
* Belgrade Genie - travel guide
* Belgrade City Guide

**2.1 Niš Talking**

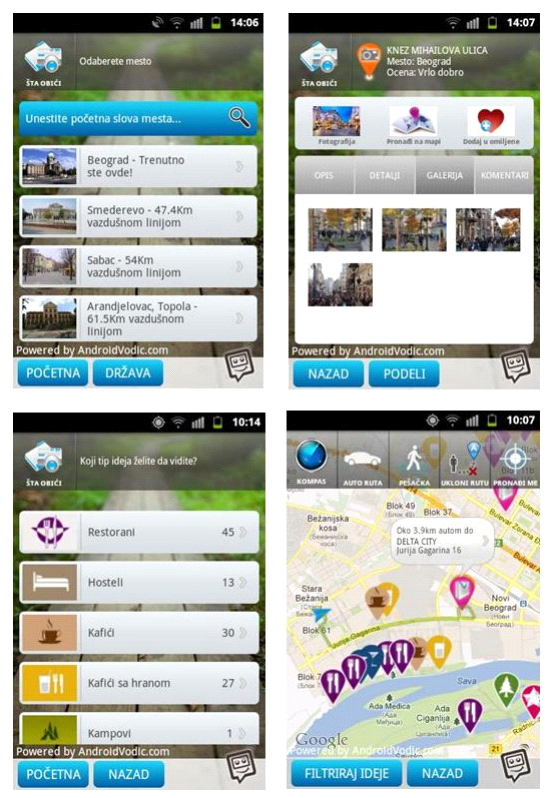
Najpoznatija i jedina aplikacija o našem gradu koju turisti mogu koristiti radi lakše snalaženja u Nišu. Ona poseduje istorijske podatke, kao i lokacije gde i kako se može stići do njih. Takođe poseduje izuzetno veliki broj zanimljivosti o našem gradu kao i neke anegdote koje se vezuju za Niš. Aplikacija ima mogućnost da odredi koliko nam se daleko nalazi određena destinacija, kao i mogućnost da se prema slici odredi o kom objektu se radi. Aplikacija ima interaktivni kompas tako da nas navođenjem olakšava dolazak do željenog cilja. Poseduje i audio-video vodiča koji može da nas upozna sa istorijom određenog objekta. Niš Talking nudi mogućnost posete Niša u već određenim pešačkim rutama da bi turista mogao da se upozna sa spomenicima, ali i drugim kulturnim vrednostima našeg grada. Aplikaciju je moguće preuzeti sa Google Play Store-a nakon čega će ona biti sačuvana na lokalnom uređaju i prilikom narednog pokretanja neće zahtevati internet konekciju. [1.]



Slika 1 - Niš Talking

**2.2 Turistički vodič sa mapama**

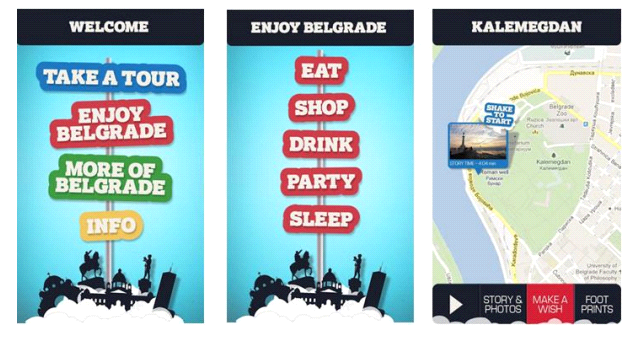
Ovaj turistički vodič nudi veliki broj ideja za odmor i putovanje u nekoliko država Balkana sa interaktivnom mapom uz foto galeriju i čitanje komentara putika. Turistički vodič podržava 50-ak gradova i ideje su podeljene u veliki broj kategorija (etno sela, salaši, bazeni, plaže, apartmani, hoteli, muzeji, restorani, reke, vodopadi itd.) Poseduje i napredne opcije za rezervisanje godišnjih odmora kao i turističke pogodnisti (first i last minute ponude).



Slika 2 - Turistički vodič sa mapama

**2.3 Belgrade Genie - travel guide**

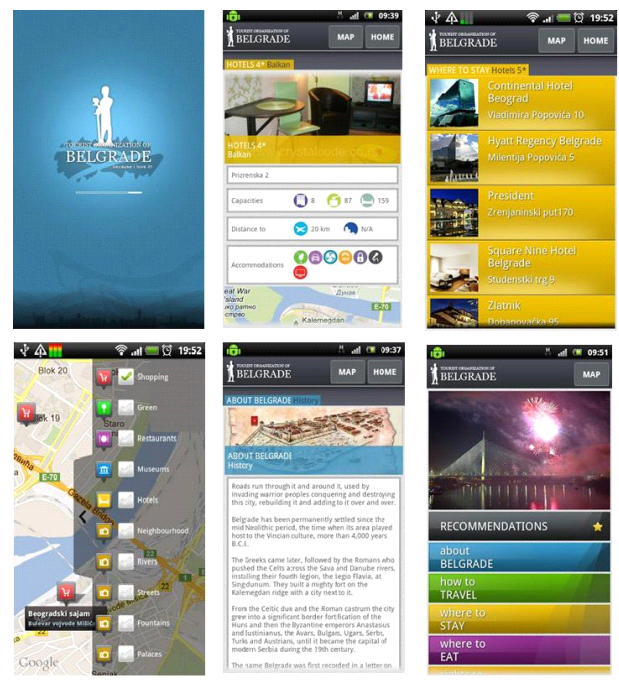
Belgrade Genie je najpopularniji interaktivni vodič za grad Beograd. Dizajnirana je da radi offline, bez korišćenja interneta što je glavna prednost. Koristi GPS radi određivanja lokacije. Ova aplikacija ima audio vodič, tj. audio ture koje nam govore o određenim destinacijama u Beogradu. Pozicioniranje se vrši preko GPS-a na offline mapu, na kojoj možemo pronaći muzeje, pozorišta, restorane, a poseduje i markere tako da možemo da dobijemo informaciju i da li određeni objekat trenutno radi. Postoji mogućnost objavljivanja komentara kao i određivanja pozicije besplatnih WiFi hotpotova. U nastavku sledi par slika iz Belgrade Genie aplikacije:



Slika 3 - Belgrade Genie

**2.4 Belgrade City Guide**

Ovo je zvanična aplikacija Turističke organizacije Beograda. Sadrži listu preko 650 objekata u Beogradu i veliku biblioteku slika lokaliteta. Takođe možete pročitati komentare o lokacijama kao i najkvalitetnija mesta za izlazak, hranu i smeštaj.



Slika 4 - Belgrade City Guide

Kod svih prethodno navedenih rešenja možemo da uočimo da je bitno postojanje mape sa iscrtanim znamenitostima, kako bi se korisnici lakše snalazili prilikom korišćenja. Takođe su kao dodatne opcije napravljene i liste objekata iz kojih se može doći do više detalja o svakom objektu, kao i mogućnost komentarisanja istih, ali kao najbitniju karakteristiku navešćemo mogućnost rada aplikacija u offline režimu. Ovo omogućava aplikaciji da najnormalnije funkcioniše i u slučaju nedostupnosti interneta, što svakako može biti korisno prilikom obilaska inostranih gradova ili usred loše konekcije. Ideja sa TripPlanner aplikacijom je da korisnici pored svih navedenih funkcionalnosti, imaju i mogućnost kreiranja i dodavanja novih objekata na mapu.

Nakon pregleda ovih postojećih rešenja mobilnih aplikacija u turizmu u ostatku ovog rada fokus će biti na TripPlanner aplikaciji. U narednom poglavlju upoznaćemo se detaljno sa korišćenim tehnologijama i objasniti zašto su bas one odabrane, a ne neke druge.

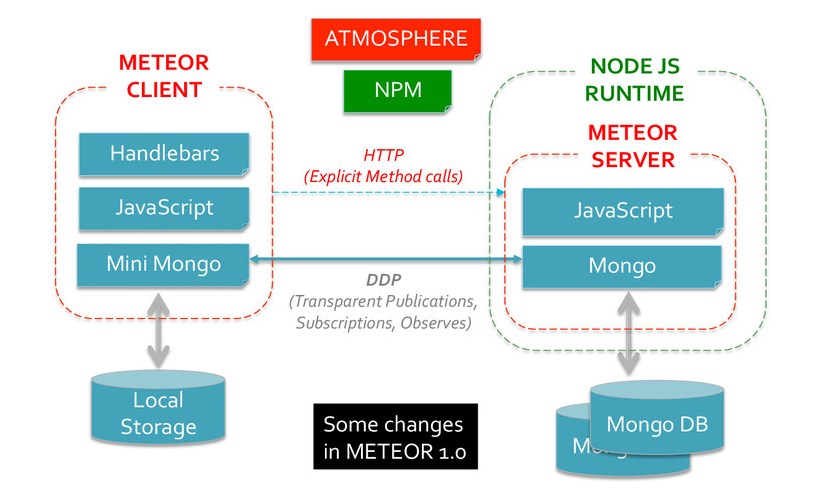
# 3. Pregled tehnologija za izradu Android mobilnih aplikacija koje podržavaju offline režim rada

Da bismo realizovali ideju o pravljenju turističkog vodiča koji može da radi u offline režimu, bilo je najpre neophodno odabrati odgovarajuće tehnologije. S obzirom na veliki izbor modernih frameworka koji danas postoje i nude najrazličitije funkcionalnosti, u izradi TripPlanner aplikacije izabran je Meteor framework, kao framework koji nam može pomoći u brzoj izradi mobilne aplikacije sačinjene od serverskog i klijentskog dela između kojih postoji jaka reaktivna veza, kako bi sve promene bile odmah vidljive svima. S obzirom na tematiku aplikacije, koja zahteva rad sa mapama, ali da on bude moguć i u offline režimu, korišćena je Leaflet biblioteka, ali je posebna pažnja posvećena tehnologijama za skladištenje podataka kako na serverskom, tako i u klijentskom delu aplikacije. Na serverskom delu izabrana je Mongo baza podataka, zbog svoje brzine, jednostavnog načina skladištenja podataka kao i mogućnosti perzistiranja podataka. Na klijentskoj strani je korišćena Minimongo baza podataka, medjutim posto je ona in-memory, a nama je za realizaciju offline turističkog vodiča neophodno i perzistiranje podataka, kao dodatak njoj korišćen je SQLite plugin. Svoju primenu u izradi ove aplikacije našao je i HTML5 Local Storage, a detalji za svaku od navedenih tehnlogija biće izloženi u nastavku.

## 3.1 Meteor

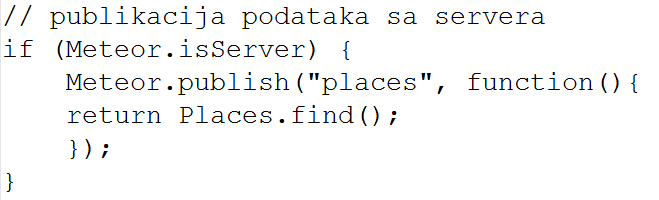
Meteor je open source platforma za kreiranje web i mobilnih aplikacija u JavaScriptu-u i predstavlja primarnu tehnologiju korišćenu u izradi TriPlanner aplikacije. Suština ovog frameworka-a leži upravo u njegovim osnovnim principima. Svedoci smo da je danas prilikom izrade aplikacija jako bitno da ona bude izuzetno reaktivna i to u realnom vremenu. Ova karakteristika se očekuje i od TripPlanner aplikacije, pa je to razlog zašto je upravo izabran Meteor, a ne neki drugi framework. Da bi se postigla reaktivnost, prilikom izrade aplikacije, ovu funkcionalnost nam ne mogu obezbediti niti HTTP protokol, niti baze podataka, jer su i jedni i drugi previše spori. Takođe još jedan od zahteva TripPlanner aplikacije je da kada dodje do promene podataka u bazi, promena bude automatski vidljiva svim korisnicima, ali bez osvežavanja strane ili drugih nepoželjnih intervencija. Meteor predstavlja jedinstveno rešenje za sve ove probleme. [2.]

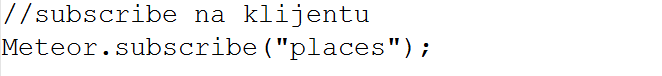
Meteor je zapravo sačinjen od paketa koje možemo nesmetano i nezavisno da koristimo po potrebi, ali i da ukoliko nam neki od njih nije potreban, postoji mogućnost da ga isključimo. Tipična Meteor aplikacija se sastoji iz 3 dela: servera, klijenta i komunikacionog kanala. Ovu arhitekturu uz manje promene i dodatke zadržala je i TripPlanner aplikacija, koja je tema ovog rada. Kada se klijent poveže na server između njih se uspostavi jedinstveni komunikacioni kanal – poznat kao soket, a zatim se čitava razmena podataka obavlja upravo preko formiranog soketa. U nastavku sledi detaljan opis arhitekture prosečne Meteor aplikacije.



