UDC: 004.42:621.395.721.5 INFO M: str. 11-18

RAZVOJ HIBRIDNIH MOBILNIH APLIKACIJA
DEVELOPMENT OF HYBRID MOBILE APPLICATIONS

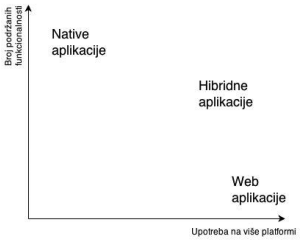
Igor Dujlović, doc. dr Zoran Đurić
Elektrotehnički fakultet Banjaluka, RS, BiH

REZIME: U ovom radu analiziran je način realizacije hibridnih mobilnih aplikacija. Hibridne mobilne aplikacije predstavljaju spoj native i web mobilnih aplikacija, a sastoje se od dijela koji je realizovan pomoću web tehnologija (HTML5, CSS, JavaScript) i dijela koji omogućava pristup funkcionalnostima mobilnog uređaja. Posebna pažnja posvećena je arhitekturi ovih aplikacija. Arhitektura hibridnih mobilnih aplikacija veoma je značajna prilikom realizacije kompleksnih aplikacija, zbog toga što omogućava jednostavnije pisanje i održavanje izvornog koda aplikacije. Podjela aplikacije na module i logičke nivoe omogućava jednostavnije dodavanje novih funkcionalnosti aplikacije, a upotrebom odgovarajućih tehnologija mogu se poboljšati performanse izvršavanja aplikacije. U radu je prikazan jedan način realizacije modularne hibridne mobilne aplikacije, pri čemu su opisane funkcije svakog sloja i tehnologije koje se mogu koristiti pri realizaciji.
KLJUČNE REČI: hibridne mobilne aplikacije, PhoneGap, HTML5
ABSTRACT: This paper analyzes the way of developing of hybrid mobile applications. Hybrid mobile applications are a combination of native and web mobile applications, and consist of a part that is developed using web technologies (HTML5, CSS, JavaScript) and a part that provides access to the functionality of mobile devices. Special attention is dedicated to architecture of these applications. The architecture of hybrid mobile applications is very important in developing of complex applications, because it provides simple development and simple application source code maintaining. Modularization and multi-level architecture provide simple adding of new functionalities, and using relevant technologies applications may have better performance. This paper presents one method of modular hybrid mobile application development, with description of each level and used technologies.
KEY WORDS: hybrid mobile applications, PhoneGap, HTML5

1. UVOD

Mobilne aplikacije mogu se realizovati pomoću različitih tehnologija, u zavisnosti od platforme na kojoj će se aplikacija izvršavati i funkcionalnosti koje treba da budu implementirane. Da bi se jedna aplikacija mogla koristiti na više različitih platformi neophodno je napraviti aplikaciju za svaku pojedinačnu platformu, što predstavlja *native* način razvoja. Drugi način realizacije mobilne aplikacije baziran je isključivo na *web* tehnologijama. Ovako razvijena aplikacija ne može da upotrijebi većinu funkcionalonosti mobilnih uređaja [1].

Hibridne mobilne aplikacije nastale su kao kombinacija *native* i *web* pristupa u razvoju mobilnih aplikacija. Osnovni razlog razvoja hibrnidnih mobilnih aplikacija jeste prevazilaženje ograničenja u pogledu upotrebe funkcionalnosti mobilnih uređaja kod *web* aplikacija, i upotrebe iste aplikacije na više različitih mobilnih platformi što nije moguće u slučaju *native* aplikacija.



Slika 1 – *vrste mobilnih aplikacija*

Mobilne *web* aplikacije ne mogu da koriste većinu funkcionalnosti mobilnih uređaja zbog toga što se aplikacija izvršava u okviru *web* čitača operativnog sistema mobilnog uređaja. Međutim, ista mobilna *web* aplikacija može se koristiti na različitim mobilnim platformama, bez dodatnih prilagođenja. Razlog za to je što je izvorni kod napisan isključivo pomoću *web* tehnologija, tako da se aplikacije mogu izvršavati u *web* čitaču operativnog sistema mobilnog uređaja.

S druge strane, *native* mobilne aplikacije mogu da koriste sve funkcionalnosti određenog uređaja. Osnovni razlog za to je što se ove aplikacije razvijaju u programskom jeziku koji se koristi za razvoj aplikacija za datu platformu. Osim toga, proizvođači mobilnih platformi obezbjeđuju razvojna okruženja i dodatne alate pomoću kojih se aplikacije optimizuju za izvršavanje na određenoj platformi. Iz razloga što se za razvoj aplikacija na određenoj platformi koriste alati i programski jezici specifični za datu platformu, isti izvorni kod aplikacije ne može se koristiti za razvoj aplikacija koje bi se izvršavale na različitim platformama.

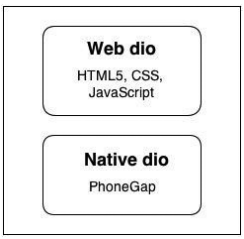
2. HIBRIDNE APLIKACIJE

Hibridne mobilne aplikacije sastoje se iz *native* i *web* dijela, kako bi se iskoristile prednosti *native* i *web* mobilnih aplikacija [1].

Native dio hibridne mobilne aplikacije treba da obezbijedi pristup funkcionalnostima mobilnog uređaja. Za razvoj *native* dijela aplikacije koriste se različite *native* tehnologije za svaku platformu za koju se aplikacija razvija. Kako bi se *native* funkcionalnosti mogle koristiti iz *web* dijela aplikacije, neophodno je napraviti interfejs (obično JavaScript interfejs) koji se koristi za pozivanje *native* funcionalnosti. Postoje

