IZRADA APLIKACIJE ZA UDALJENO UPRAVLJANJE ANDROID UREĐAJEM

Begović, Ivan

Master's thesis / Specijalistički diplomski stručni

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:159496

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

IVAN BEGOVIĆ

Z A V R Š N I R A D

IZDRADA APLIKACIJE ZA UDALJENO UPRAVLJANJE ANDROID UREĐAJEM

Split, kolovoz 2021.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Mobilne tehnologije

Z A V R Š N I R A D

Kandidat: Ivan Begović

Naslov rada: Izrada aplikacije za udaljeno upravljanje android uređajem

Mentor: Marina Rodić, predavač

Split, kolovoz 2021.

Sadržaj

Sažetak ... **3**

Summary ... **4**

1. Uvod ... **5**

2. Korišteni alati i tehnologije ... **6**

2.1. JAVA programski jezik ... 6

2.2. Apache Netbeans IDE ... 6

2.3. Android Studio ... 6

2.4. Klijent-poslužitelj arhitektura ... 7

3. Računalna aplikacija ... **8**

3.1. Početni prikaz ... 8

3.2. Popis kontakata ... 12

3.3. Dohvaćanje imena datoteka sa poslužitelja ... 16

3.3.1. Preuzimanje datoteke ... 18

3.3.2. Postavljanje slike zaslona android uređaja ... 21

3.4. Slanje datoteke na poslužitelj ... 22

3.5. Slanje SMS poruke ... 25

3.6. Dodavanje događaja u kalendar ... 26

3.7. Kontrola zvuka ... 29

4. Android aplikacija ... **31**

4.1. Potrebne dozvole ... 31

4.2. Koncept izvršavanja android aplikacije ... 33

4.3. Prihvaćanje konekcije ... 34

4.4. Slanje liste kontakata ... 36

4.5. Slanje SMS poruke ... 37

4.6. Slanje imena datoteka ... 38

4.7. Dodavanje događaja u kalendar ... 40

4.8. Postavljanje slike zaslona 41

4.9. Prilagodba glasnoće 42

4.10. Slanje datoteke 43

4.11. Primanje datoteke 44

5. Zaključak 46

Literatura 47

Popis slika 48

Popis ispisa 49

Sažetak

Izrada aplikacije za udaljeno upravljanje android uređajem.

Kako bi se pospješilo upravljanje android uređajem bez potrebe spajanja istog kabelom na računalo ili korištenjem nekog posrednog medija kao na primjer nekog servisa u oblaku, ovaj rad donosi rješenje sa direktnim povezivanjem i upravljanjem android uređaja putem računala. U ovom radu su napravljene dvije aplikacije, jedna za android uređaj te jedna za računalo.

Kroz jednostavno sučelje na računalu moguće je u svega par klikova dodati novi podsjetnik u kalendar, poslati SMS poruku, osvježiti glazbenu listu dodajući novu glazbu sa računala direktno na android uređaj ili pak možda preuzimati slike i druge datoteke sa android uređaja na računalo.

Ključne riječi: Android, Java, povezivanje, računalo, upravljanje

Summary

Development applications for remote management of android device

To make management of android device more easier, without need for connecting it via cable on computer or to use some external cloud service, this paper is bringing solution by directly connecting android device and computer for better and easier management of android device. In this paper, two applications are made, one for computer and one for Android system.

With simple graphical design of computer application it is possible in just few clicks to add new reminder in calendar, send SMS message, refresh music playlist by adding new songs directly from computer or maybe download some images and other files from android device to computer

Key words: Android, computer, connection, Java, management

1. Uvod

Mobilni uređaji su postali neophodan dodatak svakodnevici. Svakim danom dobivaju sve više mogućnosti, inicijalno počevši od poziva, preko slanja SMS-ova pa sve do danas kada se moguće spojiti na internet i raditi gotovo iste stvari koje je prije bilo moguće raditi samo računalom.

No ipak i danas je nekad potrebno spojiti mobilni uređaj na računalo, a najčešći razlog je prebacivanje nekih datoteka, sa ili na mobilni uređaj. Do nedavno najčešći način prijenosa podataka između računala i mobilnog uređaja je bio korištenjem kabela, a u zadnje vrijeme da bi se izbjeglo fizičko spajanje računala i mobilnog uređaja koriste se servisi u oblaku. Tako da se onda podaci sa jednog uređaja prebace u oblak, pa se onda sa drugog uređaja pristupi tom oblaku i preuzmu željeni podaci. No i taj način ima mane poput dostupne količine memorije u oblaku, brzine, cijene ili pak sigurnosti.