Slika 5 – Arhitektura Meteor aplikacije

Serverska komponenta aplikacije sadrži JavaScript kod koji omogućava serveru da uspešno komunicira sa klijentom i Mongo bazom podataka. Server se izvršava na Node.js-u. Na serveru se nalazi kod koji omogućava publikaciju podataka ka klijentu. Ova publikacija potiče od publish/subscribe pattern-a. Ukoliko se on ne bi koristio, klijentu bi se slale čitave kolekcije podataka, što smo smo svakako hteli da izbegnemo u TripPlanner aplikaciji. Zato je uz pomoć publish/subscribe pattern-a klijentu omogućeno da se pretplati samo na neophodne podatke koje server publikuje. Ono što ovde treba istaći jeste da ukoliko dodje do bilo kakve promene podataka u MongoDB bazi, ova promena će automatski biti prosledjena svim subscribe-ovanim klijentima. Sledi primer publish/subscribe mehanizma:





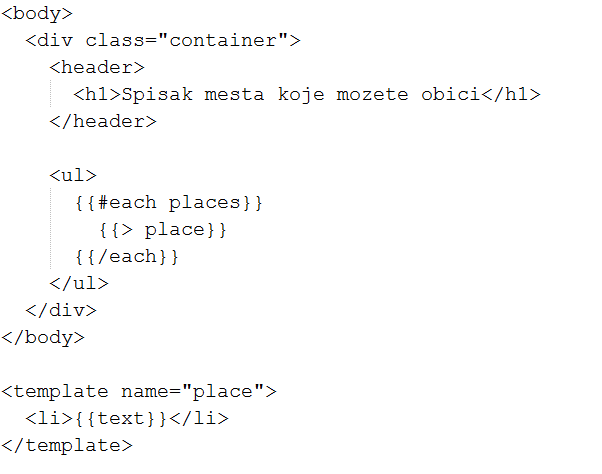
Slika 6 – Publish / subscribe mehanizam

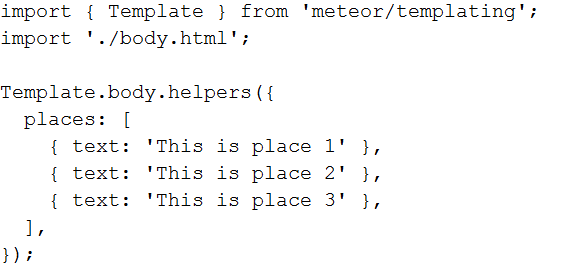
Na ovaj način klijentu se sa servera ne prosleđuju sve kolekcije, već samo one na koje se subscribe-ovao, u ovom slučaju je to kolekcija Places. Glavna baza podataka u kojoj se skladište svi podaci u aplikaciji je MongoDB. Ona spada u NoSQL baze podataka i svaki zapis u njoj prestavlja dokument, dok skup više dokumenata predstavlja kolekciju koja se može poistovetiti sa tabelom iz relacionih baza podataka. MongoDB je in-memory baza podataka u kojoj se podaci skladište u JSON formatu i kao takva spada u grupu izuzetno brzih baza podataka. Ova baza podataka biće detaljno opisana u nastavku ovog rada. [3.]

Druga bitna komponenta Meteor aplikacije je komunikacioni kanal preko koga se odvija transfer podataka sa klijenta na server i obrnuto. Ovaj soket je zadužen za prenos podataka u JSON formatu izmedju klijenta i servera. Za uspostavljanje konekcije se u pozadini koristi socket.io biblioteka. Suština komunikacije izmedju klijenta i servera se obavlja preko soketa, jer soketi obavljaju transfer podataka mnogo brže nego što bi to činio HTTP protokol i samim tim je mnogo pogodniji u izradi reaktivnih aplikacija, kao što je TripPlanner aplikacija.

Treći neizostavni deo svake Meteor aplikacije, pa i TripPlanner aplikacije je klijentska komponenta. Klijent se subscribuje na sve one podatke koji su mu potrebni. Za podatke koje dobija od servera klijent kreira keš memoriju poznatu kao Minimongo, koja je ništa drugo do jednostavna baza podataka na klijentu. Kada pristupamo podacima na klijentu, upit koji se pravi je identičan sa onim koji bismo pozvali na serveru, što predstavlja još jedan od razloga zašto je izabrana baš ova baza i baš ovaj framework. Ovde je jako bitno istaći da Meteor koristi mehanizam kojim su podaci iz baze na klijentu i serveru u svakom trenutku sinhronizovani. Ukoliko se desi promena na klijentu, ta promena je odmah vidljiva i na serveru, a server dalje obaveštava ostale klijente da se desila promena. Minimongo je in-memory baza, pa ukoliko nam je bitna perzistencija podataka na klijentu, onda treba koristiti neki pomoćni mehanizam skladištenja podataka. U našoj aplikaciji je za takve potrebe korišćena SQLite baza podataka, kako bi nam svi neophodni podaci bili dostupni i u offline režimu i po ponovnom uključivanju aplikacije.

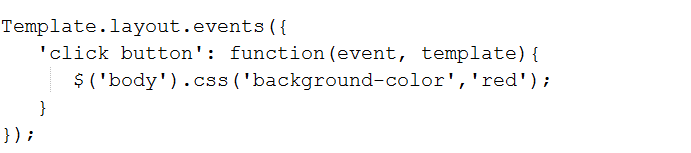
Važan koncept Meteor-a i TripPlanner aplikacije su i templejti. Templejti nam omogućavaju da definišemo blokove HTML koda, koji potom mogu biti iznova korišćeni koliko god je nama to potrebno. Kao rezultat korišćenja templejta, Meteor aplikacije imaju manje linija koda, a sam kod je jednostavniji za čitanje, a pritom se i lakše menja i održava. Ovi blokovi HTML koda u sebi mogu da sadrže i prostor koji se može popuniti podacima koje nam vrati aplikaciona logika. Prilikom rada sa templejtima, često se javlja potreba da se ista funkcionalnost primeni više puta. U tom slučaju koristimo helpere. Helperi su JavaScript funkcije koje obavljaju bilo koji vid procesiranja podataka, na primer pribavljanje podataka iz baze. Helperi mogu da se odnose na jedan templejt, ali mogu biti i globalno dostupni. Globalni templejti mogu biti ponovo korišćeni u svim dostupnim templejtima. Sledi primer korišćenja templejta i helpera.





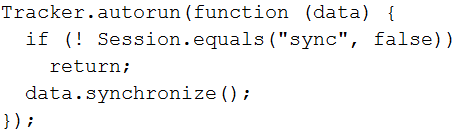
Slika 7 – Korišćenje templeta

U primeru smo definisali helper: places koji nam vraca niz place-ova, i za svaki od njih zatim pozivamo odgovarajući template place da ga iscrta. Pošto nam je jedan od ciljeva u TripPlanner aplikaciji bio i da se korisniku omogući da interaguje sa aplikacijom, zato je bilo potrebno obrađivati i dogadjaje kao što su klik na dugme, dupli klik, gubljenje fokusa i slično. Neophodno je definisati događaje koje će aplikacija da osluškuje, kao i akcije koje će da se trigeruju kada se desi dogadjaj. Sledi primer koda (Slika 8) koji registruje klik na dugme, a kao posledica toga doći će do promene boje pozadine:



Slika 8 – Obrada dogadjaja

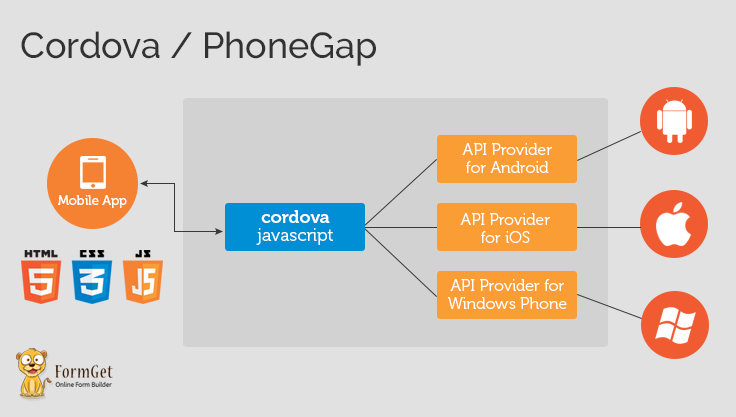
Svakako glavna prednost korišćenja Meteor frameworka u razvoju TripPlanner aplikacije i generalno je postojanje mehanizma kojim se postiže reaktivnost aplikacije. Reaktivnost se postiže tako što se uspostavlja zavisnost između reaktivnog izvora i reaktivnog izračunavanja. U reaktivne izvore spadaju kolekcije odnosno kursori, sesije i reaktivne promenjive koje sami definišemo. S druge strane, reaktivna izračunavanja mogu biti helperi, kao i Tracker.autorun( ) funkcija. Ideja je da se unutar reaktivnog izračunavanja, postavi reaktivni izvor. Ukoliko se desi da se bilo gde u kodu promeni vrednost reaktivnog izvora, recimo neke sesije, to će automatski prouzrokovati da se reaktivno izračunavanje ponovo obavi, zbog postojanja zavisnosti između reaktivnog izvora i izračunavanja.



Slika 9 – Primer reaktivnosti u aplikaciji

## 3.2 Cordova

Cordova je open source platforma koja nam omogućava izradu hibridnih moblinih aplikacija korišćenjem standradnih tehnologija za izradu web aplikacija, kao što su HTML, CSS i JavaScript. Izvršava se u web view-u koji je ugrađen u nativnu aplikaciju, umesto standardnog web browsera. Web view je u osnovi browser, ali bez korisničkog interfejsa klasičnog browser-a. U zavisnosti od toga u kom web view-u se izvršava aplikacija može imati veliki uticaj na performanse same aplikacije. Važna prednost Cordova aplikacija je činjenica da sav repozitorijum (eng. Assets) aplikacije dolazi u paketu sa aplikacijom. Ovim se doprinosi da se aplikacija brže učitava u odnosu na neku web aplikaciju koja se izvršava na udaljenom serveru što stvara dodatne probleme u slučaju slabe internet konekcije. [4.]



Slika 10 – Cordova

Još jedan bitan koncept Cordovinih aplikacija je postojanje ***hot code push*** mehanizma, koji omogućava update-ovanje aplikacije bez potrebe da se prate uobičajeni koraci. Ovaj mehanizam funkcioniše tako što postoji Meteor alat koji detektuje relevantne promene nastale u kodu aplikacije, zatim ih ponovo kompajlira i obaveštava sve povezane klijente da je nova verzija aplikacije dostupna. Na taj način klijenti mogu da automatski da ponovo učitaju aplikaciju i time postižu da kod sebe uvek imaju najnoviju verziju. Hot code push mehanizam postoji i kod browser-a, medjutim način na koji on funkcioniše kod mobilnog klijenta je malo drugačiji zbog činjenice da se u tom slučaju čitav repozitorijum (eng. Assets) aplikacije čuva lokalno. Ovaj mehanizam sastoji se iz dva dela:

1. Update-ovani repozitorijum se preuzima se servera korišćenjem nativnog mehanizma za preuzimanje, a potom se skladišti na uredjaju.
2. Strana se opet učitava, a web view sav materijal zahteva od lokalnog web servera.

Bitna činjenica ovakvog načina funkcionisanja je to što preuzimanje preko mobilne konekcije može biti jako sporo, pa se ono obavlja u pozadini. Dokle god se preuzimanje u potpunosti ne završi aplikacija neće pokušati da učita novi sadrzaj. Preuzimanje sadrzaja (engl assets) se obavlja inkrementalno, što znači da će se preuzeti samo onaj sadržaj koji je promenjen. U slučaju problema da konekcija nestane i preuzimanje se prekine, kada problem bude otklonjen aplikacija nastavlja sa preuzimanjem tamo gde je stala.

Još jedna od prednosti Cordove i razlog zašto je baš ona izabrana za izradu TripPlanner aplikacije je njena integrisanost sa Meteorom. Sa integracijom Cordove i Meteora, moguće je postojeću aplikaciju pokrenuti i na iOS i na Android uredjaju i sve to uz pomoć nekoliko linija koda.

### 3.2.1 Cordovini plugin-ovi

Kako bismo proširili funkcionalnosti TripPlanner aplikacije korišćeni su Cordovini pluginovi. Cordova nudi mnošto plugin-ova uz pomoć kojih možemo pristupiti nativnim karakteristikama uredjaja, što nije moguće u slučaju klasičnih web aplikacija. Plugin-ovi su dodaci koje sadrže i JavaScript i native kod, što omogućava da se poziv koji stiže od aplikacije se prilagodi za specifičnu platformu. Postoji čitav set plugin-ova koji je moguće dodati aplikaciju, a kao rezultat moći ćemo da pristupamo nativnim karakteristikama uređaja kao što su kamera, kontakti, geolokacija, pristup fajl sistemu i drugi. [5.]

Pluginovi se identifikuju po imenu, koje je uglvanom isto kao i ime njihovog npm paketa. Prema konvenciji ima plugin-a treba da počinje sa: cordova-plugin-, ali postoje uvek oni koji se toga ne pridržavaju. Ukoliko je potrebno da se da se plugin doda direktno u sam projekat, koristi se ista naredba koja se koristi i za Meteor pakete, ali ovog puta sa dodatkom za Cordovin plugin:

*meteor add cordova:cordova –plugin-camera@1.2.0*

Kod 1 – Uključivanje plugina za kameru

Kao što možemo da primetimo, kada plugin dodajemo na prethodno prikazani način, očekuje se da navede tačna verzija plugin-a, što može da bude malo otežavajuća okolnost, ali nije nerešiva. Još jedan način dodavanja pluginova je uz pomoć registrovanja zavisnosti u Meteor paketu na sledeći način:

*Cordova.depends({*

*‘cordova-plugin-camera’: ‘1.2.0’*

*});*

Kod 2 – Opcioni način za uključivanje plugina za kameru

Ukoliko pluginove dodajemo na ovaj način, moramo biti obazrivi da je moguće da različiti Meteor paketi uključe isti plugin, ali sa različitim verzijama, tako da neće nam biti poznato na kraju koja verzija nam je dostupna. Ukoliko se pak plugin dodaje direktno u projekat, kao što je pokazano u prvom slučaju (Kod 1), onda se ovakav problem ne može desiti, jer će uvek poslednja verzija da prepiše prethodne.

Što se tiče samog korišćenja pluginova, bilo koja funkcionalnost, koja se se oslanja na Cordovin plugin, mora da se nalazi unutar Meteor.startup() bloka, kako bismo se osigurali da je plugin u potpunosti incijalizovan pre korišćenja.