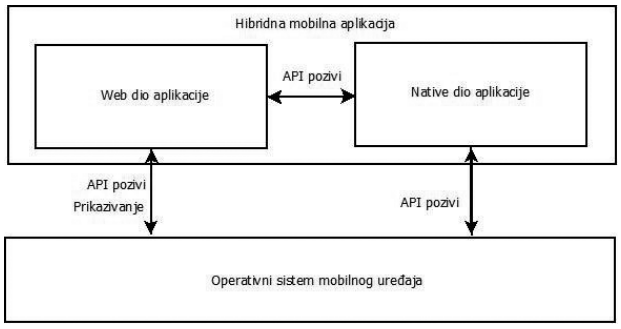
različiti rješenja (komercijalna i besplatna) za razvoj hibridnih aplikacija, poput Appspresso-a, Application Craft-a, Rhodes-a i MoSync-a. Kao primjer rješenja za razvoj hibridnih mobilnih aplikacija, u ovom radu je korišten PhoneGap koji predstavlja jedno od trenutno najpopularnijih rješenja za razvoj hibridnih mobilnih aplikacija [2]. Upotrebom PhoneGap-a nije potrebno razvijati različite implementacije *native* funkcionalnosti za različite mobilne platforme, što doprinosi bržem i jednostavnijem razvoju aplikacije. Osim toga, PhoneGap omogućava da se hibridna mobilna aplikacija zapakuje u format *native* aplikacija, i da se distribuira i koristi na isti način kao i *native* aplikacije.

Web dio hibridne mobilne aplikacije treba da omogući razvoj grafičkog korisničkog interfejsa (ekrana) i razvoj poslovne logike aplikacije, kao i pozive *native* funkcionalnosti mobilnog uređaja. Za razvoj *web* dijela aplikacije koriste se *web* tehnologije: HTML5 (*Hyper Text Markup Language*), CSS3 (*Cascading Style Sheet*), JavaScript, jQuery [3], kao i neka druga rješenja. Principijelna šema hibridne aplikacije prikazana je na slici 2.



Slika 2 – *principijelna šema hibridne mobilne aplikacije*

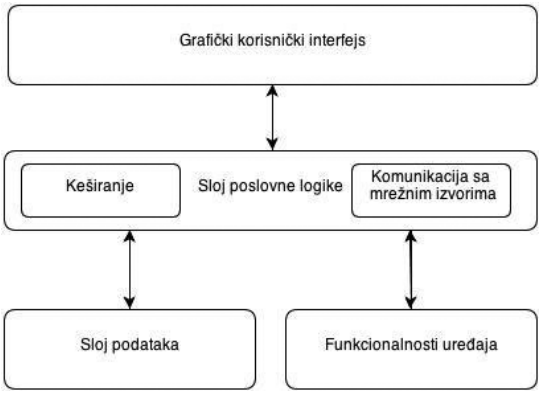
Web dio hibridne mobilne aplikacije se izvršava u *web* čitaču operativnog sistema mobilnog uređaja, dok *native* dio aplikacije ima direktan pristup API-ju (*Application programming interface*) operativnog sistema, što omogućava pristup različitim funkcionalnostima uređaja [4]. Pošto *web* dio hibridne mobilne aplikacije ima direktan pristup *native* dijelu aplikacije, iz *web* dijela aplikacije moguće je koristiti dostupne funkcionalnosti uređaja kao kod *native* aplikacija. Na slici 3 prikazan je način izvršavanja hibridnih mobilnih aplikacija.



Slika 3 – *izvršavanje hibridnih mobilnih aplikacija*

3. LOGIČKA ARHITEKTURA APLIKACIJE

Hibridne mobilne aplikacije sastoje se od nekoliko logičkih nivoa, u zavisnosti od funkcije koju određeni nivo treba da obavlja i tehnologija koje se koriste za realizaciju tog nivoa. Na slici 4 prikazana je logička arhitektura hibridne mobilne aplikacije.



Slika 4 – *arhitektura hibridne mobilne aplikacije*

Sa slike 4 se vidi da se hibridna mobilna aplikacija sastoji od nekoliko slojeva. Sloj grafičkog korisničkog interfejsa koristi se za prikazivanje sadržaja, kao i za upravljanje sadržajem i funkcionalnostima aplikacije. Sloj za izvršavanje poslovne logike omogućava obradu podataka i pripremu za prikazivanje ili upravljanje podacima, kao i obezbjeđivanje dodatnih funkcionalnosti koje su potrebne pri korištenju aplikacije. Ovaj sloj može sadržavati dodatne funkcionalnosti za komunikaciju sa udaljenim izvorima i keširanje podataka, što je bitno za poboljšanje performansi aplikacije. Sloj podataka omogućava pohranjivanje podataka na uređaju, što je posebno bitno prilikom upotrebe aplikacije u slučaju kada nije moguće koristiti mrežnu komunikaciju. Svi prethodno navedeni slojevi pripadaju *web* dijelu aplikacije, jer se za realizaciju navedenih slojeva koriste *web* tehnologije. Sloj koji se odnosi na obezbjeđivanje upotrebe funkcionalnosti samog mobilnog uređaja predstavlja *native* dio aplikacije. U nastavku rada su prikazani detalji realizacije svakog od navedenih slojeva.

I. SLOJ GRAFIČKOG KORISNIČKOG INTERFEJSA

Grafički korisnički interfejs predstavlja jedan od najvažnijih dijelova svake mobilne aplikacije, zbog toga što se korisnici upotrebom ovog dijela aplikacije upoznaju sa funkcionalnostima cijele aplikacije. Osim toga, podaci, opcije i druge funkcionalnosti prikazuju se upravo preko grafičkog korisničkog interfejsa (tj. putem korisničkih ekrana), pa je prilikom razvoja mobilne aplikacije potrebno napraviti što intuitivnije i jednostavnije ekrane. Za razvoj korisničkih ekrana u hibridnim mobilnim aplikacijama obično se koriste HTML5, CSS3 i drugi jezici, alati i okruženja (eng. *frameworks*), poput jQuery Mobile [5].