Stoga je cilj ovog rada izraditi dvije aplikacije, jednu računalnu i jednu za android uređaj, koja će povezati ta dva uređaja direktno bez potrebe za fizičkim kabelom ili nekim trećim posredničkim servisom. Sve što je potrebno je da oba uređaja budu spojena na istu bežičnu mrežu.

Jednom kad su oba uređaja povezana, korisnik može pomoću sučelja računalne aplikacije pristupiti svim kontaktima spremljenim na android uređaj, slati SMS poruke, dodavati događaje u kalendar, pristupiti svim slikama, audio datotekama, videima i dokumentima, upravljati glasnoćom mobilnog uređaja, preuzimati te slati datoteke na mobilni uređaj.

U prvom dijelu rada je objašnjeno koji su sve alati i tehnologije korištene za realizaciju ovog rada. Potom u drugom dijelu se objašnjava na koji način radi računalna aplikacija, dok se u trećem dijelu objašnjava princip rada android aplikacije.

Na kraju je dan zaključak kao objedinjenje cjelokupnog rada.

2. Korišteni alati i tehnologije

2.1. JAVA programski jezik

Java je objektno orijentirani programski jezik razvijen od strane tvrtke Sun Microsystems [1]. Posebnost Java programskog jezika u odnosnu na druge programske jezike je sposobnost da je jednom kompajliran kôd moguće pokrenuti na bilo kojem uređaju koji na sebi ima Java Virtualnu Mašinu (JVM), što znači da je praktički kompajliran kôd neovisan o operacijskom sustavu.

Obje aplikacije, i računalna i android aplikacija, ovog rada su napisane u Java programskom jeziku.

2.2. Apache Netbeans IDE

Za razvoj računalne aplikacije korišten je Apache Netbeans IDE, razvojni alat otvorenog kôda, primarno namijenjen za Java programski jezik, ali uz određene dodatke moguće ga je koristiti i za razvoj aplikacija u drugim programskim jezicima, poput C, C++, PHP, itd.

Netbeans razvojni alat je pogodan zato što osim razvoja same logike aplikacije omogućuje i razvoj grafičkog sučelja i jednostavno povezivanje istog sa pozadinskom logikom aplikacije.

2.3. Android Studio

Najpogodniji alat za razvoj android aplikacija je Android Studio [2], koji je ujedno i službeni alat za Android operacijski sustav. Programski jezici koji se koriste za razvoj android aplikacije u Android Studiu su Java i Kotlin (od svibnja 2019. godine Google predlaže Kotlin kao preferirani programski jezik za razvoj android aplikacija u odnosu na Javu, međutim Java je i dalje korištena, čak možda i više zastupljena od Kotlina). U ovom radu, za razvoj android aplikacije, korišten je Java programski jezik.

Pogodnost razvoja u Android Studiu je da jednom kad je aplikacija napravljena i kompajlirana, moguće ju je objaviti na Google Play Store-u (Google-ovoj službenoj aplikaciji za objavu i preuzimanje aplikacija).

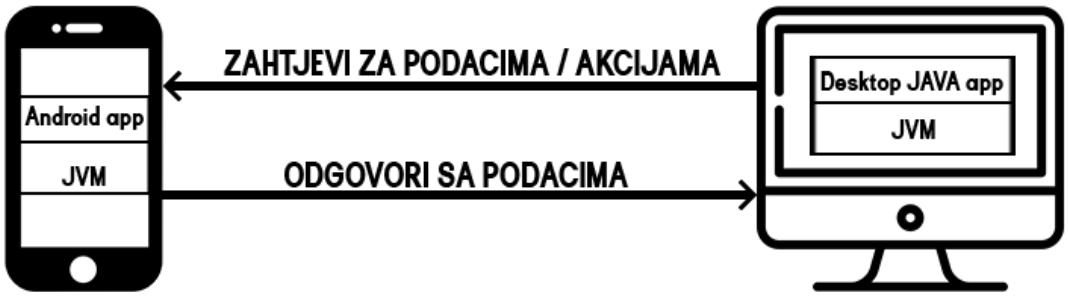
2.4. Klijent-poslužitelj arhitektura

Obzirom da aplikacije u ovom radu razmjenjuju podatke u stvarnom vremenu, potrebno je uspostaviti arhitekturu klijent-poslužitelj [3].

To znači da će jedna aplikacija slati upite drugoj te će potom ta druga aplikacija te upite obrađivati i rezultate obrade slati natrag prvoj aplikaciji. Ona aplikacija koja šalje upite je klijentska aplikacija i to je u slučaju ovog rada, računalna aplikacija, dok je android aplikacija poslužiteljska aplikacija jer ona obrađuje sve upite i šalje ih natrag desktop aplikaciji.