*Meteor.startup(function() {*

*navigator.geolocation.getCurrentPosition(success);*

*});*

Kod 3 - Meteor startup blok

Unutar same aplikacije je potrebno detektovati Cordovu unutar samog JavaScript koda i odvojiti ga uz pomoć sledećeg poziva:

*if (Meteor.isCordova){*

*console.log(“Ovo je vidljivo samo u mobilnim Cordova aplikacijama“);*

*}*

Kod 4 – Detektovanje Cordova koda

## 3.3 Leaflet

 Leaflet je vodeća open source JavaScript biblioteka koja omogućava rad sa interaktivnim mapama na mobilnom uređaju. Ona zauzima jako malo memorije, nekih 30-ak KB JavaScript koda, ali i pored toga nudi sve funkcionalnosti vezane za mape koje su programerima potrebne u izradi mobilnih aplikacija. Slika 11 – Leaflet logo

Leaflet je odrađen tako da se odlikuje jednostavnošću i dobrim performansama. Leaflet je korišćen u TripPlanner aplikaciji zbog mogućnosti rada sa mapama i u offline režimu. Moguće je njegove funkcionalnosti dodatno proširiti dodavanjem pluginova. Prilično je jednostavan za korišćenje i ima prilično čitljiv kod, tako da rad sa Leafletom ne predstavlja nikakav problem. Da bi Leaflet mogao da prikaže mapu neophodno je da najpre pribavi potrebne tile-ove, koji su ustvari delovi mape u vidu slike. On će to učiniti tako što će proslediti zahtev provider-u da mu obezbedi tileove, a zatim će njih postaviti kao mapu.U ovom slučaju za provider-a je korišćen OpenStreetMap, koji pruža podatke o mapama, kao i servere na kojima se ti podaci nalaze. Sledi primer postavljanja mape (Kod 5): [6.]

var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {

attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'

}).addTo(map);

Kod 5 – Postavljanje mape

Leaflet ima podršku i za postavljanje markera, što je moguće učiniti u svega par linija koda:

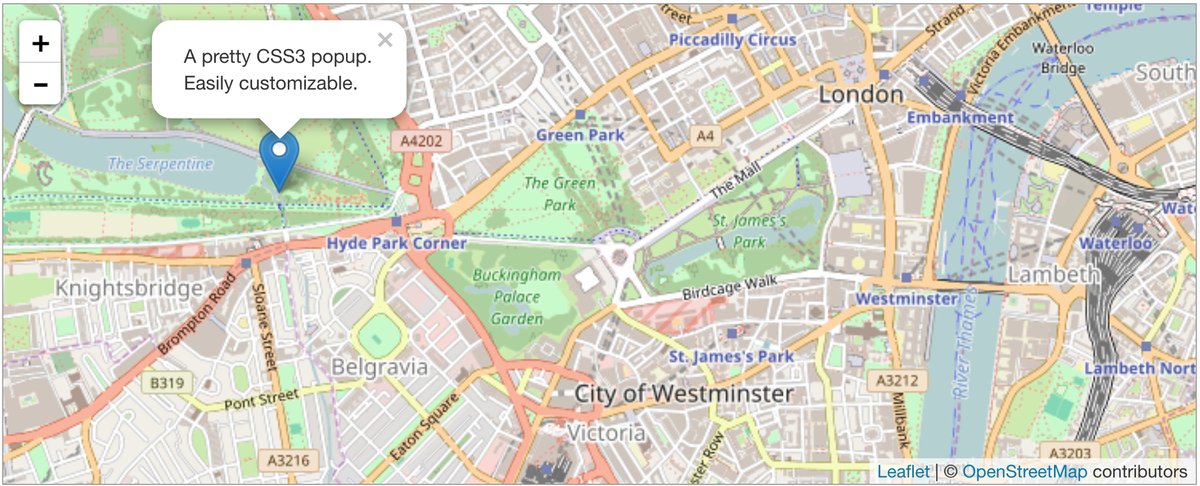
L.marker([51.5, -0.09]).addTo(map)

.bindPopup('A pretty CSS3 popup.<br> Easily customizable.')

.openPopup();

Kod 6 – Postavljanje markera na mapi

Kao rezultat prethodnih delova koda dobijemo sledeću mapu sa postavljenim markerom i popup prozorom:



Slika 12 – Mapa postavljena korišćenjem Leafleta

Moguće je iscrtavati i razne oblike na samoj mapi, poput kružnica, kao i obrađivati događaje koji su nastali prilikom interakcije sa mapom, kao što je klik na mapu ili marker, zatim zumiranje mape i slično.

function onMapClick(e) {

popup

.setLatLng(e.latlng)

.setContent("You clicked the map at " + e.latlng.toString())

.openOn(mymap);

}

mymap.on('click', onMapClick);

Kod 7 – Klik na mapu

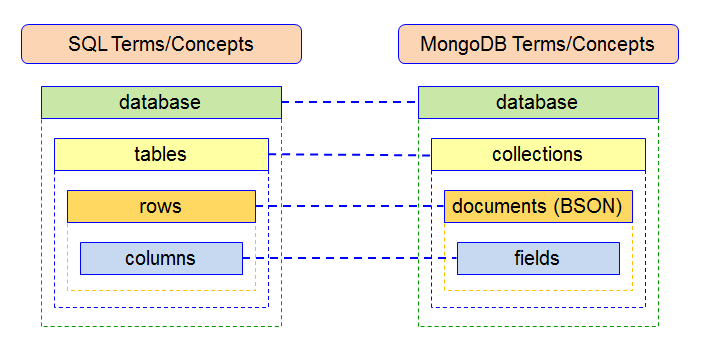
Leaflet u kombinaciji sa mapama koje su preuzete od OpenStreetMap provajdera, funkcioniše jako dobro i pouzdano i u offline režimu, što predstavlja zahtev sa kojim se ovih dana sve češće susrećemo, a za šta mnoge druge biblioteke nemaju podršku.

## 3.4 Tehnologije za skladištenje podataka na mobilnom uređaju

Gotovo sve aplikacije sa kojima se susrećemo rade sa velikom količinom podataka koju je potrebno najpre negde skladištiti, a zatim i obradjivati. Baza u kojoj ćemo skladištiti podatke zavisi od toga kakvi su zahtevi same aplikacije. U okviru TripPlanner aplikacije korišćeno je više načina skladištenja, među kojima je najzastupljeniji MongoDB koji dolazi kao direktna podrška uz Meteor. Mongo kao što smo već rekli predstavlja izuzetno brzu i pouzdanu bazu podataka koja ima mogućnost perzistiranja podataka, pa je iskorišćena za skladištenje podataka na serveru. Preostale baze koje su korišćene na klijentu (Minimongo, SQLite i Local Storage) imaju neke svoje prednosti i nedostatke, zbog kojih je kod izrade TripPlanner aplikacije bilo neophodno njihovo kombinovanje. U nastavku ovog poglavlja ćemo se detaljno upoznati sa svakom od njih.

### 3.4.1 MongoDB

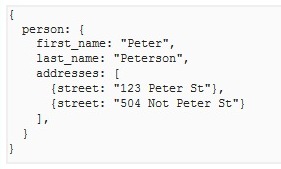
MongoDB spada u kategoriju document-oriented baza podataka. Ova kategorija podrazumeva baze podataka čija je osnovna namena skladištenje, pribavljanje i upravljanje document-oriented podacima ili polustruktuiranim podacima. Kod relacionih baza podataka smo imali slučaj da su podaci organizovani oko koncepta relacije (tabele), dok su document-oriented baze podataka organizovane oko koncepta dokumenta sa različitim nivoima detaljnosti. Ovakav tip baze podataka je jako zgodan za rad, jer ne zahteva striktno definisanje šeme podataka, a to nam je bilo neophodno u TripPlanner aplikaciji. Ne moraju svi dokumenti da imaju istu strukturu. Kod relacionih baza podataka je ideja da sve vrste moraju da imaju iste kolone, što kod document-oriented uglavnom nije slučaj. Ukoliko nam je kasnije potrebno da dodamo još neko polje, to možemo uciniti, bez ikakvog uticaja na prethodno skladištene dokumente. Takođe, kod ovog tipa baze podataka, ukoliko za neko polje je vrednost nepoznata ili nedefinisana u tom slučaju se ono ne čuva. Na slici 13 možemo videti neke razlike koje postoje u terminologiji između SQL i Mongo baze podataka. [7.]



Slika 13 – Razlike u terminologiji između SQL i Mongo baze

Kao što možemo videti MongoDB predstavlja kolekciju dokumenata koji se pak sastoje od skupa odgovarajućih polja. Koncept tabele je sada poistovećen sa konceptom kolekcije. Ono što je bila vrsta u SQL bazi podataka, u document-oriented bazi to je dokument, dok se kolona može poistovetiti sa poljem.

U MongoDB bazi, što je ujedno slučaj i sa drugim document-oriented bazama podataka, objekti se snime u dokument i tako čuvaju u bazi. Dokumenti mogu da budu i dosta kompleksni, jer je moguće čitav objektni model snimiti odjednom, a samim tim izbegavamo veliki broj CRUD operacija. Dokumenti su najčešće nezavisni što doprinosi povećanju performansi. Mongo baza podataka je namenjena za skladištenje JSON (JavaScript Object Notation) dokumenata. JSON je tekstualni format koji je razumljiv i ljudima i mašinama i zbog svoje jednostavnosti i dobre preglednosti je jako zastupljen.



Slika 14 - Primer JSON dokumenta

Slede neke od bitnijih karakteristika koje ima svaka Mongo baze podataka:

* Podrška za **indeksiranje** dokumenata. MongoDB indeks je struktura podataka koja nam omogućava brzo lociranje dokumenata, na osnovu vrednosti određenih polja u dokumentima. Indeksi mogu biti jako korisni, jer doprinose efikasnijem izvršenju upita u Mongo bazi podataka. Ukoliko indeksi ne bi postojali, Mongo bi morao da skenira svaki dokument u kolekciji kako bi mogao da selektuje one koji odgovaraju zadatom upitu. Sa postojanjem indeksa on ograničava broj dokumenata koje mora da pretražuje. Koncept indeksa je jako dobar kod pretraživanja, ali može da uspori proces ažuriranja podataka.
* **Kompleksni upiti**. Kod ovog tipa baze podataka, ne primenjuje se koncept spajanja tabela, tako da sve što nam je potrebno držimo u jednom dokumentu. Samim tim kasnije kada kreiramo upit, on zna da bude jako kompleksan.
* Visok nivo **skalabilnosti**. MongoDB obezbedjuje horizontalnu skalabilnost što spada u jednu od njegovih bitnih funkcionalnosti.
* Podršku za **replikaciju.** Replikacija nam obezbedjuje redudantnost i povećenu dostupnost podataka. Ideja je da postoji više kopija podataka koje se čuvaju na različitim serverima da bi se smanjila mogućnost gubljenja podataka ukoliko se neki od servera pokvari ili bude nedostupan.
* Visok nivo **dostupnosti.** Ova karakteristika je direktna posledica postojanja skupa replikacionih servera kojim se povećava redudantnost podataka i oporavak u slučaju nedostupnosti jednog od servera.
* **Map / Reduce** podrška. Ova podrška nam omogućava paralelnu obradu velike količine podataka koja se nalazi na distribuiranim čvorovima. Odvija se u dva koraka:
  + Map – Imamo jedan master čvor koji prihvata ulaz, a zatim ga deli ga na manje potprobleme i distribuira ih worker čvorovima. Worker čvor može ponoviti isti korak kreirajući stablo čvorova koji obavljaju procesiranje podataka. Worker čvor obrađuje manji problem, i rezultate obrade prosleđuje master čvoru.
  + Reduce – master čvor prikuplja od worker čvorova rezultate obrade

Zbog svih ovih prednosti i funkcionalnosti, kao i integrisanosti sa Meteor frameworkom, Mongo baza podataka je u TripPlanner aplikaciji korišćena kao primarna baza na serveru gde su skladištene sve neophodne kolekcije podataka.

### 3.4.2 MiniMongo

U osnovi Meteor framework je postojanje i keš memorije, poznate kao MiniMongo. To je zapravo primer jednostavne baza podataka koja je smeštena na klijentu. Ova baza funkcioniše identično kao i Mongo baza, osim što je možemo smatrati neperzistentnom, pa ukoliko želimo da neke podatke sačuvamo nakon gašenja aplikacije u offline režimu, onda je neophodno koristiti neki drugi način za perzistiranje, kao što je u ovom radu, za tu svrhu, korišćena SQLite baza.

Razlike izmedju MiniMongo i Mongo baze podataka su neznatne, tako da sve upite koje kreiramo i izvršavamo na serveru je moguće izvršavati i nad MiniMongo bazom podataka na klijentu. Postojanje ove baze u TripPlanner aplikaciji nam je omogućilo da podatke koje nam publishuje server, možemo čuvati lokalno na klijentu i samim tim značajno se smanjuje potreba za stalnom mrežnom komunikacijom sa serverom, što može biti jako nezgodno i sporo kada koristimo mobilni internet.

Postojanje konekcije izmedju MiniMonga i Mongo baze podataka je bitna funkcionalnost TripPlanner aplikacije iz razloga što je ta konekcija reaktivna. To znači da ove dve baze, kada su povezane na internet, imaju ugradjen mehanizam za sinhronizaciju podataka. Ukoliko dodje do nekog upisa u MiniMongo bazu na klijentu, ta promena se automatski salje Mongo bazi podataka na serveru, koji dalje obaveštava sve ostale klijente o nastaloj promeni. Ovaj ugradjeni koncept sinhronizacije Meteora je u mnogome olakšao izradu online verzije TripPlanner aplikacije, a u osnovni samog koncepta leži ugradjena reaktivnost Meteor aplikacija.

### 3.4.3 SQLite baza podataka

SQLite spada u grupu Embedded baza podataka koju čine sistemi za upravljanje bazama koji su usko povezani sa aplikacijom koja vrši obradu podataka. Ove baze su najčešće sakrivene od krajnjeg korisnika, što bi značilo da on uopšte nije svestan njihovog postojanja. Jako su pogodne za korišćenje jer zahtevaju vrlo malo ili nimalo održavanja.