HTML5

HTML5 se prilikom razvoja hibridnih mobilnih aplikacija koristi za definisanje strukture korisničkog interfejsa. Pomoću strukturnih HTML5 elemenata i brojnih kontrola koje se nalaze

u okviru HTML5, moguće je napraviti interfejse koji se funkcionalno mnogo ne razlikuju od interfejsa koji su realizovani *native* tehnologijama. Prikazivanje HTML sadržaja vrši se pomoću *web* čitača mobilnog uređaja, tako da performanse prikazivanja mogu biti lošije u odnosu na *native* implementacije. Drugi potencijalni problem prilikom razvoja grafičkog korisničkog interfejsa pomoću HTML5 je činjenica da u različitim *web* čitačima, a samim tim i na različim platformama, sve funkcionalnosti nisu podržane na isti način, ili uopšte nisu podržane [6]. Ovaj problem je prisutan i pri razvoju *web* aplikacija. Zbog navedene činjenice, moguće je da se naruši univerzalnost primjene istog izvornog koda aplikacija za upotrebu na različitim platformama. Osim toga, određene *native* funkcionalnosti, vezane za rad sa grafičkim korisničkim interfejsom, ali i karakterističan *native* izgled određenih grafičkih elemenata nije moguće realizovati pomoću HTML5 i povezanih tehnologija. Ipak, aplikacije realizovane pomoću *web* tehnologija često imaju isti dizajn na svim platformama. Ukoliko je potrebno obezbijediti izgled aplikacije koji je sličan izgledu *native* aplikacija, neophodno je izvršiti prilagođenja kroz CSS ili HTML za svaku platformu na kojoj se aplikacija koristi.

Za realizaciju grafičkog korisničkog interfejsa najčešće se koriste elementi *div* i *span*, na sličan način kao kod izrade *web* stranica. Navedene elemente je potrebno formatirati tako da se ostvari željeni raspored komponenti grafičkog korisničkog interfejsa, a njihov izgled i izgled kompletnog grafičkog korisničkog interfejsa definiše se pomoću CSS-a. Osim što se koristi za realizaciju grafičkih korisničkih interfejsa, HTML5 omogućava i upotrebu dodatnih elemenata i funkcionalnosti, kao što je, na primjer, rad sa multimedijalnim sadržajem [6].

CSS

Pomoću CSS-a definiše se izgled HTML elemenata na ekranu, kao i lokacije na kojima će se određeni elementi nalaziti. Pristup HTML elementima, i definisanje izgleda vrši se na osnovu vrste elementa, ID vrijednosti elementa ili naziva CSS klase koja će biti primjenjena na dati element. Osim definisanja statičkih vizuelnih efekata, CSS3 (kao aktuelni CSS standard) omogućava kreiranje animacija, transformacija, efekata prelaza preko elementa, dodavanja vizuelnih efekata na ivice elemenata, slova i brojnih drugih efekata. Prilikom definisanja izgleda mobilne aplikacije, veoma je bitno prilagoditi dimenzije i raspored elemenata kako bi se aplikacija mogla korektno prikazivati na različitim rezolucijama mobilnih uređaja. Takođe, veoma je bitno da aplikacija bude jednostavna za korištenje, i da navigacija pomoću menija ili tipki bude jednostavna i intuitivna. Pri izradi menija, često se koriste HTML liste, koje se dizajniraju pomoću CSS-a. Bitno je primijetiti da određeni elementi CSS-a nisu podržani na svim platformama, što može dovesti do poteškoća pri razvoju aplikacije.

jQuery Mobile

Kako bi se pojednostavio i ubrzao razvoj grafičkih korisničkih interfejsa, moguće je koristiti gotova rješenja među kojima je veoma popularan jQuery Mobile radni okvir. Ovaj okvir za razvoj grafičkog korisničkog interfejsa omogućava dodavanje HTML elemenata na ekran, dok se podešavanje izgleda definiše dodavanjem odgovarajućih atributa. jQuery Mobile [5] omogućava upotrebu šablona (eng. *template*) koji definiše strukturu jednog ekrana, koji je u osnovi HTML stranica. Šablon se obično sastoji od zaglavlja (*header*), prostora za sadržaj ekrana (*content*), i podnožja stranice (*footer*). Sadržaj svakog od navedenih dijelova se može mijenjati. Prilikom izrade aplikacija koje sadrže različite korisničke ekrane, poželjno je svaki ekran definisati u posebnom HTML fajlu, kako bi bilo jednostavnije realizovati potrebne funkcionalnosti, ali i izgled svakog ekrana. Svaka jQuery Mobile stranica je po svojoj prirodi *div* element, kome je dodijeljena uloga stranice (*data-role=page*). Navigacija između različitih stranica vrši se pomoću ugrađene funkcije za promjenu stranice na osnovu ID svake stranice, kojim se identifikuje odredišna stranica. Drugi način navigacije je upotrebom linkova. Prilikom promjene stranice moguće je koristiti neki od ugrađenih efekata prelaza, čime se izbjegava ručno kreiranje prelaza u CSS-u. jQuery Mobile sadrži veliki broj komponenata koje su dizajnirane i prilagođene upotrebi u mobilnim aplikacijama. Komponente sadrže atribute na osnovu kojih se aktiviraju dodatne funkcionalnosti komponente. Tako se, na primjer, definisanjem atributa za filtriranje u listi dobija mogućnost pretraživanja prikazanih podataka u listi, bez pisanja dodatne programske logike koja bi bila neophodna u slučaju da se piše standardni HTML kod. Za definisanje dizajna aplikacije mogu se koristiti ugrađene teme, mogu se izvršiti prilagođenja izmjenom CSS-a ili kreirati nova tema. Osim definisanja komponenti, jQuery Mobile omogućava definisanje događaja koji će se aktivirati prilikom interakcije sa komponentama. Na slici 5 prikazan je primjer izvornog koda jQuery Mobile stranice i povezivanja sa drugom stranicom.

