# Razvoj Android aplikacija u interakciji sa udaljenim DBMS-om sa osvrtom na JDBC

Miljan Radojičić, Jelena Ćosović
Studenti drugog ciklusa studija
Elektrotehnički fakultet
Istočno Sarajevo, Bosna i Hercegovina
e-mail adrese:
miljan.radojicic@yahoo.com, jelena-cosa@hotmail.com

*Sadržaj*— **Dati rad opisuje perspektivu i popularnost Android aplikacija. Posebna pažnja je data Android aplikacijama u client-server arhitekturi koje komuniciraju sa udaljenim DBMS-om uz upotrebu Java DataBase Connectivity (JDBC). Izvršeno je i njihovo poređenje sa aplikacijama koje koriste SQLite bazu podataka koja je ugrađena u Android. Takođe su dati značaj i upotreba ove baze podataka. Na samom kraju je data trenutna upotreba i perspektiva razvoja Android aplikacija u client-server arhitekturi.**

***Ključne riječi-Java; Eclipse; Android; JDBC;DBMS;SQLite***

## I. UVOD

Ideja ovog rada je da se napravi osvrt na mogućnosti aplikacija Android operativnog sistema za smartphone uređaje sa aspekta pristupa aplikacije na udaljenu bazu podataka i manipulacije sa njenim podacima. Potreba za razvojem ovih aplikacija se javlja kada se vrši manipulacija sa većim brojem podataka, koji ne mogu biti sačuvani u ugrađenoj SQLite bazi podataka, koja je relativno malog kapaciteta.

## II. ANDROID SQLITE

Da bi se postigle mnoge aktivnosti koje nude moderni mobilni telefoni, kao što su praćenje kontakata, događaja i zadataka, operativni sistem i aplikacije moraju biti prilagođeni za čuvanje i praćenje velikih količina podataka. Većina ovih podataka je strukturirano u tabele, u obliku redova i kolona. Svaka Android aplikacija je izdvojena za sebe, što znači da može da čita i piše samo one podatke koje je ona stvorila, ali dijeljenje podataka između okvira aplikacija je takođe potrebno. [1]

SQLite je open-source baza podataka koja je ugrađena u Android. SQLite podržava standardne funkcionalnosti relacionih baza podataka, kao što su SQL sintaksa, transakcije i prepared statements. Dodatna osobina ove baze podataka je da zahtijeva malu memoriju u toku izvršavanja aplikacije. [2]

SQLite podržava tipove podataka TEXT (sličan tipu String u Javi), INTEGER (sličan tipu long u Javi) i REAL (sličan tipu double u Javi). Svi podaci sa drugim tipovima moraju biti konvertovani u jedno od ova tri polja prije nego što budu smješteni u bazu podataka. SQLite ne vrši sopstvenu validaciju da li su podaci upisani u kolone tipa polja u koje su uneseni. [2]

Za razliku od većine relacionih baza podataka, SQLite nema klijent/server arhitekturu. Većina velikih sistema baza podataka ima veliki serverski paket koji je glavni pokretač baze podataka. Nasuprot, SQLite nema odvojen server. Pokretanje baze podataka je integrisano u bilo koju aplikaciju kojoj je potreban pristup bazi. Jedini dijeljeni resurs za sve aplikacije je jedan fajl baze podataka, sačuvan na disku. Uklanjanjem servera znatno se smanjuje kompleksnost rada sa bazom. Ovo pojednostavljuje i softverske komponente i skoro eliminiše potrebu za podrškom naprednog operativnog sistema. Na kraju, zahtjevi SQLite baza podataka se svode samo na čitanje i upis određenih tipova skladištenja podataka. [3]

Osnovna osobina ove baze podataka je pouzdanost. Njena biblioteka je dizajnirana tako da podnosi raznovrsne sistemske greške, kao što su mala memorija, greške na disku, iznenadna gašenja uređaja. Teško se može desiti da baza podataka dođe u nepovratno stanje. [1]