Kako bi klijentska i poslužiteljska aplikacija bile povezane koristiti će Protokol upravljanja prijenosom (engl. *Transmission Control Protocol*, TCP) i programiranje korištenjem utičnica (engl. *Socket*).

Važno je naglasiti da kako bi sustav utičnica pravilno funkcionirao, oba uređaja (klijent i poslužitelj) moraju biti spojena na istu mrežu.



Slika 2-1: Shematski prikaz načina komunikacije android aplikacije i računalne aplikacije

Slika 2-1 prikazuje načelni koncept komunikacije i razmjene poruka između klijenta i poslužitelja.

3. Računalna aplikacija

Računalna aplikacija u ovom radu predstavlja klijentsku aplikaciju. To znači da ona šalje zahtjeve za određenim podacima ili akcijama prema android aplikaciji te potom ovisno o odgovoru, dohvaćene podatke prikazuje na određeni način.

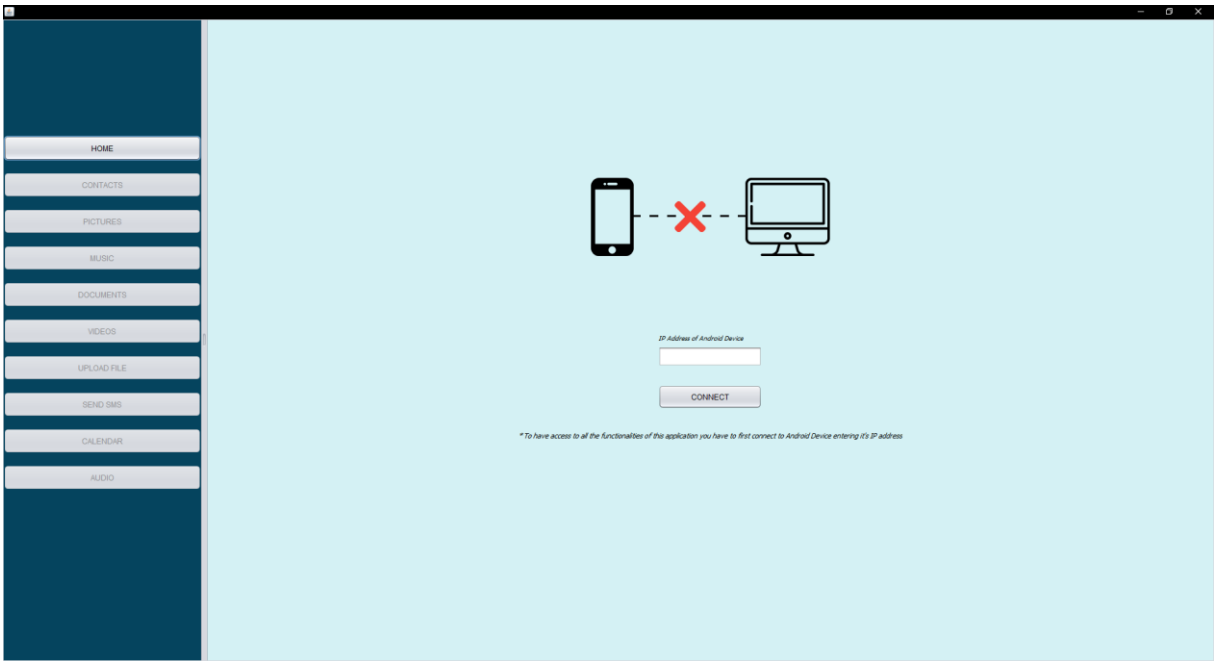
Stoga je računalna aplikacija podijeljena u dva dijela. Prvi dio je izbornik sa ponuđenim akcijama, a drugi dio je prostor u kojem se ispisuju dohvaćeni podaci odnosno prostor za unos podataka koji će biti poslani za realizaciju određene akcije.

Mogućnosti koje računalna aplikacija dozvoljava su:

- Dohvaćanje svih kontakata.
- Slanje SMS-a.
- Dohvaćanje svih slika, dokumenata, videa i glazbe iz memorije android uređaja.
- Preuzimanje bilo koje slike, dokumenta, videa ili glazbe sa uređaja na računalo.
- Slanje bilo koje datoteke sa računala na android uređaj.
- Dodavanje događaja u kalendar.
- Mijenjanje glasnoće zvuka poziva, zvuka obavijesti, alarma i glasnoće zvuka tijekom poziva.