Njihove najznačajnije karakteristike su ujedno bile i ključne karakteristike zbog kojih je predstavnik ove baze SQLite korišćen u izradi TripPlanner aplikacije, a to su :

* **Mobilnost** - Mobilnost predstavlja osnovu sistemima baziranim na mobilnim bazama podataka.
* **Disconnected režim rada** – Podaci koji se čuvaju se nalaze bliže aplikaciji, pa samim tim nije potrebna stalna mrežna aktivnost, već je omogućeno da i kada smo offline možemo da radimo sa lokalno skladištenim podacima. Ovo je glavni razlog zbog koga smo u projektu za perzistiranje u offline režimu koristili upravo SQLite bazu podataka.
* **Privremeno keširanje podataka** – U bazi je moguće i privremeno smeštanje podataka koji su nam neophodni dok ne prodje odredjena operacija, nakon čega ih ne moramo dalje skladištiti.
* **Sastavni deo aplikacije** - zbog svoje jednostavnosti i male količine memorije koju zahteva (oko 275 KB), SQLite predstavlja sastavni deo velikog broja aplikacija.

Čitav DBMS se nalazi u biblioteci koja je implementirana korišćenjem C programskog jezika. Aplikacije mogu da linkuju SQLite biblioteku na 2 načina, statički i dinamički. Statički podrazumeva da biblioteka postaje sastavni deo izvršne datoteke aplikacije, dok dinamički bi značilo da se ona učitava po potrebi.

Kao što smo već rekli, zbog male količine memorije koju zahteva, SQLite baza podataka je jako pogodna za uređaje koji raspolažu ograničenom memorijom. Međutim ta memorija koja je na raspolaganju direktno utiče na performanse same biblioteke. [8.]

SQLite baza podataka ima delimičnu podršku za trigere kao i nemogućnost ažuriranja pogleda. Takođe treba spomenuti da je ovde ograničavajuća i ALTER TABLE podrška, zbog nemogućnosti izmene i biranja kolona. Što se tipova podataka tiče, kod SQLite baze su zastupljeni slabi tipovi podataka. Tip podatka se dodeljuje konkretnoj vrednosti, a ne koloni u tabeli. U svaku kolonu može da se upiše bilo koja vrednost. U zavisnosti od deklaracije kolone SQLite će pokušati da izvrši konverziju, a ako konverzija nije moguća, upisaće zadatu vrednost.

U nastavku će biti nabrojane prednosti zbog kojih je SQLite baza podataka korišćena u TripPlanner aplikciji:

* zero-configuration
* smeštena je u samo jednoj datoteci, koja je pritom portabilna
* kompaktne je veličine od svega par stotina KB.
* ima mogućnost dinamičkog određivanja tipova
* promenljiva veličina vrsta

### 3.4.4 HTML5 Local Storage

I pored svih prethodno navedenih načina za skladištenje podataka, u okviru ovog projekta javila se potreba za korišćenjem Local Storage tehnologije, koja spada u grupu novijih tehnlogija. Cilj Local Storage-a je da omogući lokalno skladištenje podataka, ali u okviru pretraživača. Ranije se ovaj isti proces obavljao tako što su se podaci smeštali u cookie-e, prilikom svakog obraćanja serveru. [9.]

Ideja sa Local Storage načinom skladištenja je jako dobro prihvaćena kod programera zbog činjenice da je ova tehnologija mnogo sigurnija, a ujedno omogućava i skladištenje daleko veće količine podataka nego što je to moguće sa cookie-ima, a pritom ne utiče na performanse web sajta. HTML Local Storage obezbeđuje 2 tipa objekta za skladištenje podataka na klijentu:

* window.localStorage - skladišti podatke na neodređeno vreme. Podaci se čuvaju i nakon što se pretraživač zatvori.
* window.sessionStorage – skladišti podatke za jednu sesiju. U ovom slučaju podaci će se izgubiti nakon što se pretraživač zatvori.

Pre korišćenja local storage-a, uvek treba najpre proveriti podršku za rad sa local i session storage-om:

if (typeof(Storage) !== "undefined") {  
    // *Code for localStorage/sessionStorage.*  
} else {    // There is No Web Storage support.}

Kod 8 – Provera da li postoji podrška za Web Storage

U sledećem delu koda (Kod 9) prikazano je kako se smestaju podaci u local i session storage, a zatim i čitaju odatle.

localStorage.lastname = "Smith"; // Store  
document.getElementById("result").innerHTML = localStorage.lastname; // Retrieve

Kod 9 – Provera da li postoji podrška za Web Storage

Local Storage, kao što je već spomenuto, može do skladišti veću količinu podataka u osnosu na cookie-e. Skladišti do 5MB podataka i pritom se podaci nikada šalju na server, već se čuvaju lokalno. Za rad TripPlanner aplikacije u offline režimu ovih 5MB je bilo ograničavajuće, pa je pored Local Storage-a uvedena i prethodno objašnjena SQLite baza podataka, a LocalStorage je korišćen za čuvanje datuma poslednjih sinhronizacija aplikacije sa serveru.

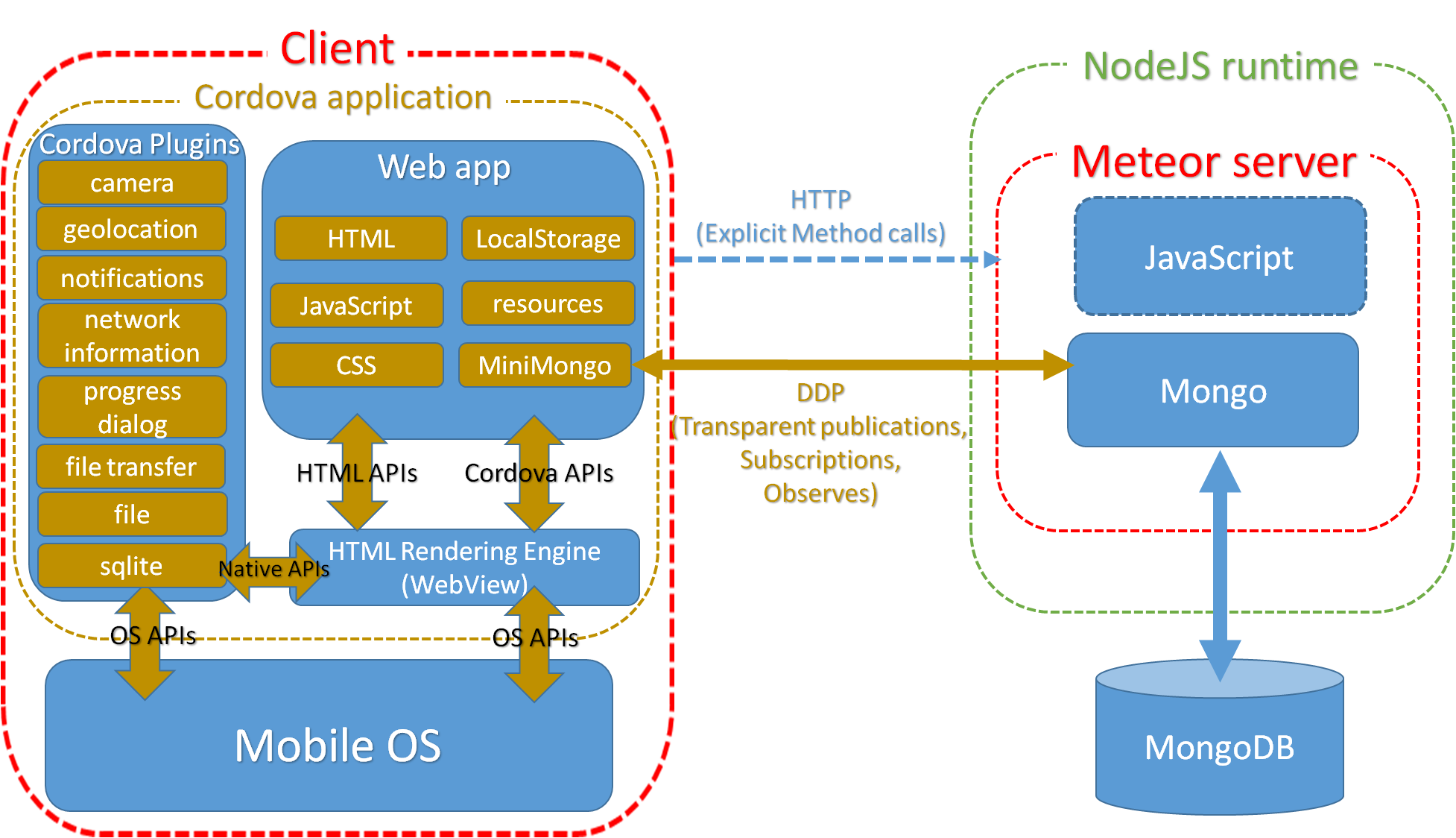
# 4. Mobilna aplikacija za planiranje obilazaka turističkih znamenitosti

Nakon poglavlja u kojem su detaljno opisane tehnologije koje su korišćene u izradi TripPlanner aplikacije, sledi četvrto poglavlje u kojem ćemo proučiti samu aplikaciju. Najpre ćemo se upoznati sa arhitekturom same aplikacije, a zatim i sa svim njenim funkcionalnostima. Potom sledi implementacija aplikacije, odnosno neki karakteristični delovi propraćeni kodom.

## 4.1 Arhitektura aplikacije

Arhitektura same TripPlanner aplikacije zasnovana je na postojanju serverskog, klijentskog dela kao i komunikacionog kanala koji postoji između njih (slika 15). U nastavku ovog poglavlja ćemo detaljno proučiti svaki od navedenih delova.

**Server** aplikacije se izvršava na Node.js-u što se može videti i na slici 15. Server sadrži JavaScript kod koji doprinosi uspešnoj komunikaciji sa klijentom i Mongo bazom podataka. Taj kod zapravo omogućava publikaciju podataka ka klijentu što je posledica postojanja publish/subscribe pattern-a. Kao što smo već spomenuli, ukoliko se ovaj pattern ne bi koristio, klijentu bi se slale čitave kolekcije podataka, što smo svakako hteli da izbegnemo u TripPlanner aplikaciji. Takođe na serveru postoje i metode. To su funkcije koje omogućavaju klijentu da direktno obavlja operacije kao što su brisanje, ažuriranje i dodavanje podataka u glavnoj bazi na serveru. Prilikom poziva neke od ovih metoda, nastale promene se automatski propagiraju do svih ostalih klijenta, što je omogućeno dobro razvijenim mehanizmom sinhronizacije koji postoji kod online Meteor aplikacija. Na serveru se takođe nalazi i Mongo baza podataka, koja predstavlja primarnu bazu TripPlanner aplikacije. Ona sadrži kolekcije svih podataka koji su neophodni za pravilan rad aplikacije. Podaci koji se nalaze u ovoj bazi su perzistentni. Takođe zbog mehanizma reaktivnosti koja postoji u Meteor aplikacijama, pa i u TripPlanner aplikaciji, omogućena je jako dobra sinhronizacija podataka koji se nalaze na klijentu, sa podacima koji su smešteni na serveru. Što bi značilo da ukoliko klijent u online režimu rada aplikacije lokalno izmeni neke podatke, ta promena biće odmah vidljiva i na serveru, a potom će se propagirati i do svih ostalih klijenata, čime je osigurano da svi klijenti u isto vreme vide iste podatke.



Slika 15 – Arhitektura TripPlanner aplikacije

Pored serverskog dela, TripPlanner aplikacija sadrži i **klijentski deo**. Klijentska komponenta je sačinjena od web aplikacije i Cordovinih plugin-ova koji su obavijeni Cordovinim wrapperrom što ovu našu mobilnu aplikaciju čini mobilnom hibridnom aplikacijom. Naime zahvaljujući Cordovi, klijentska aplikacija može da pristupa nativnim karakteristikama uređaja iako je pisana kao klasična web aplikacija. Takođe ju je moguće kompajlirati da radi za različite mobilne operativne sisteme kao što su Android i iOS. Što se tiče web dela klijentske aplikacije ona se sastoji od HTML, CSS i JavaScript koda, a za skladištenje podataka su korišćeni Minimongo baza podataka, HTML5 Local Storage, kao i SQLite plugin. HTML delovi koda su korišćeni u vidu templejta što je omogućilo da isti kod koristimo više puta na različitim mestima. CSS je doprineo lepšem dizajnu same aplikacije, a JavaScript kod je korišćen kako bi se odradila logika čitave aplikacije. Sto se načina za skladištenje tiče Minimongo baza podataka predstavlja in-memory bazu podataka i omogućava lokalno skladištenje podataka na koje se klijent pretplatio. Ona jako lepo funkcioniše u kombinaciji sa Mongom, jer omogućava sinhronizaciju podataka u online režimu, o kojoj je već bilo reči. Međutim Minimongo baza ne može da perzistira podatke, što je prvenstveno bio problem kada je rađena offline verzija aplikacije. Naime da bi aplikacija mogla da radi u offline režimu i da pritom ima sve iste funkcionalnosti kao i u online režimu, neophodno je sve mape, komentare i objekte za koje je korisnik zainteresovan smestiti lokalno Takođe sve izmene koje korisnik napravi offline treba sačuvati dok se ne uspostavi internet konekcija, kako bi se oni sinhronizovali sa glavnom bazom na serveru. Za sve ove prethodno navedene potrebe je korišćen SQLite plugin, jer nam omogućava lokalnu perzistenciju podataka. Takođe uz SQLite plugin kao dodatna mera perzistencije podataka korišćen je HTML5 Local Storage, kog smo već detaljno obradili.

Pošto Meteor poseduje ugrađenu podršku za Cordovu, nativnim karakteristkama pristupamo preko Cordova-inog wrapper-a, odnosno Cordova-inih plugin-a koji omogućavaju JavaScript API za korišćenje nativnih karakteristika uređaja kao što su SQLite, fajl sistem, kamera, galerija, geolokacija, kontakti, itd. U nastavku biće dat pregled korišćenih plugina i nativnih karakteristika uređaja.