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

    <div data-theme="c" data-role="header" data-position="fixed">
        <h1>Naslov</h1>
    </div>

    <div data-role="content">
        <ul id="list" data-role="listview" data-filter="true" data-inset="true">
        </ul>
        <a href="index.html" data-role="button">Statičko povezivanje</a>
        <a href="#" id="btnNext" data-role="button">Dinamičko povezivanje</a>
    </div>

    <div data-theme="c" data-role="footer">
      <h4 id="message"></h4>
    </div>

</div>

//registrovanje događaja
$( "#btnNext" ).on( "click", function(event, ui) {
  ...
  //prelazak na drugu stranicu uz efekat tranzicije
  $.mobile.changePage( "secondPage.html", { transition: "slideup"} );
});
```

Slika 5 – *primjer izvornog koda jQuery Mobile stranice*

Upotrebom gotovih rješenja za razvoj grafičkog korisničkog interfejsa, poput jQuery Mobile-a, olakšava se i ubrzava razvoj. Isto tako, smanjuje se mogućnost, ili se otežava kreiranje proizvoljnih kontrola ili vizuelnih efekata. Takođe, bitno je primijetiti da, u slučaju kada se na ekranu nalazi mnogo elemenata ili da se na ekranu nalaze elementi sa većom količi-

nom podataka (na primjer, liste sa mnogo elemenata), prikazivanje komponenata može da naruši performanse aplikacije.

II. SLOJ POSLOVNE LOGIKE

Sloj poslovne logike u hibridnim mobilnim aplikacijama omogućava obradu podataka koje aplikacija koristi, njihov prikaz, čuvanje i prenos. Osim toga, sloj poslovne logike često je zadužen za upravljanje različitim procesima ili operacijama u aplikaciji. Neki od dodatnih procesa mogu se odnositi na upravljanje privilegijama za pristup pojedinim dijelovima aplikacije, poboljšanje performansi, komunikaciju i sinhronizaciju sa drugim izvorima i resursima, upravljanje tokom aktivnosti, kao i druge operacije.

Rad sa podacima i obrada događaja

Sloj poslovne logike može se posmatrati i kao veza između slojeva u kojima se čuvaju podaci, ili se preuzimaju iz proizvoljnog izvora, i sloja na kome se vrši prikaz ili unos podataka. Kada se podaci unesu na grafičkom korisničkom interfejsu – formi, potrebno je izvršiti validaciju i dalju obradu tih podataka. Zbog činjenice da je grafički korisnički interfejs realizovan pomoću HTML-a, podaci se iz DOM strukture preuzimaju pomoću JavaScript funkcija za dobijanje vrijednosti iz HTML elemenata. Elementi za unos podataka najčešće su varijante „input“ elementa. Validacija unesenih podataka vrši se pomoću JavaScript funkcija za provjeru tipa podataka, operatora poređenja ili provjere unosa u odnosu na regularni izraz. Dodatnu obradu podataka moguće je realizovati upotrebom JavaScript-a. Prilikom prikaza podataka iz programskog dijela aplikacije potrebno je izvršiti pripremu sadržaja, što se takođe obavlja JavaScript-om. Pripremljeni podaci se zatim programski ubacuju u DOM strukturu stranice.

```
<input type="text" id="name" />
<input type="text" id="message" />

//JavaScript
//uzimanje vrijednosti iz HTML elementa
var name=document.getElementById("name");
//primjer validacije
if(name!=null && name.length<15){
  ...
  var patt1 = /abc/;  //regularni izraz
  var temp=str.match(patt1);
  ...
}
//postavljanje sadržaja u HTML element
document.getElementById("message").innerHTML = "Korisnik "+name;

//jQuery
var name = $('#name').val();  //uzimanje vrijednosti iz HTML elementa
$('#list').append(html);      //postavljanje sadržaja u HTML element

//obrada click događaja pomoću JavaScript-a
<h1 id="element" onclick="call()">JavaScript događaj</h1>

function call(){
  ...
}

//obrada click događaja pomoću jQuery-ja
<h1 id="element">jQuery događaj</h1>

$( "#element" ).on( "click", function(event, ui) {
  call();
});
```

Slika 6 – *rad sa podacima i obrada događaja*

U aplikaciji je moguće definisati obradu događaja koji se registruju na elementima ekrana. Obrada događaja obavlja se u definisanim funkcijama u kodu aplikacije. Osim upotrebe JavaScript-a, za realizaciju sloja poslovne logike moguće je koristiti i jQuery, a moguće ih je i kombinovati. Razlika između njih, u pogledu obrade događaja je u tome što se kod JavaScript-a događaji koji se vezuju za HTML elemente definišu u samim HTML elementima. Kod jQuery-a događaji se povezuju sa elementima u pozadini aplikacije. Dobijanje i prikazivanje podataka se vrši na sličan način, upotrebom odgovarajućih funkcija. Na slici 6 prikazan je primjer validacije unesenih podataka i prikaz povezivanja događaja na elementima pomoću JavaScript-a i jQuery-ja.

Organizacija programskog koda aplikacije

Organizovanje koda aplikacije može se obaviti na nekoliko načina. Jedan od načina je definisanje kompletne poslovne logike u okviru jednog JavaScript fajla. Ovaj način organizovanja je pogodan za manje aplikacije, dok je tako organizovan programski kod često nepregledan kod većih aplikacija. Drugi način je definisanje koda u zasebne JavaScript fajlove, za svaki pojedinačni grafički interfejs. U ovom slučaju poboljšava se preglednost koda, što kao posljedicu ima jednostavnije održavanje programskog koda i dodavanje novih funkcionalnosti aplikacije. Programski kod aplikacije moguće je organizovati i u okviru HTML fajlova, ali ovakav pristup može dovesti do problema u održavanju aplikacije, zbog povećane nepreglednosti koda. Ukoliko se programski kod organizuje na način da se kod za izvršenje poslovne logike definiše u jednom fajlu, a kod odgovarajućih korisničkih interfejsa definiše u drugim fajlovima, postiže se *view-controller* organizacija. Ovakav način organizacije je posebno pogodan ukoliko se aplikacija razvija pomoću jQuery programskog okvira.

Razmjena podataka preko mreže

Za slanje ili dobijanje podataka sa udaljenih izvora , kao što su *web* servisi, pogodno je koristiti odgovarajuće jQuery funkcije. Podaci koji se razmjenjuju su najčešće u XML (*eXtensible Markup Language*) ili JSON (*JavaScript Object Notation*) formatu. Za prenos podataka često se koristi jQuery *ajax*() funkcija. Pomoću ove funkcije moguće je realizovati POST i GET zahtjeve, uz opcioni prenos podataka, mogućnost keširanja, definisanja vrste podataka, podešavanja zaglavlja zahtjeva i podešavanja dodatnih parametara. Ukoliko se zahtjev uspješno izvrši, pozvaće se odgovarajuća *callback* funkcija, koja će prihvatiti podatke pristigle sa serverske strane. Ukoliko dođe do greške pozvaće se odgovarajuća funkcija za obradu greške. Upravo ove dvije *callback* funkcije omogućavaju jednostavnu manipulaciju prenesenim sadržajem. Osim *ajax* funkcije, postoje i *get()* i *post()* funkcije koje omogućavaju pojedinačne zahtjeve. Prilikom razvoja aplikacija, moguća su ograničenja u uspostavljanju komunikacije između različitih domena. Ova ograničenja se javljaju zbog *cross domain* politike koja je definisana na većini *web* čitača mobilnih ure-

đaja koji se koriste za obradu *web* dijela hibridne aplikacije. Navedena ograničenja mogu se prevazići upotrebom jQuery funkcija. Osim upotrebe jQuery-a razmjenu podataka između aplikacije i udaljenih izvora moguće je realizovati pomoću JavaScript zahtjeva. Na slici 7 prikazane su metode koje se mogu koristiti za razmjenu podataka preko mreže.