3.1. Početni prikaz

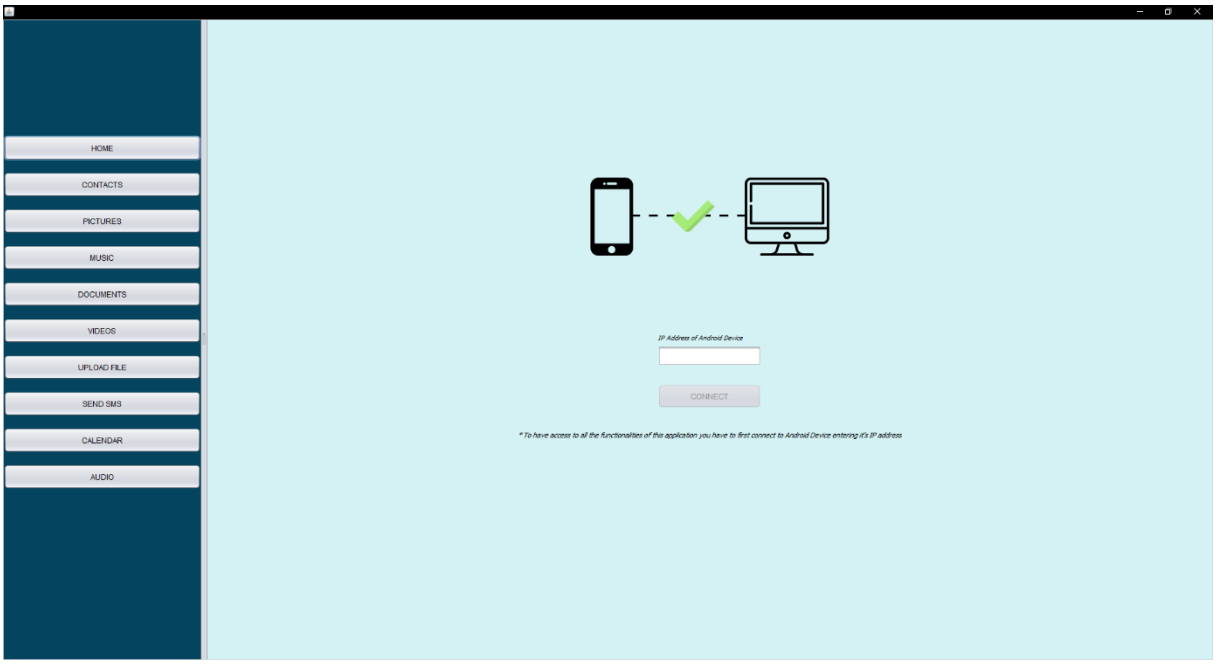
Po pokretanju računalne aplikacije otvara se Početni zaslon (engl. *Home*) prikazan na slici Slika 3-1.



Slika 3-1: Prikaz početnog zaslona prije uspostavljene konekcije

Da bi se mogle koristiti mogućnosti i akcije, potrebno je prvo povezati računalo i android uređaj. To se radi na način da se u za to predviđeno polje računalne aplikacije na početnom zaslonu unese IP adresa android uređaja. IP adresa android uređaja je vidljiva u android aplikaciji na samome dnu zaslona. Unosom IP adrese android uređaja i klikom na dugme *CONNECT*, šalje se zahtjev za uspostavu konekcije prema android uređaju. Android uređaj ima mogućnost prihvatiti ili odbiti zahtjev za konekcijom.

Kad je zahtjev za konekcijom prihvaćen, omogućuju se sve mogućnosti i akcije koje se mogu napraviti prema spojenom android uređaju. Početni zaslon nakon prihvaćene konekcije se može vidjeti na slici Slika 3-2.



Slika 3-2: Prikaz početnog zaslona nakon uspostavljene konekcije

Pozadinska logika koja vrši konekciju može se podijeliti u dva dijela.