Korišćeni Cordova-ini plugin-i:

* **Plugin za korišćenje SQLite baza podataka** - [*cordova-sqlite-ext@0.10.1*](mailto:cordova-sqlite-ext@0.10.1)

Omogućava Javascript interfejs, sličan WEB SQL API-u, za rad sa SQLite bazama podataka. Podržava transakcije, prethodno popunjene baze i BLOB-ove kao tip podataka.

Potrebno je kreirati referencu na bazu podataka i nakon toga je moguće izvršavanje naredbi standardnom SQLite notacijom u okviru transakcije ili bez (Kod 15)

**var** db = **null**;

document.addEventListener('deviceready', **function**() {

db = window.sqlitePlugin.openDatabase({name: 'demo.db', location: 'default'});

});

Kod 10. Otvaranje SQLite baze podataka sa imenom „demo.db“, koja se nalazi na standardnoj lokaciji

* **Plugin za pristup fajl sistemu uređaja** - [cordova-plugin-file@4.3.0](mailto:cordova-plugin-file@4.3.0)

Omogućava pristup fajl sistemu uređaja i rad sa fajlovima. U aplikaciji se koristi prilikom smeštanja preuzetih SQLite baza sa tile-ovima za prikaz mape u offline režimu, na odgovarajuće mesto.

* **Plugin za preuzimanje i upload fajlova** - [cordova-plugin-file-transfer@1.6.0](mailto:cordova-plugin-file-transfer@1.6.0)

Plugin definiše globalni FileTransfer objekat, koji je dostupan nakon deviceready događaja. U aplikaciji se koristi, u kombinaciji sa plugin-om za pristup fajl sistemu, za preuzimanje SQLite baza koji sadrže tile-ove za određeno područje (grad) i njihovo smeštanje u fajl sistem uređaja, radi kasnijeg prikazivanja kada je aplikacija u offline režimu rada.

* **Plugin za prikaz nativnog progres dialoga** - [cordova-plugin-pdialog@2.0.1](mailto:cordova-plugin-pdialog@2.0.1)

Plugin omogućuva rad sa nativnim progres dialozima i u aplikaciji se koristi da bi se prilikom preuzimanja fajla sa mapom određene oblasti sa udaljenog fajl servera, prikazao progres prezimanja.

* **Plugin za informaciju o stanju mrežne konekcije uređaja** - [cordova-plugin-network-information@1.3.0](mailto:cordova-plugin-network-information@1.3.0)

Plugin pruža imlementaciju API-a o mrežnim informacijama, odnosno informacije o wifi i konekciji mobilnog interneta uređaja i da li je uređaj uopšte povezan na internet. U aplikaciji se koristi prilikom provera da li je uređaj online ili offline (preko navigator.connection.type), zbog različitih režima rada aplikacije. Takođe se u aplikaciji koriste listeneri za prelazak u offline ili online stanje.

* **Plugin za pribavljanje informacija u vezi korisnikove lokacije** - [cordova-plugin-geolocation@2.3.0](mailto:cordova-plugin-geolocation@2.3.0)

Plugin omogućava informacije u vezi korisnikove lokacije, kao što su geografska širina i dužina. Uobičajeni izvori koje plugin koristi uključuju Global Positioning System (GPS) i lokacije zaključene preko mrežnih signala kao što su IP adrese, RFID, WiFi i Bluetooth MAC adrese i GSM/CDMA ID-evi ćelija. Plugin omogućava informacije na osnovu kojih se prikazuje korisnikova lokacija na mapi u realnom vremenu, i u online i u offline režimu tako da može da prati svoje kretanje na mapi.

* **Plugin za prikaz notifikacija u status baru** - [cordova-plugin-local-notifications-mm@1.0.4](mailto:cordova-plugin-local-notifications-mm@1.0.4)

Plugin omogućava prikaz nativnih notifikacija u status baru android-a. U aplikaciji se koristi za obaveštenja korisniku o postojanju njemu podeljenog plana od strane njegovog prijatelja/korisnika.

* **Plugin za prikaz nativnih dialoga sistema** - [cordova-plugin-dialogs@1.3.0](mailto:cordova-plugin-dialogs@1.3.0)
* Korišćen je i **Meteor-ov paket mdg:camera** koji omogućava korišćenje nativne kamere i izbor slika iz galerije, koje se mogu dodati prilikom kreiranja recenzija i registrovanja

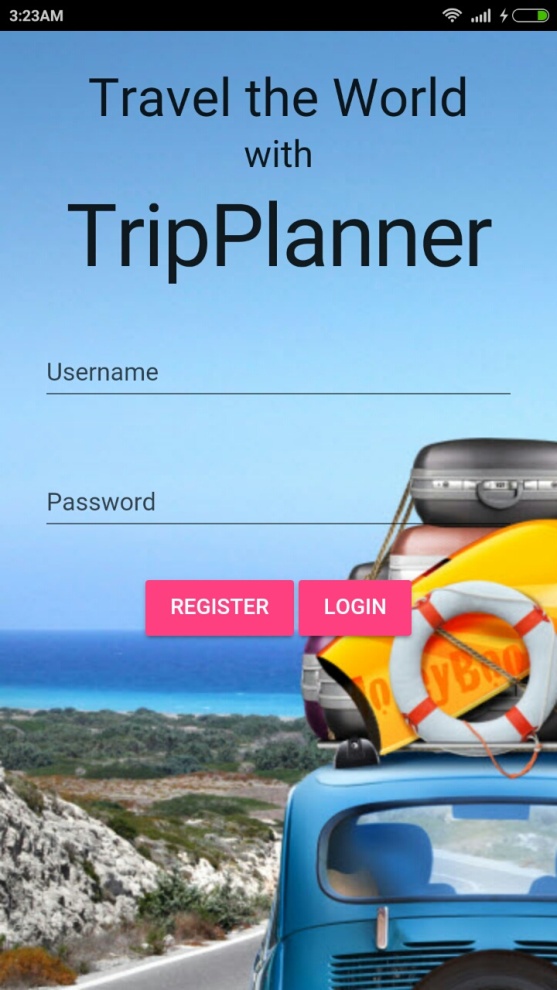
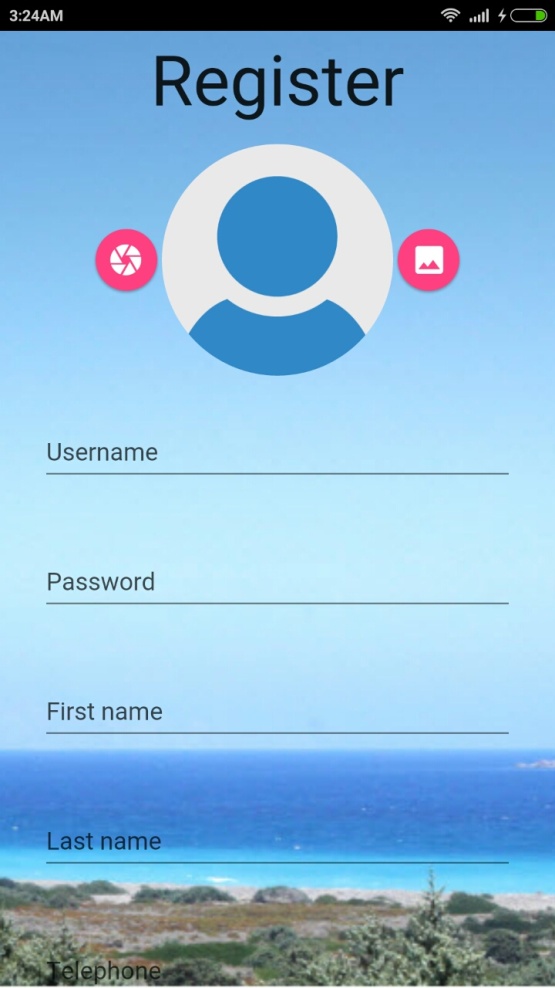
Nakon detaljnog pregleda svih komponenti klijentske aplikacije, ostalo je još sagledati na koji to način klijentska komponenta komunicira sa serverskom. Zato ćemo kao treću bitnu komponentu spomenuti i komunikacioni kanal preko koga se odvija transfer podataka sa klijenta na server i obrnuto. Ovde se koristi koristi Distrubuted Data Protocol (DDP) koji je primer REST servisa koji funkcioniše preko Web soketa. Ovaj soket je zadužen za prenos podataka u JSON formatu izmedju klijenta i servera. Za uspostavljanje konekcije se u pozadini koristi socket.io biblioteka. Suština komunikacije izmedju klijenta i servera se obavlja preko soketa korišćenjem DDP protokola. Ovaj protokol je čitljiv od strane ljudi i moguće je pratiti šta se prenosi soketom korišćenjem odgovarajućeg DDP analizatora. DDP protokol nije striktno vezan za Meteor, već ga je moguće koristiti i u drugim projektima, gde god postoji implementiran mehanizam soketa. Soketi obavljaju transfer podataka mnogo brže nego što bi to činio HTTP protokol i samim tim je mnogo pogodniji u izradi reaktivnih aplikacija.

Nakon svih pregledanih komponenti možemo uočiti da je arhitektura TripPlanner aplikacije kompleksnija od klasične Meteor aplikacije, zbog korišćenih pluginova, sa ciljem da se proširi funkcionalnost same aplikacije. U narednom poglavlju ćemo se upoznati detaljno sa funkcionalnostima aplikacije, dok će implementacija karakterističnih delova koda biti prikazana u dodatku A.

## 4.2. Opis funkcionalnosti TripPlanner aplikacije

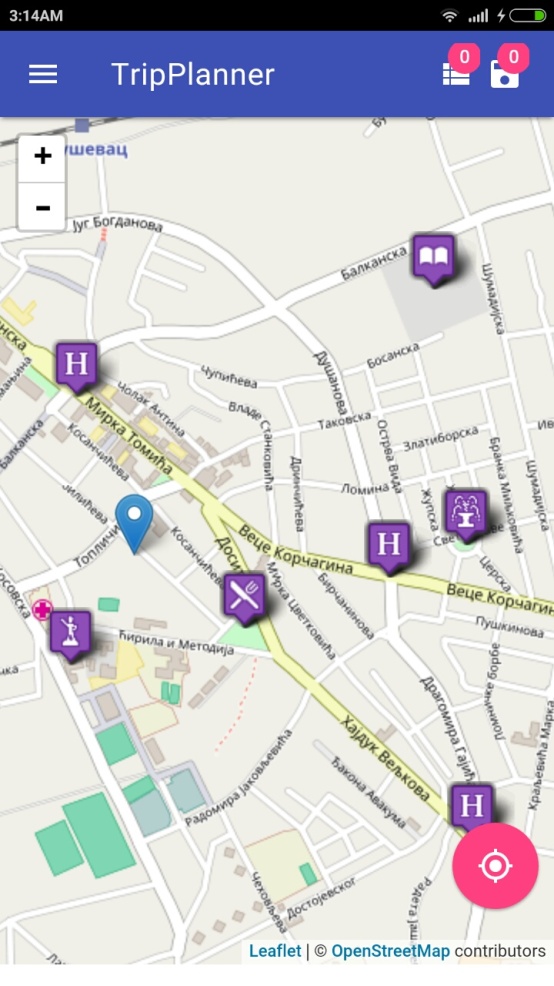
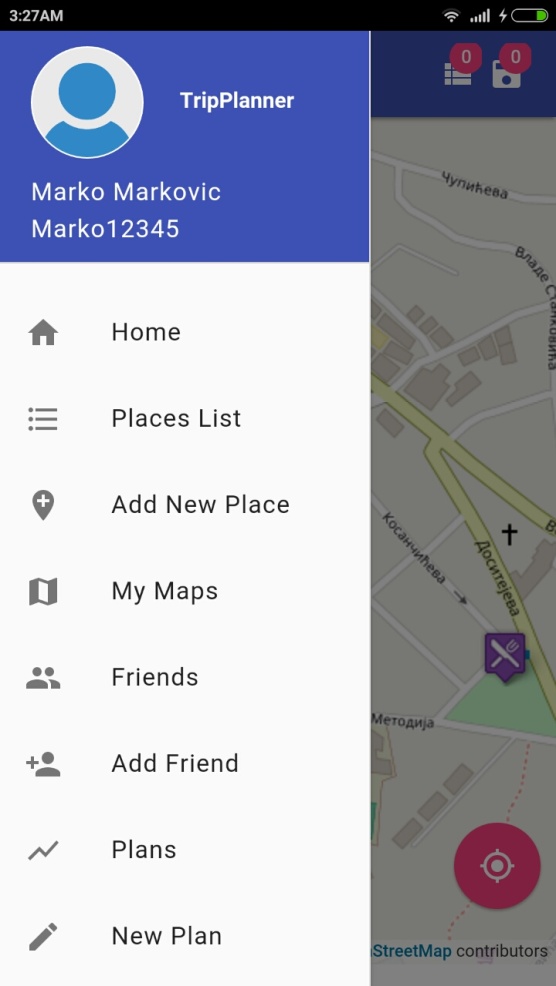
TripPlanner je mobilna aplikacija čija je namena da se korisnicima omogući lakše planiranje obilazaka turističkih znamenitosti u nekom gradu, a pritom je prilagođena Android uređajima. Iako u oblasti turizma postoji veliki broj aplikacija sa različitim funkcionalnostima, TripPlanner spada u grupu retkih aplikacija koja mogu da rade u online, ali i u offline režimu, pri čemu su sve funkcionalnosti u oba slučaja uvek dostupne. Ideja za izradom ovog tipa aplikacije proistekla je iz činjenice da na tržistu nema mnogo aplikacija koje služe kao turistički planeri konkretno za gradove Srbije, pogotovo one manje.