SQLite je dostupan na svakom Android uređaju. Njegovo korišćenje na Androidu ne zahtijeva nikakvu instalaciju ili administriranje. Potrebno je samo da se definišu SQL statementi za kreiranje i update baze. Nakon toga bazom se može direktno upravljati pomoću Android platforme. Pristup ovoj bazi podataka uključuje i pristup fajl sistemu. Ovo može biti jako sporo. Zato je preporučljivo da se operacije sa bazom podataka izvršavaju asinhrono, na primjer koristeći AsyncTask klasu. [2]

Paket android.database unutar Android SDK sadrži sve potrebne klase za rad sa bazom podataka. Unutar ovog paketa se nalazi android.database.sqlite paket, koji sadrži sve SQLite specifične klase. [2]

## III. ANDROID APLIKACIJE I UDALJENI DBMS

Kao što je rečeno ranije, Android operativnom sistemu, a samim tim i svim Android aplikacijama je omogućena SQLite baza podataka, koja nema klijent/server arhitekturu, i iz tog razloga je sa njom jednostavno manipulisati. Android aplikacije na jednostavan način i relativno brzo mogu

pribavljati i smještati podatke u SQLite bazu podataka koja se nalazi na klijentskoj strani.

Određene Android aplikacije imaju potrebu za manipulacijom sa većim brojem podataka, koji ne može biti sačuvan unutar SQLite baze podataka, koja je relativno malog kapaciteta. Iz tog razloga se javlja potreba za komunikacijom Android aplikacije sa udaljenim bazama podataka na serveru, koje su znatno većeg kapaciteta i služe da čuvaju velike količine podataka. Android SDK ne sadrži paket sa klasama za komunikaciju sa udaljenom bazom podataka, pa iz tog razloga se konekcija na udaljenu bazu mora obezbijediti preko drugih paketa unutar projekta.

Dva osnovna tipa komunikacije Android aplikacije sa bazom podataka na server, kao i bilo koje druge aplikacije koja se razvija u Java okruženju, su komunikacija preko web servisa i komunikacija preko Java konektora. Oba tipa komunikacije omogućavaju Androidu da uspostavi konekciju na bilo koju relacionu bazu podataka na serveru.

## IV. KOMUNIKACIJA APLIKACIJA SA BAZOM PODATAKA KORIŠĆENJEM JAVA API

Većina aplikacija kreira podatke koji produžuju životni ciklus njihovog procesa, za šta je potrebno da objekti aplikacije budu sačuvani u trajnu memoriju i ponovo pribavljeni iz nje. Posmatrajući čiste objektno-orijentisane aplikacije, tj. aplikacije koje su kompletno razvijene koristeći okruženje sa objektno-orijentisanim jezicima kao što je Java, objekti jedne aplikacije se odnose i na mnoge druge objekte unutar te aplikacije, što za rezultat daje kompleksnu hijerarhiju objekata. Dakle, potrebno je da se pronađu mehanizmi za efikasno upravljanje kompleksnim objektima, zajedno sa njihovim vezama, posebno ako su objekti sačuvani na različitim lokacijama. [4]

Jedan od načina za upravljanje trajnim objektima je da se koristi DBMS (Database Management System) ili alat za trajno čuvanje podataka. U ovom slučaju aplikacija može biti povezana sa DBMS-om koristeći standardni paket za povezivanje koji omogućava funkcije za čuvanje i pribavljanje objekata za upravljanje transakcijama i oporavkom i druge standardne funkcije baze podataka. Jedan od ovih načina povezivanja je JDBC (Java Database Connectivity) koji omogućava Java aplikacijama konekciju na relacioni DBMS za evaluaciju SQL iskaza (eng. SQL statements) i obradu rezultata. [4]

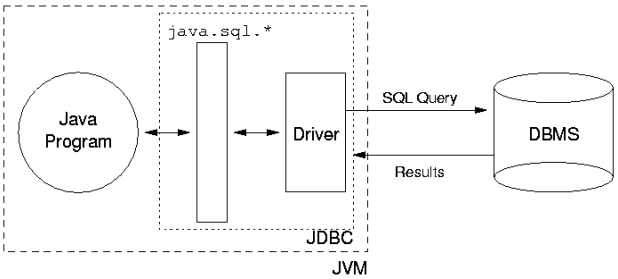
### *A. JDBC API*

JDBC API je jedan od najznačajnijih i najkorišćenijih API-ja Java platforme. On obezbjeđuje jednostavan i konzistentan način rada sa podacima iz baza podataka i drugih tabelarnih izvora podataka. Ovaj API se koristi za uspostavljanje konekcije prema izvoru podataka, izvršavanje složenih SQL (Structured Query Language) naredbi, uzimanje podataka iz izvora podataka, mijenjanje i brisanje podataka u izvoru podataka. [5]