```
$.ajax( {
  type: ajaxType, //POST ili GET
  timeout: 5000,
  url: destinationUrl,
  data: dataXML,
  contentType: 'application/xml',
  dataType: 'xml',
  ...
  success: function(data) {
     ...
  },
  error: function( jqXHR, textStatus, errorThrown ) {
    ...
  }
});

$.get(
  "test.php",
  function(data) {  //onSuccess metoda
    ...
  },
  "json"
);

$.post(
  "test.php",
  { name: "abc", time: "2"  },  //podaci koji se prenose
  function(data){ //onSuccess metoda
    ...
  },
  "json"
);
```

Slika 7 – *metode za razmjenu podataka preko mreže*

Optimizacija i kreširanje

Mobilne aplikacije treba da budu optimizovane za izvršavanje tako da koriste što manje resursa. Osim toga potrebno je obezbijediti što bolje performanse izvršavanja. Optimizacija poslovne logike aplikacije može se izvrštiti upotrebom optimizovanih programskih struktura za obradu ili pretragu većih količina podataka. Osim toga, svi dodatni fajlovi koje aplikacija koristi poput dodatnih biblioteka mogu se optimizovati komprimovanjem koda, što dovodi do poboljšanja performansi aplikacije. Prikaz kompleksnih grafičkih interfejsa može degradirati performanse aplikacije, pa je korisno omogućiti keširanje DOM strukture, ili prikazivati sadržaj u dijelovima ili na zahtjev. Dodatno poboljšanje vezano za prikaz sadržaja može se postići keširanjem elemenata iz DOM strukture u kodu aplikacije kako bi se izbjeglo često pretraživanje DOM strukture ekrana. Pored toga, moguće je keširati i podatke. Podatke je potrebno keširati da bi se izbjeglo često pretraživanje većih količina podataka, preuzimanje podataka iz mrežnih izvora, fajl sistema ili memorije uređaja. U hibridnim mobilnim aplikacijama postoje tri vrste keširanja, i to:

1. memorijsko keširanje. Ovaj način keširanja podrazumijeva da se podaci smještaju u memoriju aplikacije, kao promjenljive, i da se na taj način izbjegava njihovo često dohvatanje.
2. keširanje pomoću *storage* HTML objekata. Postoje dvije vrste storage objekata: *localstorage* i *sessionstorage*. Podaci sačuvani u *localstorage* objektu su trajno pohranjeni, dok se podaci u *sessionstorage* objektu čuvaju samo u toku sesije. Ovi objekti omogućavaju čuvanje podataka u string formatu u obliku ključ-vrijednost. Prednost ovakvog načina čuvanja podataka je podrška za korištenje na većini mobilnih platformi i jednostavnost upotrebe. Nedostatak predstavlja ograničenje u količini podataka koji se mogu pohraniti, a koje iznosi 5 MB. Ukoliko je podatke potrebno čuvati kao niz, za smještanje podataka može se koristiti JavaScript metoda *JSON.stringify* koja niz pretvara u string. Za dobijanje niza iz *localstorage* objekta moguće je koristiti *JSON.parse* funkciju koja vrši parsiranje pohranjenog stringa u niz sačuvanih objekata. Na slici 8 prikazan je način upotrebe *localStorage* objekta za čuvanje niza podataka i njegovo dohvatanje.
3. keširanje pomoću lokalnih SQLite baza podataka. Na ovaj način može se čuvati veća količina podataka, i omogućiti relaciona struktura. Ovakav način keširanja posebno je pogodan u slučaju da se podaci preuzimaju sa udaljene lokacije (preko mreže). Osnovni nedostatak ovog načina keširanje je što SQLite baze podataka nisu podržane na mnogim platformama.

```
var dataArray = new Array();
...
//smještanje niza podataka u localStorage objekat
localStorage.setItem("dataArrayStorage",  JSON.stringify(dataArray));