Nakon kraćeg uvoda u samu aplikaciju u ovom delu biće detaljno opisane funkcionalnosti same aplikacije. TripPlanner, kao i veći broj aplikacija, na prvom ekranu zahteva od korisnika da se uloguje ili registruje, kao na slici 16. Prilikom prvog susreta sa TripPlanner aplikacijom neophodno je proći kroz proces registracije, pošto je ideja aplikacije da se za svakog korisnika čuvaju odredjene informacije, njegovi planovi, mape i slično. Na Slici 17 je prikazana forma za registrovanje gde se od korisnika traže neki osnovni podaci kojim bi se on identifikovao, ali se nudi i mogućnost unosa slike, bilo preko kamere ili iz galerije. Nakon obavljene registracije, korisnik sada može da se uloguje u aplikaciju, a svaki naredni put ulaziće direktno u aplikaciju, posto se od njega neće tražiti ponovno logovanje.

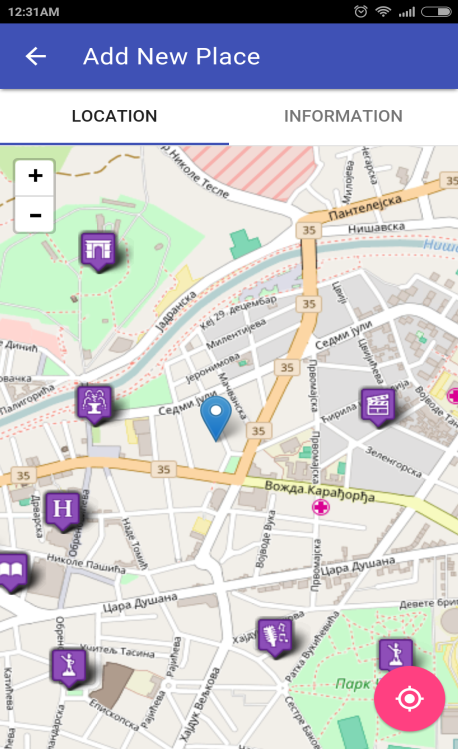
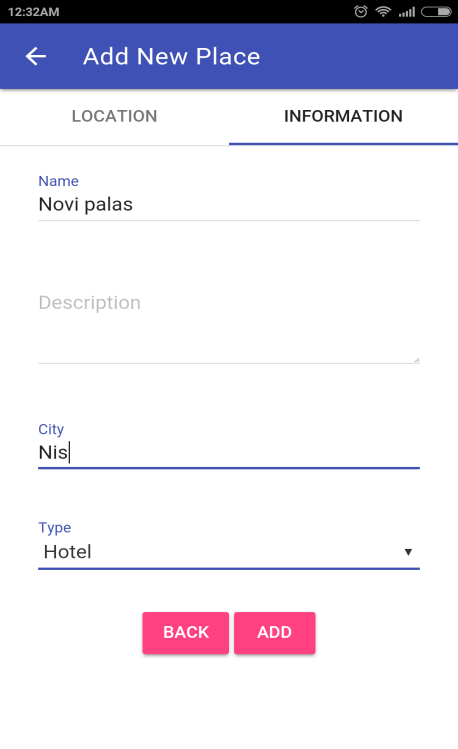
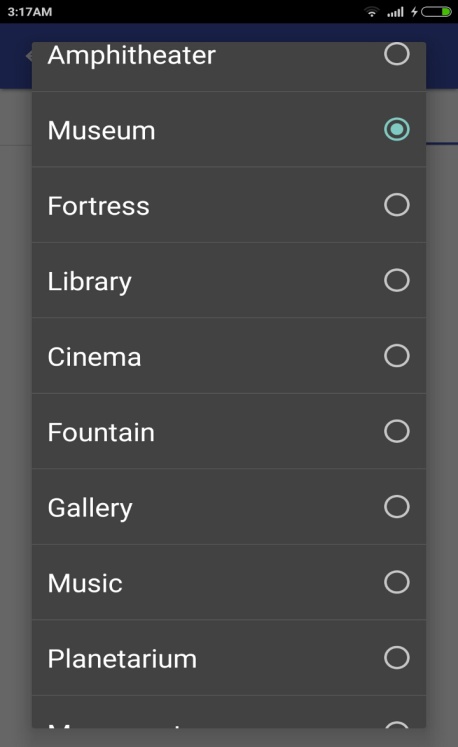
Slika 16 – Ekran za logovanje Slika 17 – Ekran za registrovanje

Nakon uspešno obavljenog logovanja korisnikovi podaci su evidentirani u bazi podataka i korisniku se prikazuje početna strana aplikacije (slika 18) sa mapom i označenom lokacijom na kojoj se korisnik sada nalazi. Ukoliko korisnik prstom prevuče preko ekrana s leva u desno, otvoriće mu se slide drawer kao na slici 19. Ovaj slide drawer sadrži osnovne podatke o korisniku, ali prikazuje i mnoge funkcionalnosti aplikacije kao što su : Add new place, My maps, Friends, Add Friend, Plans i New Plan. Sada ćemo proći detaljno kroz svaku od njih.

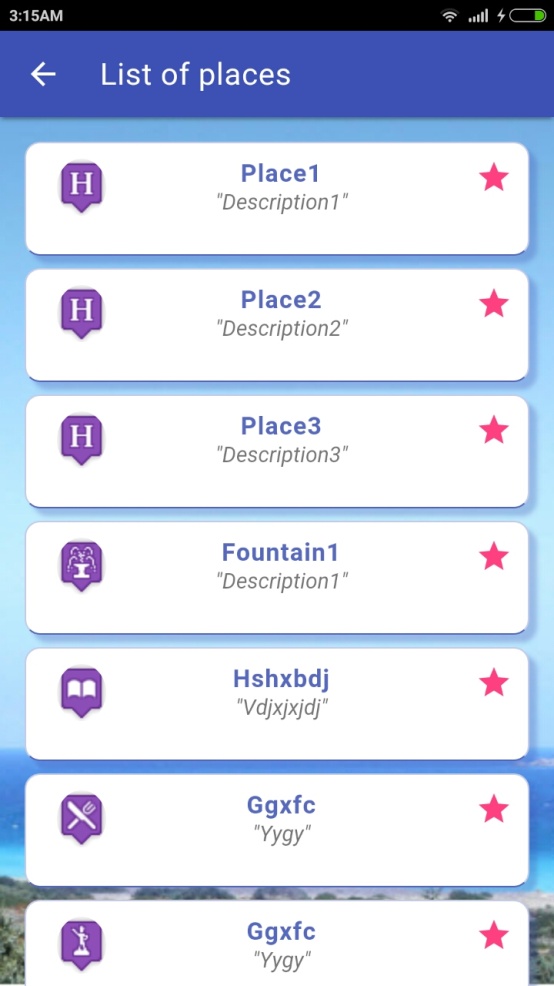
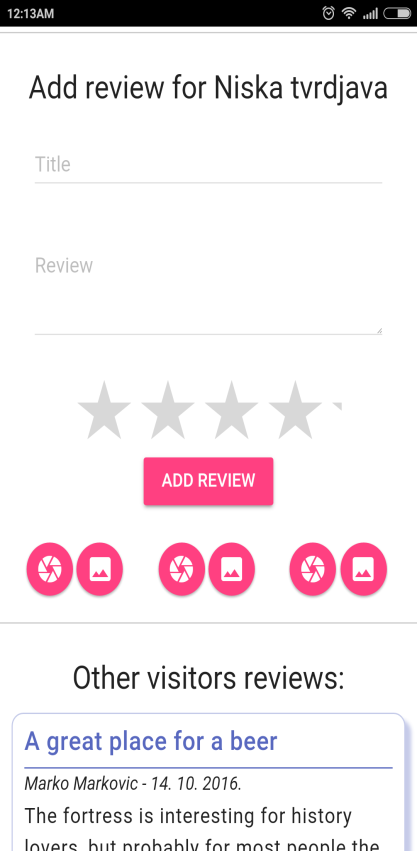
 

Slika 18 – Početna strana Slika 19 – Slide drawer

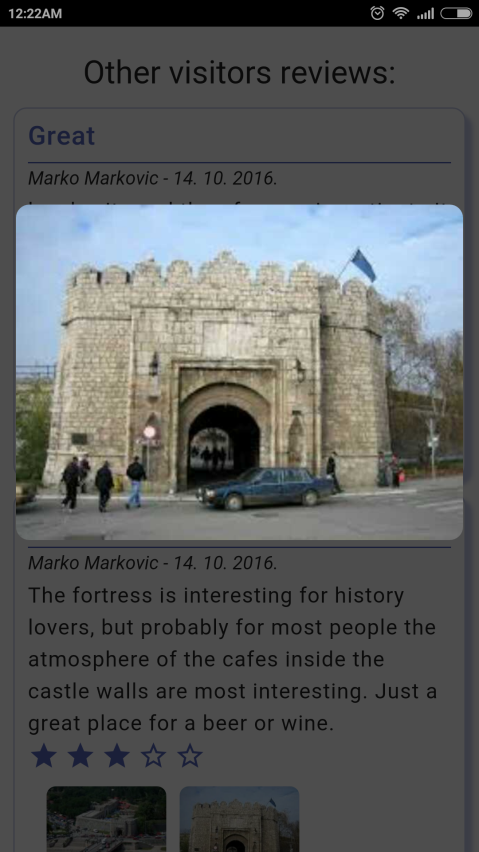
Ideja aplikacije je da na mapi koja se prikazuje korisniku budu iscrtane znamenitosti koje on može da poseti, ali korisnik može da doda na mapu neku znamenitost koja ne postoji. Prilikom dodavanja korisniku se prikaže mapa gde on klikom na mapu označi gde će znamenitost biti pozicionirana. Potom je potrebno da unese i podatke kao što su naziv, opis i tip dodatog objekta.

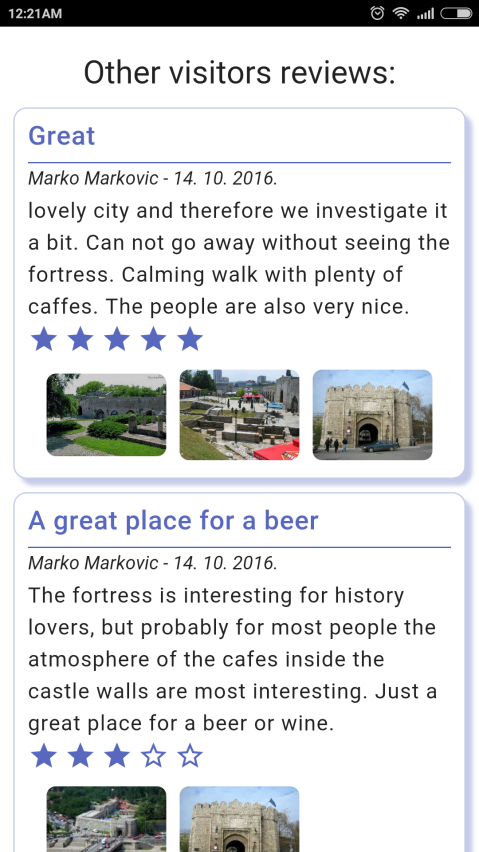
   Slika 20 – Odabir pozicije objekta Slika 21 – Unos detalja Slika 22 – Odabir tipa objekta

Nakon što korisnik doda znamenitost, ona je automatski vidljiva na mapi, a to je moguće videti i klikom na Places List iz slide drawer-a. U tom slučaju otvara se lista znamenitosti za izabrani grad kao na slici 23. Korisnik sada može klikom na svaku od znamenitosti da sazna više informacija (slika 24), ali i da podeli svoje utiske sa drugima kao na slici 25.

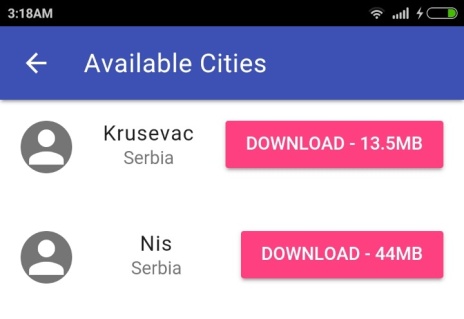
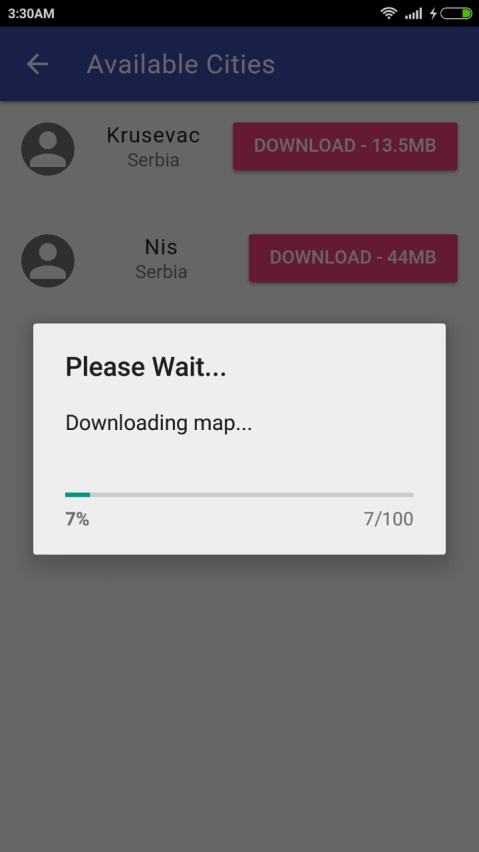
** ** 

Slika 23 – Početna strana Slika 24 – Prikaz znamenitosti Slika 25 – Unos komentara