JDBC API omogućava da se na jednostavan način radi sa različitim sistemima za upravljanje bazama podataka (eng. DBMS), bez obzira na kojoj platformi se DBMS nalazi, bez obzira na to koji tip DBMS-a je u pitanju, kao i bez obzira na to koja kombinacija platforme i sistema je u pitanju. Iz ovog razloga identičan programski kod može da se koristi, na primjer, za rad sa Oracle sistemom na Linux platformi i MySQL sistemom na Windows platformi. [5]

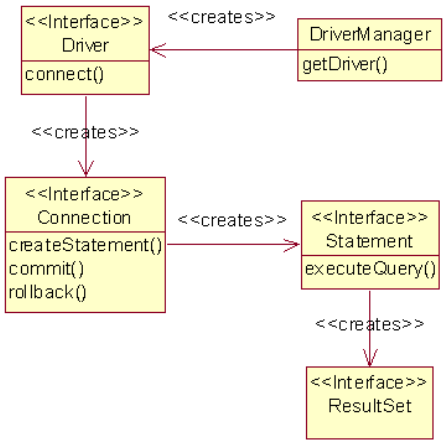
JDBC drajveri ne dolaze uz JDBC API. Proizvođači sistema za upravljanje bazama podataka kreiraju odgovarajuće JDBC drajvere. Svi drajveri se koriste na isti način, a instalacija drajvera podrazumijeva uključivanje jar arhiva u kojima se oni nalaze. [5]

JDBC je jako pogodan za klijent-server aplikacije. Klijent-server ili dvoslojne aplikacije predstavljaju najjednostavniju arhitekturu aplikacija u kojoj se koriste DBMS sistemi. Ova arhitektura se sastoji od klijentskog i serverskog sloja. Klijentski sloj komunicira direktno sa serverskim slojem, bez bilo kakvog međusloja. Poslovna logika, kao i upravljanje konekcijama i transakcijama nalazi se u klijentskom sloju, dok se u serverskom sloju nalazi DBMS. U ovakvoj arhitekturi JDBC komponente se nalaze u klijentskom sloju. [5]




Slika 1.0. Dvoslojna arhitektura sa JDBC komponentama.

Jezgro podsistema za rad sa bazama podataka predstavljaju klasa java.sql.DriverManager i interfejsi java.sql.Connection, java.sql.Statement i java.sql.ResultSet. [5]




Slika 1.1. Veze između osnovnih klasa i interfejsa pri komunikaciji sa bazom podataka

Klasa DriverManager predstavlja osnovni servis za rad sa JDBC drajverima. Objekat tipa Connection predstavlja konekciju na sistem za upravljanje bazama podataka. Statement interfejs enkapsulira sve naredbe za rad sa relacionim bazama podata ka, kao što su SELECT, INSERT, UPDATE, DELETE. Objekat tipa ResultSet predstavlja tabelu koja sadrži podatke koji su generisani izvršavanjem upita nad bazom podataka. [5]

*1) Konekcija na DBMS i rad sa bazom podataka korišćenjem SQL upita i uskladištenih procedura*

Java aplikacija uspostavlja konekciju sa DBMS-om korišćenjem JDBC drajvera. Konekcija na DBMS predstavljena je objektom tipa Connection. Java aplikacija može imati više konekcija ka različitim DBMS sistemima, ka različitim bazama podataka na jednom DBMS-u ili ka jednoj bazi podataka na jednom DBMS-u. Objekat tipa Connection može se kreirati korišćenjem DriverManager klase ili korišćenjem klase koja implementira DataSource interfejs. [5]

U ranijim verzijama JDBC specifikacije Java aplikacija je morala eksplicitno učitavati drajver (Driver klasu) pozivom metode `forName` klase `Class`:

```
Class.forName("com.mysql.jdbc.Driver");
```

Metoda `getConnection` klase `DriverManager` vraća objekat tipa `Connection` ako je uspješno uspostavljena konekcija sa bazom podataka koja se nalazi na specificiranoj URL adresi:

```
Connection con = DriverManager.getConnection(
"url", "user", "password");
```

U slučaju greške javlja se izuzetak `SQLException`.