//dobijanje niza podataka iz localStorage objekta
var tempArray = JSON.parse(localStorage.getItem("dataArrayStorage"));
```

Slika 8 – *upotreba localStorage objekta za čuvanje niza podataka*

Web workers

Za dodatnu optimizaciju operacija u sloju poslovne logike moguće je koristiti i HTML *web workers* mehanizam. Ovaj mehanizam omogućava izvršavanje JavaScript koda u pozadini aplikacije čime se može postići paralelizacija izvršavanja operacija. Slično podrški za SQLite, i ovaj mehanizam nije podržan na mnogim platformama što može dovesti do ograničenja pri razvoju aplikacija koje treba da se izvršavaju na različitim mobilnim platformama.

Offline mehanizam

Hibridna mobilna aplikacija može koristiti sadržaje i dijelove koji se učitavaju sa mrežnih izvora. Kako bi se preuzimanje sadržaja optimizovalo moguće je koristiti HTML5 *offline* mehanizam kojim se definiše sadržaj koji se ne preuzima sa mreže. Sadržaj koji treba da bude učitan sa fajl sistema definiše se u *manifest* dokumentu koji se uključuje u HTML stranicu aplikacije. Osim toga, moguće je definisati sadržaj koji se svaki put preuzima sa mrežnih izvora, kao i fajlove koji će se koristiti kao zamjenski, ukoliko određeni fajlovi sa mreže nisu dostupni.

III. SLOJ ZA ČUVANJE PODATAKA

Veliki broj aplikacija koristi podatke koji su preuzeti sa mrežnih izvora. Kako bi se aplikacija mogla koristiti i u slučaju da nije moguće ostvariti prenos podataka između aplikacije i proizvoljnih mrežnih izvora (*offline* dostupnost), potrebno je omogućiti pohranjivanje sadržaja na mobilnom uređaju. Osim toga, aplikacije često koriste memoriju mobilnih uređaja za smještanje različitih podataka ili fajlova koji su potrebni za izvršavanje funkcionalnosti koje aplikacija omogućava.

Načini čuvanja podataka

Sloj za čuvanje podataka može omogućiti trajno ili privremeno pohranjivanje proizvoljne količine podataka. Podaci se mogu čuvati na nekoliko načina:

1. pomoću *localstorage* HTML objekta, koji se, kako je to ranije i opisano, može upotrijebiti i za keširanje podataka.
2. upotrebom lokalnih baza podataka. Iako lokalne baze podataka kao dio HTML-a nisu podržane na većini platformi, moguća je njihova upotreba pomoću PhoneGap implementacije.
3. pohranjivanjem podataka na fajl sistem uređaja. Ovaj način je pogodno koristiti u slučaju kada je potrebno čuvati fajlove.

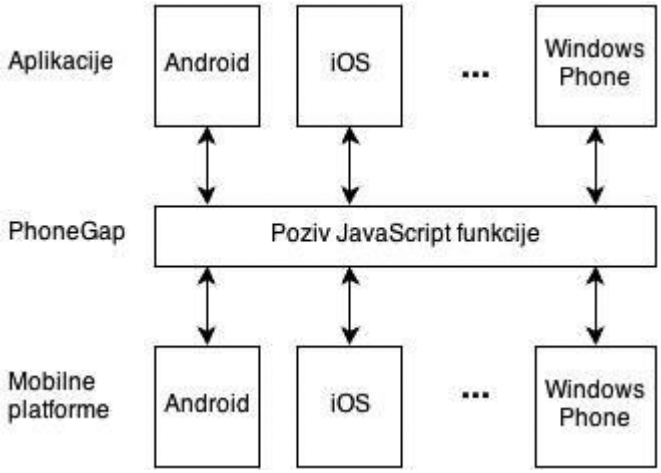
Ukoliko se za čuvanje podataka koriste lokalna baza podataka ili fajl sistem, poželjno je koristiti rješenja koja mogu da realizuju odgovarajuće operacije. Jedno od tih rješenja je PhoneGap, koji implementira sve navedene načine čuvanja podataka. Upotrebom rješenja kao što je PhoneGap olakšava se upravljanje navedenim sistemima i postiže se univerzalnost koda za različite platforme. Razlog za to je što PhoneGap i slična rješenja implementiraju *native* funkcionalnosti ukoliko na određenoj platformi ne postoji njihova odgovarajuća HTML implementacija. Ipak, bitno je znati da se na nekim platformama ne mogu koristiti svi navedi načini smještanja podataka. Tako, na primjer, na Windows Phone platformi nije moguće koristiti SQLite baze podataka, dok je na Android ili iPhone platformi upotreba SQLite baze podataka moguća[1].

IV. SLOJ ZA OBEZBJEĐIVANJE FUNKCIONALNOSTI MOBILNIH UREĐAJA

Svi prethodno navedeni slojevi mogu se realizovati upotrebom *web* tehnologija bez potrebe za korištenjem *native* funkcionalnosti. Međutim, u mobilnim aplikacijama potrebno je obezbijediti upotrebu *native* funkcionalnosti mobilnog uređaja, što se može postići upotrebom PhoneGap programskog okruženja i njemu sličnih rješenja. Sloj za obezbjeđivanje funkcionalnosti mobilnih uređaja predstavlja *native* dio hibridne mobilne aplikacije.

PhoneGap je kontejner koji sadrži *native* biblioteke, JavaScript interfejse i konfiguracione fajlove koji obezbjeđuju kreiranje hibridnih mobilnih aplikacija koje će po svojoj funkcionalnosti biti slične *native* aplikacijama [7]. JavaScript interfejsi omogućavaju da se programski kod aplikacije koji koristi *native* funkcionalnosti uređaja realizuje na gotovo identičan način za sve mobilne platforme. *Native* biblioteke omogućavaju izvršavanje *native* funkcionalnosti koje su pozvane pomoću JavaScript interfejsa. Konfiguracioni fajlovi služe za definisanje parametara potrebnih za izvršenje aplikacije.

Upotreba određene funkcionalnosti vrši se pozivom odgovarajuće JavaScript funkcije koja je definisana u PhoneGap-ovom API-ju. Pozvana JavaScript funkcija pomoću interfejsa pronalazi *native* implementaciju pozvane funkcionalnosti, koja se potom izvršava. *Native* implementacije su različite za svaku platformu, dok je JavaScript interfejs isti, što omogućava da funkcionalnosti imaju identične pozive na svim podržanim platformama [8]. Na slici 9 prikazano je pozivanje *native* funkcionalnosti na različitim mobilnim platformama.