Prilikom unosa komentara (slika 25) od korisnika se oćekuje da stavi naslov komentara, tekst, da oceni znamenitost klikom na zvezicu, a omogućeno je i da za svaki komentar unese do 3 slike. Te slike mogu biti preuzete iz galerije telefona kao na slici 26 ili korisnik može da odabere opciju da mu se pokrene kamera i da sam uslika nesto što želi da podeli sa drugim korisnicima. Ova opcija mu je omogućena jer je u projektu korišćen Cordovin plugin koji nam dozvoljava da pristupimo nativnoj karakteristici uredjaja, a u ovom slučaju je to kamera. Nakon što je komentar dodat, u online režimu rada aplikacije on odmah postaje vidljiv drugim korisnicima (slika 27), dok u slučaju offline režima, komentar će biti vidljiv samo korisniku koji ga je dodao, a ostalima tek nakon što korisnik bude online, da bi se odradila odgovarajuća sinhronizacija. Slika 26 – Slika preuzeta iz galerije

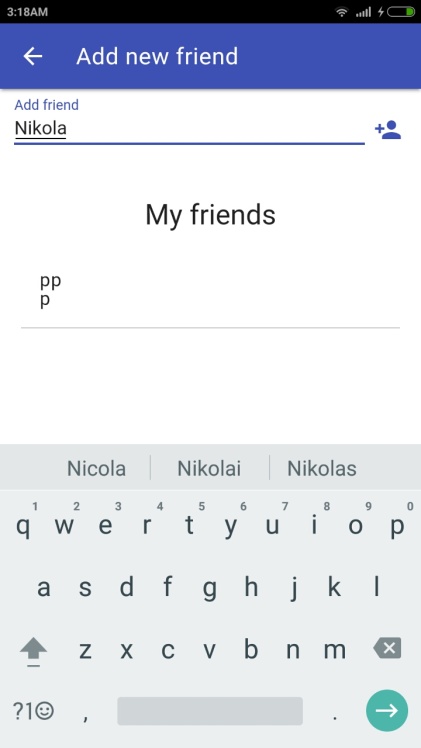


Sledeća funkcionalnost koja će biti obradjena je mogućnost preuzimanja mape nekih od ponudjenih gradova ili pokretanja iste. Ovo možemo odraditi odabirom opcije ,,My Maps” iz slide drawer-a na slici 19 i kao rezultat otvoriće nam se ekran kao na slici 28. Klikom na dugme ,,Open map” (slika 28) iz liste dostupnih gradova otvoriće se mapa izabranog grada dok se klikom na dugme ,,Download new map” korisniku se otvara lista gradova koji su dostupni za prezimanje sa servera (slika 29).Prilikom preuzimanja neke od mapa prikazaće mu se ekran kao na slici 30, a u pozadini će sa servera biti preuzeta baza za odgovarajući grad, a potom smeštena kod korisnika lokalno. U ovom trenutku pored mape preuzima se i sav sadržaj koji je neophodan da bi mogle zanemenitosti, komentari i planovi da Slika 27 – Komentari se prikažu korisniku u offline režimu rada aplikacije.

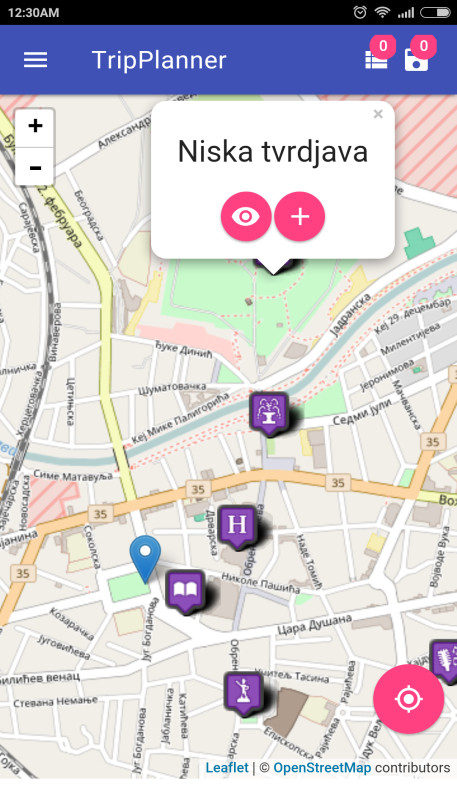
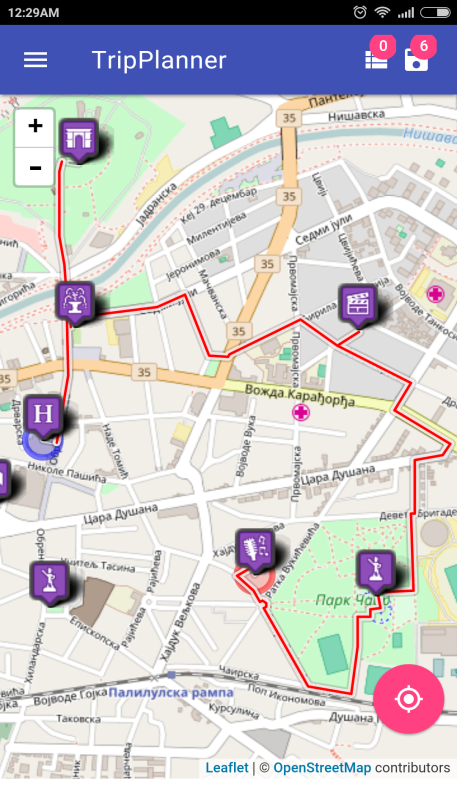
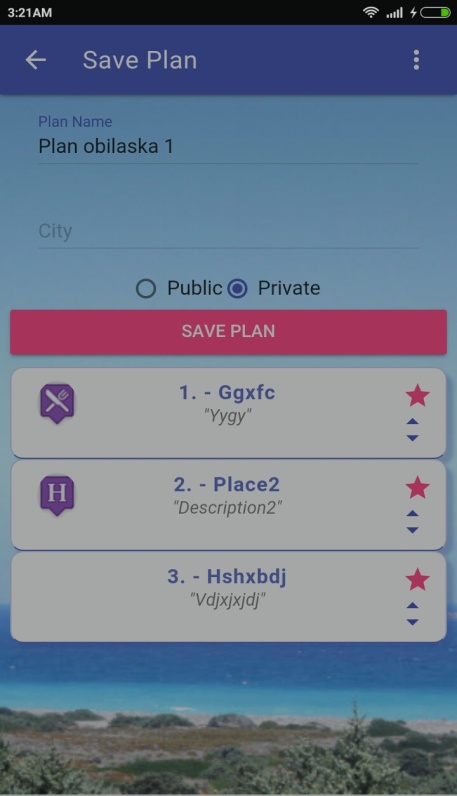
Slika 28 – Lista preuzetih mapa Slika 29 – Lista dostupnih mapa Slika 30– preuzimanje mape

TripPlanner aplikacija takođe nudi korisniku i da medju postojećim korisnicima izabere ko želi da mu bude prijatelj (Slika 31). Koncept prijatelja u ovoj aplikaciji nam omogućava da imamo bolji uvid u planove putovanja ljudi koji su nam zanimljivi, a koji pritom žele da sa drugima podele svoje planove.

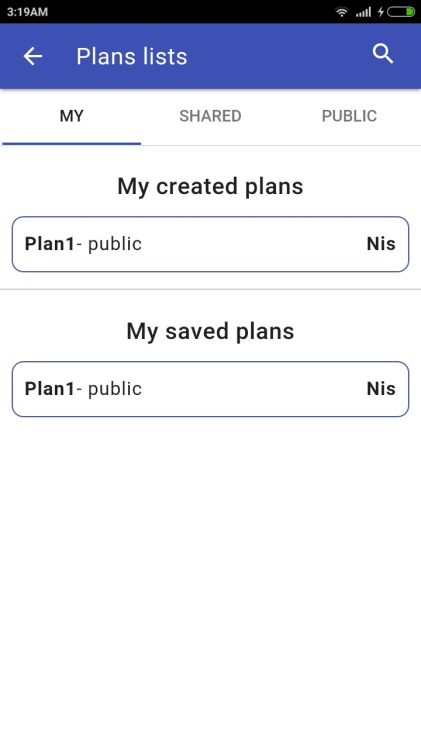
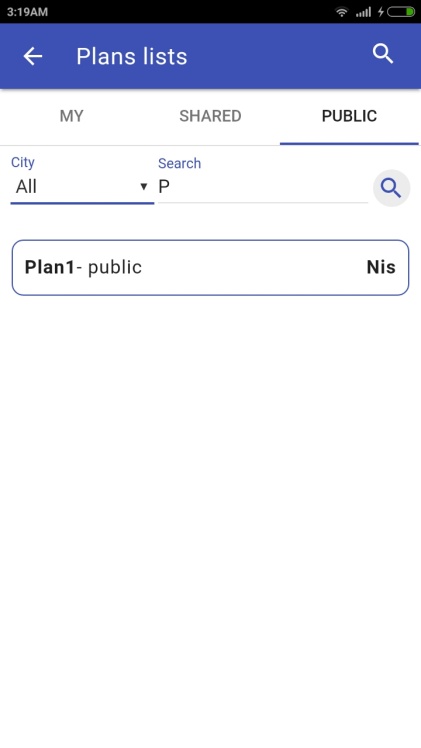


Slika 31 – dodavanje prijatelja

Ovde dolazimo i do pojma planova, po čemu je ova aplikacija i dobila naziv. Ideja sa planovima je da se omogući korisniku da na mapi obeleži znamenitosti koje želi da obidje (slika 32) i da mu se zatim iscrta jedna od mogućih ruta kojom bi mogao da obidje znamenitosti sa svog plana (slika 33). Na slici 33 takođe možemo da vidimo u gornjem desnom uglu ikonu sa brojem mesta koji smo dodali u plan koji kreiramo. Klikom na tu ikonu korisnik može da promeni redosled kojim će da obilazi znamenitosti, zatim da unese naziv plana i grad za koji je napravljen, kao i da ga označi da li će biti vidljiv samo njemu (private) ili svim korisnicima (public) i nakon što proceni da je plan konačan može da ga snimi klikom na dugme Save plan (slika 34).

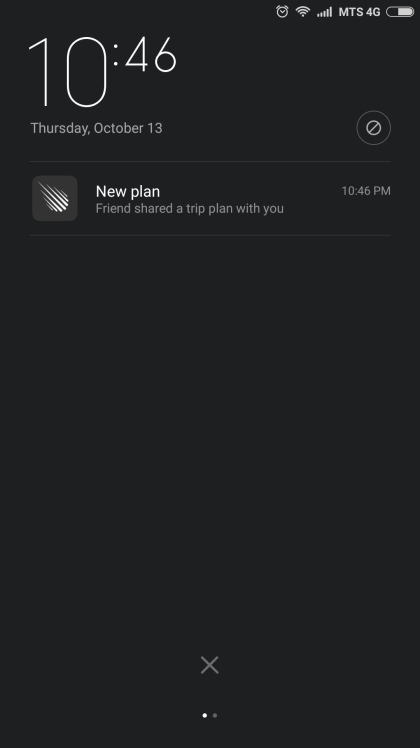
   Slika 32 – Dodavanje u plan Slika 33 – Prikaz plana na mapi Slika 34 – Snimanje plana

U okviru ove aplikacije razlikujemo tri tipa plana: My plans, shared plans i public plans. U My plans spadaju svi planovi koje smo mi kreirali i svi planovi naših prijatelja koje smo snimili kod sebe (slika 35). Grupu shared planova čine planovi naših prijatelja koje su oni podelili sa drugim prijateljima . Poslednju grupu čine public planovi gde su smešteni planovi svih korisnika koji su javno dostupni svima i prikazani su na slici 36. Ova grupa ima mogućnost i pretraživanja planova po gradu, kao i po nazivu.

Slika 35 – My plans Slika 36 – Public plans

U okviru TripPlanner aplikacije postoji još jedna funkcionalnost, a to su notifikacije koje stižu korisniku u slučaju kada neko od njegovih prijatelja podeli neki novi plan sa ostalim prijateljima. Ovaj dogadjaj prikazan je na slici 37.

Treba napomenuti da sve funkcionalnosti koje su prikazane su odrađene tako da efekat koji se postiže i sam dizajn budu identični i u offline i u online režimu. Što se same implementacije ova dva dela tiče, ona se značajnije razlikuju, ali o svemu ovome više reči u dodatku A koje se bavi implentacijom same aplikacije.

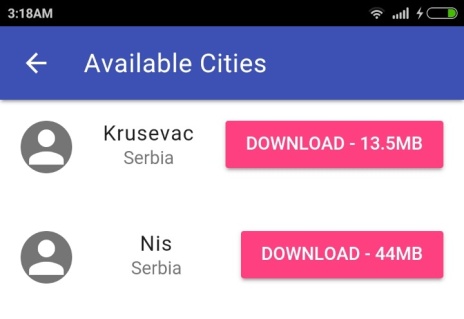
Slika 37 – Notifikacija.

# 5. Zaključak

Nakon svih pređenih poglavlja u kojima smo se upoznali sa arhitekturom TripPlanner aplikacije, njenim funkcionalnostima, kodom i tehnologijama koje su korišćene možemo zaključiti da su se tehnologije kao što su Meteor i Cordova jako dobro pokazale i da su nam omogućile da na brz i efikasan način kreiramo hibridnu mobilnu aplikaciju koja će pritom imati brojne funkcionalnosti. Mehanizam reaktivnosti i online sinhronizacije podataka koji nam nudi Meteor su dodatno ubrzale izradu ove aplikacije i pritom su doprinele efikasnosti I pouzdanosti same aplikacije. Cordovini pluginovi su omogućili da pristupamo nativnim karakteristikama uređaja i da samim tim ona bude dosta funkcionalnija, a opcije kao što su korišćenje kamere, notifikacija, preuzimanje mapa sa servera ujedno doprinose da sama aplikacija bude bolje prihvaćena od strane korisnika. Kao dobar izbor su se pokazale I baze koje su korišćene za skladištenje podataka, jer njihovom kombinacijom omogućena je perzistentnost podataka u online i offline režimu rada. Offline režim je ono po čemu se ova aplikacija razlikuje od mnogih drugih iz ove oblasti, pa je samim tim ovaj režim zahtevao dosta više pažnje prilikom izrade I sinhronizacije podataka. TripPlanner aplikacija nudi korisnicima početni sadržaj, kao što su prikazi postojećih znamenitosti, ali takođe im omogućava da I sami doprinose sadržaju proširavanjem postojećeg I dodavanjem novog. Korisnici imaju uvid u komentare drugih korisnika, kreiraju planove I da ih dele sa drugim, kako bi razmenjivali iskustva. Krajnji cilj ove aplikacije jeste da korisnici u svako doba mogu da koriste TripPlanner aplikaciju prilikom planiranja svojih putovanja i na samim putovanjima, a da pritom ne zavise od internet konekcije.