Za upotrebu SQL naredbi i rad sa relacionim bazama podataka koristi se interfejs `Statement`. Objekti klasa koje implementiraju `Statement` interfejs koriste se za izvršavanje statičkih SQL naredbi koje ne primaju parametre. Statement interfejs nasljeđuje `PreparedStatement` interfejs, kojeg nasljeđuje `CallableStatement` interfejs. Objekti klasa koje implementiraju `PreparedStatement` interfejs koriste se kada je unaprijed potrebno kreirati i kompajlirati SQL naredbe, dok se objekti klasa koje implementiraju `CallableStatement` koriste za izvršavanje uskladištenih procedura i funkcija. Objekti klasa koje implementiraju `Statement` interfejs koriste se za izvršavanje statičkih SQL naredbi i vraćanjem rezultata nastalih njihovim izvršavanjem. Rezultati se vraćaju u obliku objekata klasa koje implementiraju `ResultSet` interfejs. Najvažnije metode interfejsa Statement su `execute`, `executeUpdate`, `executeQuery` i `close`. Svaka od prve tri naredbe izvršava specificiranu SQL naredbu, koja može biti tipa SELECT, INSERT, UPDATE, DELETE. Ako je specificirana SQL naredba tipa SELECT, metoda vraća rezultat u obliku `ResultSet` objekta. Metoda `close` zatvara tekući Statement objekat. [5]

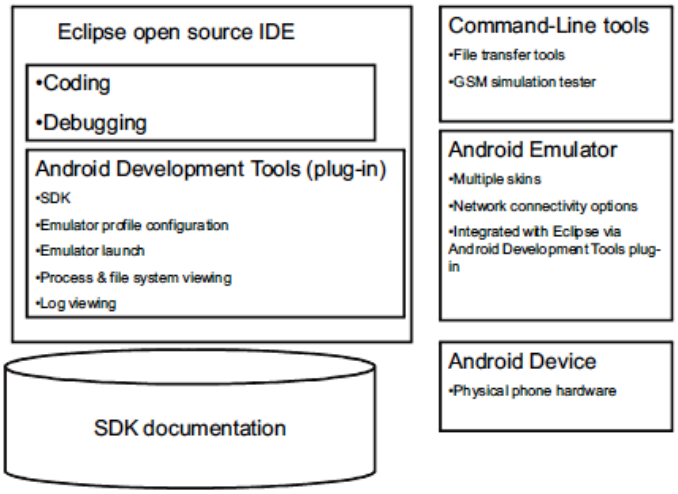
*2) Manipulacija sa rezultatima izvršavanja SQL naredbi*

Rezultati nastali izvršavanjem SQL naredbi smještaju se u objekat tipa ResultSet. U pitanju su objekti tabelarnog tipa. Informacije o nazivima kolona, tipu podataka koji se čuva u kolonama i vrijednostima mogu se dobiti putem ResultSet i ResultSetMetaData objekata. Postoje različiti tipovi ResultSet objekata koji omogućavaju različite načine manipulacije rezultatima. [5]

Objekti klasa koje implementiraju interfejs ResultSet predstavljaju rezultujući skup podataka, u tabelarnoj formi, koji je nastao izvršavanjem SQL upita nad bazom podataka. Objekat tipa ResultSet sadrži kursor koji pokazuje na poziciju tekućeg reda podataka u rezultujućem skupu podataka. Inicijalno, kursor se nalazi na poziciji ispred prvog reda podataka. Metoda `next` pomjera kursor na sljedeću poziciju i vraća istinitu vrijednost kada je ovo pomjeranje uspješno izvršeno. Ovu metodu je moguće koristiti u while petlji kako bi se izvršila iteracija kroz čitav rezultujući skup podataka. [5]