Slika 9 - *pozivanje native funkcionalnosti na različitim mobilnim platformama*

Veliki broj *native* funkcionalnosti za različite mobilne platforme implementiran je u PhoneGap-u. Ukoliko je potrebno koristiti funkcionalnost koja nije implementirana u PhoneGap-u moguće je napisati *plugin* koji funkcioniše na sličan način kao i ugrađene PhoneGap funkcionalnosti. Neke od podržanih funkcionalnosti su upotreba kamere uređaja, geolokacija, rad sa multimedijalnim sadržajem, snimanje audio i video sadržaja, rad sa fajl sistemom i pohranjivanje podataka [9]. Međutim, ugrađene funkcionalnosti nisu jednako podržane na svim platformama, ili uopšte nisu podržane na određenim platformama, što može dovesti do poteškoća pri razvoju aplikacije. To dovodi do potrebe pisanja različitog izvornog koda za ciljne mobilne platforme, kako bi se aplikacija na njima mogla i izvršavati.

Pomoću PhoneGap-a mogu se razvijati aplikacije za skoro sve mobilne platforme, među kojima su najvažnije Android, iOS, Windows Phone, BlackBerry, Palm WebOS i Symbian. Osnovna prednost upotrebe PhoneGap-a za realizaciju sloja za pristup native funkcionalnostima je mogućnost upotrebe istog koda za upotrebu funkcionalnosti uređaja na različitim platformama i podržanost velikog broja native funkcionalnosti.

1 Izvor: http://docs.phonegap.com/en/2.7.0/cordova_storage_storage.md.html#Database

4. MODULARNA ORGANIZACIJA APLIKACIJE

Iako ne postoji precizno definisano pravilo na koji način treba da se organizuje izvorni kod aplikacije, često je veoma korisno podijeliti aplikaciju na više dijelova. Jedan od načina organizacije izvornog koda aplikacije je *view-controller* organizacija ili *model-view-controller* (MVC) organizacija. MVC organizacija omogućava razdvajanje HTML sadržaja koji se koristi za prikaz grafičkih korisničkih interfejsa, odgovarajućih kontrolera koji omogućavaju realizovanje funkcionalnosti koje su vezane za odgovarajući grafički interfejs i sloja podataka. Funkcionalnosti svakog logičkog dijela odvajaju se u posebne fajlove, što omogućava bolju preglednost koda i jednostavnije održavanje. U HTML stranicama uključuju se potrebni JavaScript fajlovi čime se omogućava upotreba funkcionalnosti koje su definisane u ovim fajlovima. Međutim, na ovaj način teško je utvrditi međusobne zavisnosti pojedinih funkcionalnosti što može dovesti do nepotrebnog dupliranja koda. Osim toga, ukoliko se u aplikaciju uključe svi fajlovi, a funkcionalnosti koje su definisane u njima nisu potrebne tokom izvršavanja aplikacije, može doći do smanjenja performansi. Kako bi se navedeni problemi prevazišli može se upotrijebiti gotovo rješenje za organizaciju izvornog koda aplikacija napisanih pomoću *web* tehnologija.

U nastavku rada prikazan je jedan način organizovanja hibridne mobilne aplikacije. Ova organizacija je bazirana na upotrebi RequireJS [10], BackboneJS [11] i jQuery Mobile biblioteka. RequireJS biblioteka omogućava asinhrono učitavanje JavaScript skripti. RequireJS implementira AMD [12] (*Asynchronous Module Definition*) specifikaciju što omogućava asinhrono učitavanje dijelova koda. AMD mehanizam omogućava definisanje modula i međusobnih zavisnosti pojedinih dijelova koda. Ovaj mehanizam omogućava bolje upravljanje kodom aplikacije i poboljšanje performansi. Pomoću BackboneJS biblioteke moguće je definisati modele, kolekcije, poglede i rutere. Modeli predstavljaju dio aplikacije u kome se definišu podaci koji se koriste, kao i funkcije koje se koriste za rad sa definisanim podacima. Kolekcije predstavljaju nizove modela. Pogledi se mogu smatrati kontrolerima koji povezuju HTML stranice sa modelima. U pogledu se definišu funkcionalnosti koje omogućavaju rad sa grafičkim interfejsom i izvršavanje poslovne logike aplikacije. Bitno je naglasiti da funkcionalnosti definisane u pogledima služe kao pozadina HTML stranice. Pogledi se mogu koristiti uz proizvoljan mehanizam za rad sa šablonima. Ruteri služe za povezivanje HTML sadržaja sa odgovarajućim pogledima i događajima obezbjeđujući upotrebu istorije posjećenih stranica i pojednostavljene URL adrese stranica (ekrana). Osim prethodno navedenih biblioteka često je potrebno uključiti i dodatne biblioteke kojima se dopunjuju funkcionalnosti koje nije moguće ostvariti prethodno navedenim bibliotekama.

Primjer strukture hibridne mobilne aplikacije koja je modularno organizovana pomoću prethodno navedenih biblioteka prikazan je na slici 10 [13].

Slika 10 – *primjer strukture modularno organizovane aplikacije*

Početna stranica aplikacije je *index.html* koja sadrži definisane okvire ostalih HTML stranica aplikacije. Ovi okviri predstavljaju prazne *div* elemente sa navedenim *id* atributom (koji identifikuje stranicu) i *data-role="page"* jQuery Mobile atributom. U *script* dijelu *index* stranice obavezno se nalazi deklaracija ulaznog JavaScript fajla u kome se nalaze potrebna podešavanja aplikacije.