# Dodatak A: Implementacija karakterističnih delova koda

## A.1 – Šabloni

TripPlanner aplikacija, kao što je već objašnjeno se sastoji iz klijentskog i serverskog dela. Na klijentskom delu posebno mesto zauzimaju šabloni (templejti) kojima definišemo blokove HTML koda, koji potom mogu biti iznova korišćeni koliko god je nama to potrebno. Koncept templejta je u potpunosti korišćen u TripPlanner aplikaciji u izradi dizajna aplikacije. Da bismo ih bolje sagledali, kao i sve njihove mogućnosti, posmatraćemo ekran sa slike 38 koji prikazuje dostupne mape , tačnije kod koji stoji u pozadini. Slika 38 –Dostupne mape

<template name="MapsListLayout">

<div class="mdl-layout mdl-layout mdl-layout--fixed-header">

<header class="mdl-layout\_\_header">

<div class="mdl-layout\_\_header-row js-my-fixed-header">

<i class="material-icons js-my-back-arrow">arrow\_back</i>

<span class="mdl-layout-title">Available Cities</span>

<div class="mdl-layout-spacer"></div>

</div>

</header>

<main class="mdl-layout\_\_content">

<div class="page-content">

<ul class="maps-list mdl-list mdl-js-ripple-effect">

{{#if Template.subscriptionsReady}}

{{#each maps}}

{{> MapsListItemLayout}}

{{/each}}

{{else}}

{{> spinner}}

{{/if}}

</ul>

</div>

</main>

</div>

</template>

Kod 11 - Iscrtavanje dostupnih mapa

Header sekcija iz Koda 11 će nam omogućiti iscrtavanje hedera strane, na kojem će biti prikazano: Available cities i strelica za povratak na prethodnu stranu (Slika 38), međutim, nama to ovde nije od interesa. U main sekciji koja sledi nakon headera nalazi se deo koda:

{{#if Template.subscriptionsReady}}

{{#each maps}}

{{> MapsListItemLayout}}

{{/each}}

koji nam kaže da, ukoliko su objekti (mape) na koje smo se subscribe-ovali (Kod 12) spremni potrebno je iscrtati odgovarajući HTML kod za svaku od njih. Evo kako je to uradjeno:

Template.MapsListLayout.onCreated(function(){

this.subscribe('maps');

});

Kod 12 – Subscribe-ovali smo se na mape

Zatim sa {{#each maps}} pozivamo helper da nam vrati sve mape na koje smo se prethodno subscribe-ovali, pri čemu to izgleda ovako:

Template.MapsListLayout.helpers({

maps: function(){

return Maps.find();

}

});

Kod 13 – Helper vraća sve mape koje smo subscribe-ovanjem dobili sa servera

Nakon što dobijemo od helpera mape, potrebno je svaki taj objekat iscrtati a za njihovo iscrtavanje koristiće se template ,,MapsListItemLayout”. Možemo uočiti da je kod čitljiv i da nam treba mnogo manje linija koda, nego što je to bio slučaj bez Meteora. Kada smo objašnjavali templejte, spomenuli smo da pored helpera postoje i eventi, koje je moguće obradjivati. U nastavku sledi primer koji pokazuje šta će se desiti ukoliko kliknemo na strelicu sa slike 39:

Template.MapsListLayout.events({

'click .js-my-back-arrow': function(){

FlowRouter.go("downloadedmaps");

},

});

Kod 14 – Obrada event dogadjaja

## A.2 – FlowRouting

Prilikom obrade ovog dogadjaja, možemo uočiti da se poziva go funkcija FlowRouter objekta. Ovde dolazimo i do koncepta rutiranja koje postoji u Meteoru. Ovo je jedan veoma jednostavni ruter koji se koristi za rutiranja klijentskih strana aplikacije. Naime sledeći kod: FlowRouter.go("downloadedmaps") će izvršiti rutiranje sa postojeće stranice na rutu sa nazivom: downloadedmaps. Sledi prikaz kako se definiše navedena ruta:

FlowRouter.route('/downloadedmaps', {

name: 'downloadedmaps',

action(){

BlazeLayout.render('MainLayout', {main: 'DownloadedMapsLayout'});

}

});

Kod 15 – Obrada event dogadjaja

Ovim pozivom zapravo će nam se izrenderovati MainLayout stranica, koja će dinamički u svom main elementu da učita DownloadedMapsLayout. Da bi ovo bilo jednostavnije za razumevanje, pogledaćemo i MainLayout stranicu:

<template name="MainLayout">

// ovde može da dodje kod koji je zajednički za više strana

{{> Template.dynamic template=main}} // ovde dinamički učitavamo kod koji se

// razlikuje od strane do strane

</template>

Kod 16 – Dinamičko učitavanje elemenata.

Korišćenje ovakvih strana omogućava nam da jedan isti kod iskoristimo više puta (reuse koda), ali da samim tim i redukujemo broj linija koda samog projekta.

## A.3 – Publish/subscribe pattern

U nastavku rada sledi primer već puno puta spominjanog publish / subscribe patterna. Ovaj pattern omogućava klijentu da se pretplati samo na neophodne podatke koje server publikuje. Ovim se teži da se izbegne da se klijentu sa servera pošalju čitava kolekcije podataka. Sledi par primer-a:   
 Meteor.publish('global-places', function(){

return Places.find({limit: 10});

});

Kod 17 – Server publishuje samo prvih 10 znamenitosti

Meteor.publish('near-places', function(data){

if(data) {

return Places.find({ loc: { $geoWithin: { $centerSphere: [ [data[0].lng,data[0].lat] , data[1] / 6378.15214 ] } } });

}

});

Kod 18 – Server publishuje samo objekte koji se nalaze blizini zadate tačke

Primeru Koda 17 vidimo da server neće vratiti klijentu sve objekte, već samo prvih 10 znamenitosti, dok u Kodu 18, server vraća samo objekte koji se nalaze unutar kruga čiji je prečnik zadat, a centar kruga je poslat sa klijenta u obliku data parametra. Da bi klijent mogao da prihvati ove podatke neophodno je da se subscrubuje na njih, uz pomoc svega jedne linije koda:

this.subscribe('near-places', [Session.get('location'), Session.get('radius')]);

Kod 19 – Klijent se subscribe-uje na podatke

Ovi podaci koje klijent dobije od servera smeštaju se u MiniMongo bazu, tačnije u kolekciju koja ima istim ime kao i kolekcija na serveru odakle su pokupljeni podaci. Podacima iz Minimongo baze se zatim pristupa na isti način kao što bismo to uradili na serveru, pa ako želimo da uzmemo sve objekte iz kolekcije Places na klijentu, dovoljna je linija koda: Places.find(); U Minimongu možemo da čuvamo i kolekcije koje ne treba da postoje pod tim imenom na serveru, to je na primer kolekcija koja čuva znamenitosti koje treba da se proslede serveru radi sinhronizacije. Ovakve kolekcije treba da budu neimenovane.

## A.4 – Postavljanje mape

S obzirom da su mape neizostavan deo ove aplikacije, sledi kod postavljanja mape u online režimu, sa propratnim razlikama koje postoje kod postavljanja u offline režimu:

if(!Meteor.isCordova || navigator.onLine){ //ako nije cordova ili je online

//uzimamo tilove sa osm servera, i objekte iz glavne mongo baze

this.autorun(function() {

template.subscribe('places-within-box', Session.get("map-bounds"), function(){

//Sesija je reaktivni izvor, koji se nalazi u reaktivnom izračunavanju

layerGroupMarkers = new L.LayerGroup();

setPlaces();

});

});

map = L.map('map', {

//center: new L.LatLng(43.5780017910967, 21.3275634837577),

minZoom:0,

maxZoom:18,

maxNativeZoom:18

});

var layerInstance = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {

attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors',

minZoom:0,

maxZoom:18,

}).addTo(map);

}

Kod 20 - Postavljanje mape u online režimu

U Kodu 20 je prikazano kako se mapa prikazuje korisniku u online režimu, najpre se koristi autorun funkcija koja predstavlja reaktivno izračunavanje. Unutar nje se nalazi reaktivni izvor (sesija) čija vrednost kada se promeni bilo gde u programu, prouzrokavaće da se funkcija autorun opet izvrši. Ideja sa ovom funkcijom je da čim se registruje da je korisnik hteo da pomeri mapu, da mu se odmah pribave novi objekti i prikažu. Ovo je dobar primer gde se može videti koliko je jednostavno implementirati reaktivnost kod Meteor aplikacija. Nakon toga se instancira mapa, ali da bi ona mogla da se prikaže, neophodno je najpre pribaviti neophodne tile-ove. Uz pomoć Leafleta proslediće se zahtev provider-u da mu obezbedi tile-ove, a zatim će njih postaviti kao mapu. U ovom slučaju za provider-a je korišćen OpenStreetMap, dok će se kod offline režima tile-ovi preuzeti lokalno iz odgovarajuće baze.

## A.5 - Iscrtavanje ruta na mapi

U okviru TripPlanner aplikacije takođe postoji i mogućnost iscrtavanja plana u vidu rute na mapi. U online slučaju se kontaktira mapbox server na osnovu postojećih place-ova, dok u offline režimu, sačuvani plan učitamo iz JSONa, a zatim sa funkcijom polyline iscrtamo rutu. Za to je zadužen kod koji se nalazi u nastavku:

if (navigator.onLine){

control = L.Routing.control({

waypoints: latlngs,

router: L.Routing.mapbox('pk.eyJ1IjoicGV0cmljODc4IiwiYSI6ImNpdHhkamRudDAwMzkyeXAxa3MyeTV5aWwifQ.EKdOGxKq8BzE\_5IYtXLPBQ',

{

profile: 'mapbox/walking'

}),

show: false,

createMarker: function() {}

}).addTo(map);

}

else if(Meteor.isCordova){

if(map)

polyline = L.polyline(latlngs, {color: 'red'}).addTo(map);

}

Kod 21- Iscrtavanje plana na mapi u vidu ruta

## A.6 – SQLite plugin

S obzirom da je offline deo TripPlanner aplikacije podjednako bitan kao i online deo, prilikom izrade aplikacije, vodilo se računa da sve funkcionalnosti budu implementirane za oba slučaja. Ovo je prouzrokovalo uvođenje nekih novih tehnologija pošto postojeće nisu bile dovoljne. Ovde je reč o uvođenju SQLite baze podataka za skladištenje podataka. Naime, Minimongo ne perzistira podatke, pa je za tu svrhu uvedena SQLite baza. Prilikom preuzimanja mape za offline režim neophodno je sve objekte za tu mapu, kao i komentare o objektima sačuvati u SQLite bazi, kako bi nam oni bili dostupni po gašenju konekcije. Takođe prilikom rada u offline režimu moguće je dodavati nove objekte i nove komentare, pa je SQLite baza korišćena i u te svrhe, pri čemu su svi ti podaci čuvani u posebnim tabelama: pending-places i pending-comments, a po prelasku u online režim vršila se sinhronizacija podataka sa Mongo bazom na serveru.

## A.7 - Metode na serveru

Što se tiče serverskog dela aplikacije, treba spomenuti da na serveru postoje metode. To su funkcije koje se pozivaju sa klijenta kada je potrebno uraditi neku operaciju (insert, update, delete) nad Mongo bazom i čuvaju se u main.js fajlu na serveru.

Meteor.methods({

insertPlace: function(data){

Places.insert({

'name': data[0],

'description': data[1],

'type': data[2],

'loc': { 'type':"Point",'coordinates': [data[3].lng, data[3].lat] }, //latlng

'user\_id': data[4],

'timestamp': new Date(),

'city': data[5]

});

},

insertComment: function(data){

Comments.insert({

'title': data[0],

'content': data[1],

'grade': data[2],

'place\_id': data[3],

'date': new Date(),

'author': data[4],

'user\_id': data[5],

'image1': data[6],

'image2': data[7],

'image3': data[8]

});

});

Kod 22 – Primer metoda na serveru

U primeru koda koji je prikazan (Kod 22) nalaze se dve metode smeštene na serveru. Prva metoda nam obezbeđuje unos nove znamenitosti koja automatski postaje svima dostupna, a druga unos komentara za određenu znamenitost.

Svi nabrojani primeri služe da nam pokažu da korišćenje Meteora može biti jako korisno i efikasno u izgradnji aplikacija, a prikazani mehanizmi spadaju u grupu novih i modernih i karakteristični su za Meteor, pa su samim tim bili detaljno obrađeni u ovom dodatku.

# 6. Reference

1. <https://play.google.com/store/apps/details?id=com.hubdoo.nistalking&hl=en>
2. Stephan Hochhaus, Manuel Schoebel, “Meteor in Action”, Manning Publications Co, 2016.
3. <http://docs.meteor.com/#/full/>
4. <https://cordova.apache.org/docs/en/latest/>
5. <https://cordova.apache.org/plugins/>
6. Leaflet dokumentacija, <http://leafletjs.com/reference-1.0.0.html>
7. Stanimirović A, „Document-oriented baze podataka“, <https://cs.elfak.ni.ac.rs/nastava/mod/resource/view.php?id=3073>
8. Stanimirović A., „Embedded baze podataka SQLite“, <https://cs.elfak.ni.ac.rs/nastava/mod/resource/view.php?id=3006>
9. HTML5 LocalStorage, <http://www.w3schools.com/html/html5_webstorage.asp>