*B. Razvoj Android aplikacija i upotreba JDBC*

Android aplikacije, kao većina mobilnih aplikacija, razvijaju se u host-target korisničkom okruženju. Drugim riječima, aplikacije se razvijaju na računaru i pribavljaju se na ciljni mobilni telefon za testiranje i upotrebu. Za pisanje vlastite Android mobilne aplikacije potrebno je sakupiti potrebne alate i podesiti odgovarajuće korisničko okruženje na računaru. Najbolji skup alata za razvoj Android aplikacija je Android SDK, koji podržava nekoliko različitih integrisanih razvojnih okruženja. Od ovih okruženja za korišćenje je najbolji Eclipse IDE, jer je najbolje integrisan sa Android SDK. Unutar ovog skupa alata nalazi se većina osnovnih Java paketa (java.lang, java.io), kao i Android-specifični paketi. [1]




Slika 1.2. Razvojno okruženje za pravljenje Android aplikacija uz upotrebu Eclipse IDE okruženja

Na slici 1.2. prikazano je tipično Android razvojno okruženje uključujući i stvarni hardver i Android emulator uz upotrebu Eclipse IDE okruženja. Iako nije jedini alat koji se koristi za razvoj Androida, Eclipse može da igra značajnu ulogu u razvoju aplikacije zato što pruža bogato Java okruženje za kompajliranje i otklanjanje grešaka. Takođe je bitan i Android emulator, koji služi kao zamjena za grafički korisnički interfejs stvarnog mobilnog uređaja na kome je pogodno vršiti testiranje aplikacije.

Pri razvoju Android aplikacija koriste se četiri osnovne komponente definisane u Android arhitekturi: Activity, Service, Broadcast i Intent Receivers i Content Providers. Svaka od ovih komponenti ima posebnu ulogu, pa tako Activity najčešće predstavlja jedan prozor aplikacije i usko je vezan sa korisničkim interfejsom, dok se Service komponente izvršavaju u pozadini. Elementi korisničkog interfejsa se definišu kroz xml fajlove, koji sadrže sve atribute i osobine definisanih elemenata i uključuju se u Activity. [1]

Osim gore navedenih komponenti, Android aplikacija ima još jednu bitnu komponentu, a to je Android Manifest fajl. Prije nego što Android sistem pokrene komponentu aplikacije,

sistem mora znati da ta komponenta postoji, čitajući upravo ovaj manifest fajl. U ovom fajlu moraju biti deklarisane sve komponente aplikacije i sve korisničke dozvole koje aplikacija zahtijeva, kao što su pristup internetu, ili pristup kontaktima korisnika radi čitanja. U njemu se deklariše minimalni API nivo koji aplikacija zahtijeva, kao i hardverske i softverske osobine koje aplikacija zahtijeva ili upotrebljava (npr. kamera, bluetooth).

Što se tiče primjene JDBC konektora u Android aplikacijama, situacija je slična kao i kod ostalih Java aplikacija. Pošto JDBC konektor nije standardna biblioteka Android SDK paketa, on se mora posebno uključiti u projekat kroz libs folder koji se nalazi u korijenu projekta. Nakon toga, da bi se omogućila upotreba klasa ove biblioteke, u nekim slučajevima je potrebno je učitati odgovarajući drajver (Driver klasa). Tip drajvera se odabira zavisno od toga na koji DBMS se vrši konekcija, a unutar jedne JDBC biblioteke može se nalaziti samo jedan tip drajvera. Nakon učitavanja drajvera, JDBC konektor se može koristiti u Android aplikacijama kao i u svim drugim Java aplikacijama.