```
<script data-main="js/main" src="js/libs/require/require.js"></script>
```

Fajl *main.js* predstavlja konfiguracioni fajl aplikacije u kome se definišu putanje do svih JavaScript fajlova i biblioteka koje se koriste u aplikaciji, njihovo imenovanje i funkcije u kojima se često pozivaju funkcije za inicijalizaciju aplikacije. Ova podešavanja se navode u *require* funkciji. Fajl *router.js* sadrži podatke o svim stranicama koje se nalaze u aplikaciji i omogućava njihovo učitavanje. Ova podešavanja definišu se u *define* funkciji. U *views* folderu nalaze se pogledi za svaku pojedinačnu HTML stranicu, koji u logičkom smislu imaju funkciju kontrolera. Svaki pogled definiše potrebne biblioteke i funkciju koja sadrži objekat koji nasljeđuje *Backbone.view* objekat, a koji predstavlja kreirani kontroler. U nasljeđenom objektu definišu se različiti parametri, među kojima su značajni parametri koji definišu *id* elementa iz *index* stranice koji se popunjava šablonskom stranicom tog pogleda. Postoji više biblioteka koje omogućavaju rad sa šablonima, među kojima se ističe *underscoreJS*. Upotrebom šablona može se izbjeći direktna promjena sadržaja HTML elemenata unaprijed definisanim vrijednostima koje se programski mijenjaju u pogledu stranice. Osim parametara i globalnih promjenljivih za taj pogled, definišu se i sve funkcije kojima se obezbjeđuju funkcionalnosti pogleda i grafičkog interfejsa. Takođe, bitna funkcija koja se definiše je *render* funkcija koja popunjava prethodno definisani šablon. U *templates* folderu nalaze se HTML stranice svakog grafičkog interfejsa aplikacije. Biblioteka *underscoreJS*, osim mehanizma za rad sa šablonima, omogućava i upotrebu različitih funkcija za rad sa kolekcijama objekata. Modeli i kolekcije definišu se na sličan način kao i pogledi.

5. ZAKLJUČAK

Hibridne mobilne aplikacije se većim dijelom razvijaju pomoću tehnologija čija primarna upotreba nije vezana za razvoj mobilnih aplikacija. Upravo ova činjenica ukazuje na potencijalne probleme koji se mogu javiti pri razvoju, a koji su vezani za funkcionalna ograničenja i performanse aplikacije koje jesu bitno lošije u odnosu na *native* aplikacije [14]. Hibridne mobilne aplikacije treba da obezbijede da se jedan programski kod može, bez značajnijih izmjena, izvršavati na različitim mobilnim platformama. Upravo zbog navedenih razloga veoma je važno da aplikacija bude dobro organizovana. Organizacija se može ostvariti podjelom aplikacije na više logičkih nivoa i modularnom organizacijom izvornog koda. Podjelom aplikacije na funckionalne nivoe i module omogućava se jednostavnije održavanje aplikacije, jednostavnija prilagođenja određenim platformama i jednostavnije dodavanje novih funkcionalnosti. Osim toga, smanjuje se mogućnost dupliranja dijelova koda zbog boljeg upravljanja zavisnostima (*dependency*) biblioteka i JavaScript fajlova. Modularnim organizovanjem aplikacije poboljšavaju se performanse koje se odnose na učitavanje potrebnih skripti i pojednostavljuje se sistem navigacije kroz aplikaciju. Upotrebom biblioteka koje omogućavaju jednostavnije kreiranje grafičkog korisničkog interfejsa, tj. upotrebom gotovih grafičkih elemenata ubrzava se i olakšava razvoj cijele aplikacije. Međutim, upotrebom gotovih razvojnih alata može se smanjiti mogućnost kreiranja proizvoljnog sadržaja, elemenata i komponenti.

Prilikom razvoja jednostavnih mobilnih aplikacija, modularizovanje aplikacije može dovesti do povećanja obima izvornog koda i komplikovanijeg upravljanja pojedinim dijelovima. Međutim, pri razvoju kompleksnijih aplikacija ili aplikacija koje sadrže veći broj ekrana i funkcionalnosti, modularizovanje koda i podjela na logičko-funkcionalne nivoe značajno olakšava razvoj i pojednostavljuje održavanje. Upotrebom dodatnih biblioteka mogu se uvesti eventualna ograničenja u implementaciji funkcionalnosti koje upotrebljenje biblioteke ne podržavaju u potpunosti.

Zbog funkcionalnih ograničenja i nešto lošijih performansi izvršavanja, hibridne mobilne aplikacije još uvijek ne mogu u potpunosti zamijeniti *native* mobilne aplikacije. Zbog činjenice da se za razvoj hibridnih mobilnih aplikacija koriste *web* tehnologije može se očekivati dodatni razvoj elemenata mobilnih uređaja koji su zaduženi za izvršavanje ovih aplikacija, kao i unaprijeđenje funcionalnosti *web* tehnologija. Razvoj hibridnih mobilnih aplikacija je jednostavniji, brži i jeftiniji u odnosu na razvoj *native* mobilnih aplikacija, posebno u slučaju kada se aplikacija koristi na više mobilnih platformi, i kada se mogu implementirati sve potrebne funkcionalnosti na datim platformama.

LITERATURA

[1] "Native, web or hybrid mobile-app development", IBM Corporation, april 2012.
[2] http://phonegap.com, posjećivano februara 2013.
[3] http://jquery.com, posjećivano februara 2013.
[4] A. Zibula, T. A. Majchrzak. Cross-Platform Development Using HTML5, jQuery Mobile, and PhoneGap: Realizing a Smart Meter Application. Lecture Notes in Business Information Processing, Volume 140, pages 16-3, 2013.
[5] http://jquerymobile.com, posjećivano februara 2013.
[6] http://dev.w3.org/html5/spec/, posjećivano februara 2013.
[7] A. Charland, B. Leroux. Mobile application development: web vs. native. Communications of the ACM, Volume 54, Number 5, pages 49-53, 2011.
[8] John M. Wargo, "PhoneGap Essentials", Addison-Wesley Professional, 2012.
[9] http://docs.phonegap.com, posjećivano februara 2013.
[10] http://requirejs.org, posjećivano februara 2013.
[11] http://backbonejs.org, posjećivano februara 2013.
[12] http://requirejs.org/docs/whyamd.html, posjećivano februara 2013.
[13] http://www.appliness.com/getting-started-with-html-mobile-application-development-using-jquery-mobile-requirejs-and-backbonejs/, posjećivano februara 2013.
[14] L. Corral, A. Sillitti, G. Succi. Mobile multiplatform development: An experiment for performance analysis. Procedia CS 10, pages 736-743, 2012.

Igor Dujlović, dipl.ing.
Elektrotehnički fakultet Banjaluka, RS, BiH
dujlovic@gmail.com
Oblasti interesovanja: objektno-orijentisano programiranje i modelovanje, Internet programiranje, razvoj mobilnih aplikacija, informacioni sistemi

doc. dr Zoran Đurić
Elektrotehnički fakultet Banjaluka, RS, BiH
zoran.djuric@etfbl.net
Oblasti interesovanja: sigurnost, kriptografija, PKI, platni sistemi i protokoli, formalna verifikacija, objektno-orijentisano programiranje i modelovanje, Internet programiranje, razvoj mobilnih aplikacija, XML-bazirana međuoperativnost, Web servisi, računarske mreže, penetration testing, sistem integracija