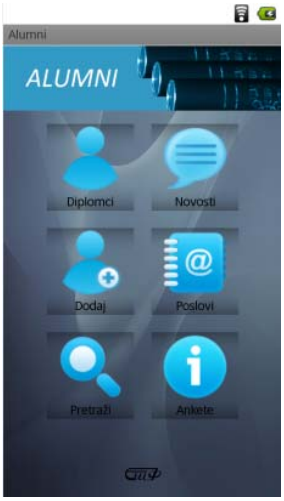
Ono što je specifično za JDBC konektor u Android aplikacijama je da je poželjno da se sve operacije sa bazom podataka, od konekcije do pribavljanja podataka, obavlja pomoću niti tipa AsyncTask. Ovaj vid upotrebe klasa JDBC konektora i njihovih metoda je najbolji iz razloga što podržava veliki broj verzija Android platforme. Osnovne metode klase tipa AsyncTask su `doInBackground(),onPreExecute()` `onPostExecute()` i `onProgressUpdate()`. Metode `onPreExecute()` i `onPostExecute()` služe da nit obavi određene operacije prije, odnosno posle glavne operacije koju izvršava, dok metoda `onProgressUpdate()` služi da prikaže progres glavne operacije koju nit izvršava. Nit tipa AsyncTask koristi metodu `doInBackground()` koja izvršava sve operacije nad bazom podataka, i na osnovu ulaznih parametara može da vrati izlazni parametar. Ovaj parametar je najčešće tipa ResultSet.

## V. REALIZOVANA APLIKACIJA

Aplikacija koristi dio sistema baze podataka Elektrotehničkog fakulteta, Univerziteta u Istočnom Sarajevu, koji se naziva „Alumni“ i sadrži podatke koji se odnose na diplomce datog fakulteta. Pomoću nje se može izvršiti pregled podataka o diplomcima fakulteta, pregled dostupnih radnih mjesta za njihovu struku, pregled anketa poslodavaca, pregled novosti vezanih za diplomce, poslove i ankete, kao i unos novih diplomaca u postojeću bazu podataka. Aplikacija takođe sadrži mogućnost pregleda informacija o fakultetu, pojedinačnim diplomcima, radnim mjestima i anketama, i mogućnost pretrage poslova i diplomaca po odgovarajućim kriterijumima.

Za implementaciju Android aplikacije „Alumni“ korišćen je Java programski jezik unutar programskog okruženja Eclipse. Unutar Eclipse okruženja korišćen je Android SDK razvojni alat koji sadrži Resource editor kojim se postavljaju elementi stranice kroz xml, i pravi njihov raspored, i Android emulator, koji predstavlja alternativni interfejs za Android uređaj. Za rad sa bazom podataka korišćena je sintaksa SQL jezika, s tim da u implementaciji preovladavaju uskladištene procedure (eng. stored procedures). Za izradu stored procedura nad bazom podataka i pregled same baze korišćen je alat za razvoj baze podataka SQL Server 2008 R2 Management Studio.

Korisnički interfejs aplikacije „Alumni“ je jednostavan za upotrebu, intuitivan i funkcionalan. Takođe, kroz dati interfejs se relativno brzo mogu opsluživati željene akcije korisnika. Jedini izuzetak je loša internet konekcija, kada pribavljanje podataka iz baze može potrajati. Važno je napomenuti i da je aplikacija „Alumni“ podržana na dosta različitih vrsta Android uređaja.




Slika 1.3. Izgled početnog prozora aplikacije „Alumni“

Klase tipa Activity su u direktnoj vezi sa korisničkim interfejsom Android aplikacije, pa se zato pomoću njih može opisati realizacija interfejsa, gdje jedna Activity klasa predstavlja jedan prozor u aplikaciji.


Slika 1.4. Realizacija dijaloga za prikaz podataka iz baze

Aplikacija „Alumni“ realizovana je kroz ukupno sedam activity-ja. Zasebne dijelove korisničkog interfejsa activity-ja predstavljaju i dijalozi (realizovani pomoću klasa tipa Dialog), koji uglavnom imaju prezentacionu ulogu (Sl. 1.3).

Android aplikacija omogućava i izmjenu prikaza pomoću selekcije odgovarajućih podataka što se postiže pretragom. Pretraga se vrši po kriterijumima koji se odnose na podatke o

diplomcima i poslovima, s tim što su vrste pretrage organizovane u tabove. Unutar svakog taba nalazi se skup polja za unos gdje sadržaj svakog polja predstavlja jedan od kriterijuma za pretragu.


Slika 1.5. Realizacija pretrage diplomaca i poslova u activity-ju „Pretraga“

Osim prikaza i pretrage, omogućen je i unos novih diplomaca u bazu podataka. To se takođe realizuje pomoću polja za unos, gdje sadržaj svakog polja predstavlja jedan podatak koji se unosi u bazu podataka. Podaci se unose u bazu podataka u redovima, gdje sadržaj skupa polja za unos predstavlja jedan red u bazi.


Slika 1.6. Realizacija interfejsa za registraciju diplomaca

*A. Komunikacija sa bazom podataka*

Aplikacija „Alumni“ uspostavlja konekciju sa bazom podataka pomoću Java klasa iz Java DataBase Connectivity (JDBC) paketa. Jedna od tih klasa je DriverManager.

Klasa SQLConnection.java realizuje pristup bazi podataka na taj način što kreira konekciju, odnosno objekat tipa Connection. Konekcija se kreira na osnovu stringova za konekciju koji se učitavaju iz properties fajla. Slanje zahtjeva za konekciju se obavlja pomoću niti (eng. Thread) koja je u Android SDK paketu tipa AsyncTask, i izvršava se pomoću metoda `execute()` i `get()` kao što je prikazano u sljedećem dijelu koda:

```
database=prop.getProperty("database");
username=prop.getProperty("dbuser");
password=prop.getProperty("dbpassword");
connString="jdbc:jtds:sqlserver://"+database;";

conn = dbConnect.execute().get();
```

Metoda `execute()` služi da nit započne operaciju koju treba izvršiti, dok se metoda `get()` koristi za dobijanja rezultata izvršenih operacija. Rezultate obično vraća standardna metoda niti `doInBackground()`.

Sve metode koje komuniciraju sa bazom podataka koriste se nitima tipa AsyncTask. Na sljedećem primjeru prikazana je klasa tipa AsyncTask koja predstavlja nit za konekciju na bazu podataka, i za ulazne parametre connString, username i password vraća izlaznu vrijednost conn koji je tipa Connection.

```
public class AsinhroniConnect extends AsyncTask<Void,
Void, Connection>
{
    @Override
    protected Connection doInBackground(
Void... params)
    {
        try
        {
            conn = DriverManager.getConnection(
connString,username,password);
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }

        return conn;
    }
}
```

Nakon što se uspostavi konekcija na bazu podataka, otvara se mogućnost za izvršavanje SELECT, INSERT, UPDATE i DELETE upita nad tom bazom. Upiti tipa SELECT kao rezultat vraćaju objekat tipa ResultSet, koji je i izlazni parametar metode `doInBackground()` niti za pribavljanje podataka.

Na početku, metoda za kreiranje sadržaja liste koristi `query` parametar za kreiranje objekta tipa PreparedStatement:

```
stmt=conn.prepareStatement(query);
```

Nakon toga, pomoću niti `AsinhroniSelect` pribavlja se skup podataka iz baze, u formi tabele:

```
rs=(ResultSet)
new AsinhroniSelect().execute(stmt).get();
```

Nit `AsinhroniSelect` unutar svoje metode `doInBackground()` izvršava komandu kojom puni objekat tipa ResultSet, čime se dobija skup podataka:

```
rs=stmt.executeQuery();
```

Ukoliko se vrši unos novih podataka u bazu podataka, koristi se nit `AsinhroniInsert`, koja kroz svoju metodu `doInBackground()` izvršava komandu:

```
stmt.executeQuery();
```

Implementacija niti za INSERT operaciju je data na sljedećem primjeru:

```
public class AsinhroniInsert extends
AsyncTask<PreparedStatement, Void, Integer>
{
   @Override
   protected Integer doInBackground(
PreparedStatement... params)
   {
      PreparedStatement stmt=params[0];
      try
      {
         stmt.executeQuery();
         return 1;
      }
      catch (SQLException e)
      {
         return 0;
      }
   }
}
```

Objekat tipa ResultSet, koji se dobija kao rezultat SELECT operacije nad bazom podataka, je tabelarnog tipa i obezbjeđuje pokazivač na selektovani red sa podacima. Prvi red je uvijek prazan, pa da bi se pristupilo podacima, potrebno je preći u naredni red pomoću metode `next()`. Sljedeći primjer pokazuje na koji način se vrši manipulacija podacima ResultSet objekta:

```
while(rs.next())
{
   listItem ="";
   for(i=0;i<parameters.size();i++)
   {
      listItem+=rs.getString(parameters.get(i));
      listItem+=" ";
   }

   datalist.add(listItem);

   if(id!=null)
   idList.add(rs.getString(id));
}
```

Na datom primjeru je prikazana upotreba metode `next()` pomoću koje se vrši selekcija redova podataka ResultSet objekta. Pored ove metode, prikazana je i jedna od metoda za pribavljanje podatka iz selektovanog reda: `getString()`. Ove metode mogu imati dvije različite vrste parametara, a to su redni broj podatka u selektovanom redu i naziv kolone tabele u kojoj se nalazi podatak. Za pribavljanje jednog podatka iz objekta ResultSet dovoljno je da metoda sadrži jedan od ova dva parametra. Upotreba metode ovog tipa zavisi od toga koja vrsta podataka se pribavlja iz objekta tipa ResultSet, pa iz tog razloga postoji više ovih metoda, kao što su `getInt()`,`getFloat()`,`getDate()`, itd.

## VI. Zaključak

Posebna pažnja u ovom radu data je komunikaciji Android aplikacija sa udaljenom bazom podataka. Iako Android ima ugrađenu SQLite bazu podataka, ona često ne može da zadovolji zahtjeve aplikacije, pogotovo ako se radi o velikim količinama podataka. Ovo je slučaj kod razvoja informacionih sistema gdje klijentski dio čini Android aplikacija, a baza podataka se nalazi na udaljenom serveru. Zato je neophodno omogućiti njihovu međusobnu komunikaciju.

Ovom aspektu razvoja Android aplikacija se još uvijek ne pridaje dovoljna pažnja. Iako ovakve aplikacije već postoje, njihova implementacija je slabo zastupljena, pa je i jedan od ciljeva ovog rada da skrene pažnju na njihovu implementaciju. Realizovana klijentska Android aplikacija „Alumni“, koja koristi dio informacionog sistema Elektrotehničkog fakulteta, najbolje prikazuje kako se ova implementacija odvija.

Nakon realizacije aplikacije, može se reći da primjena Android aplikacija u client-server arhitekturi ima perspektivu, jer se performanse pametnih telefona konstantno i brzo poboljšavaju, i približne su performansama desktop računara. Aplikacija ne realizuje u potpunosti sve mogućnosti upotrebe ovog dijela informacionog sistema, ali pokazuje da se na jednostavan način može ostvariti njena interakcija sa bazom podataka.


## Zahvalnica

Veliku zahvalnost dugujemo prof. dr Branku Perišiću, koji je ujedno i mentor ovog rada, i asistentu Vladimiru Vujoviću na neiscrpnoj pomoći tokom izrade teorijskog i praktičnog dijela rada.


## Literatura


[1] Rick Rodgers, John Lombardo, Zigurd Mednieks, Blake Meike. Android application development. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, May 2009.

[2] Lars Vogel (11.12.2012). Android SQLite Database and ContentProvider – Tutorial. Copyright © 2010, 2011, 2012 Lars Vogel. [Online] Available: http://www.vogella.com/articles/AndroidSQLite/article.html

[3] Jay A. Kreibich. Using SQLite. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, August 2010.

[4] Adrian Kobler , Moira C. Norrie. „OMS Java: Lessons Learned from Building a Multi-Tier Object Management Framework,“ in Proceedings of the Workshop on Java and Databases: Persistence Options, 1999, p. 1

[5] Zoran Đurić. Korak u Java svijet. Izdavač Elektrotehnički fakultet Banja Luka, 2010


## ABSTRACT


This paper describes the prospects and popularity of Android applications. Special attention is given to Android applications in client-server architecture that communicate with the remote DBMS using Java Database Connectivity (JDBC). They are compared with applications that use SQLite database that is built into Android. There is also described importance and use of this database. At the end is given current use and prospects of development of Android applications in a client-server architecture.


## DEVELOPMENT OF ANDROID APPLICATIONS RELYING ON REMOTE DBMS


Miljan Radojičić, Jelena Ćosović