Prvi dio je dohvaćanje vrijednosti koja je unesena u polje predviđeno za unos IP adrese android uređaja i formiranje poruke koja će biti poslana prema poslužitelju (android uređaj) te potom i samo slanje te poruke. U ispisu Ispis 3-1 je prikazano dohvaćanje unesene IP adrese, dohvaćanje IP adrese računala, formiranje poruke koja će biti poslana poslužitelju, te u konačnici poziv metode za slanje poruke poslužitelju, prosljeđujući samu poruku toj metodi.

```
IP = homeIPTextField.getText();
String deviceName = "";
String deviceIP = "";
if (IP.isEmpty()) {
	homeIPTextField.setText("This field is required!");
	homeIPTextField.setBorder(
			BorderFactory.createLineBorder(Color.RED, 2));
	return;
}

try {
	InetAddress myIP=InetAddress.getLocalHost();
	String wholeName = myIP.toString();
	String[] separationOfNameAndIP = wholeName.split("/");
	if (separationOfNameAndIP.length == 2) {
		deviceName = separationOfNameAndIP[0];
		deviceIP = separationOfNameAndIP[1];
	}

} catch (UnknownHostException ex) {
	Logger.getLogger(
		NewJFrame.class.getName()).log(Level.SEVERE, null, ex);
}
```

```
if (deviceIP.equals("")) {
	homeIPTextField.setText("This device has no IP!!!");
	homeIPTextField.setBorder(
			BorderFactory.createLineBorder(Color.RED, 2));
	return;
}

String command = "connect~" + deviceIP;
sendStringMessageToAndroidDevice(command);
```

Ispis 3-1: Dohvaćanje unesene IP adrese, formiranje poruke i slanje iste poslužitelju

sendStringMessageToAndroidDevice je motoda koja se koristi prilikom zahtjeva za konekcijom, ali i za slanje bilo koje naredbe sa klijenta na poslužitelja. Ispis 3-2 prikazuje definiciju metode *sendStringMessageToAndroidDevice*.

```
private void sendStringMessageToAndroidDevice(String messageToSend) {
	Socket st;

	try {
		st = new Socket(IP, 7801);
		PrintWriter pwt;
		pwt = new PrintWriter(st.getOutputStream());
		pwt.write(messageToSend);
		pwt.flush();
		pwt.close();
		st.close();

	} catch (IOException e) {

	}
}
```

Ispis 3-2: Definicija metode sendStringMessageToAndroidDevice

Drugi dio logike, koja vrši uspostavljanje konekcije između klijenta i poslužitelja, je kada klijent ulazi u fazu slušanja na odgovor poslužitelja. Računalna aplikacija u ovom slučaju na kratko postaje poslužitelj jer ona sada čeka na poruku od android aplikacije. Računalna aplikacija se postavlja u fazu slušanja na ulaz (engl. *Port*) 7800 i ako se nijedan klijent ne spoji na taj ulaz u roku od 10 sekundi, ulaz se zatvara te je potrebno ponovno kliknuti dugme *CONNECT* i poslati zahtjev za konekcijom.

Ako je unutar 10 sekundi stigao odgovor, provjerava se sadrži li primljena poruka odgovor „*accept*“, ako da, to je onda znak da je konekcija uspješno uspostavljena i mogu se omogućiti korisniku sve željene akcije nad android uređajem.

Ispis 3-3 prikazuje kôd koji postavlja računalnu aplikaciju u fazu slušanja i obrade pristigle poruke.

```
try {
	ServerSocket ss = new ServerSocket(7800);
	ss.setSoTimeout(10000);
	Socket s = ss.accept();
	InputStreamReader isr = new InputStreamReader(s.getInputStream());
	BufferedReader br = new BufferedReader(isr);
	String message = "";

	while ((message = br.readLine()) != null) {
		if (message.isEmpty()) {
			break;
		}

		if (message.equals("accept")) {
			homeIPTextField.setText("");
			connectBtn.setEnabled(false);
			isConnected = true;
			changeStateofButtons();
			connectionStatus.setIcon(
				new ImageIcon  ("C:/connection_accepted.png"));
		} else {
			homeIPTextField.setText("CONNECTION DECLINED!");
		}
	}

	isr.close();
	br.close();
	     s.close();
	ss.close();
} catch (IOException e) {
	e.printStackTrace();
	try {
		ss.close();
	} catch (IOException ex) {
		 Logger.getLogger(
		    NewJFrame.class.getName()).log(Level.SEVERE, null, ex);
	}
}
```

Ispis 3-3: Dio metode za konekciju prema poslužitelju, zadužen za slušanje odgovora od poslužitelja

3.2. Popis kontakata

Jedna od mogućnosti koje su korisniku na raspolaganju kada uspostavi konekciju sa android uređajem jest, da dohvati sve kontakte bili oni spremljeni na SIM karticu ili na sam uređaj. Klikom na dugme *CONTACTS* šalje se zahtjev sa klijenta prema poslužitelju kako bi se prikupila imena i brojevi svih kontakta te se poslali natrag klijentu. Kao i kod uspostavljanja

konekcije, ova funkcionalnost je podijeljena u dva dijela: slanje zahtjeva za kontaktima i prelazak u fazu slušanja na odgovor i ispis primljenih kontakata. U ispisu Ispis 3-4 može se vidjeti cijeli kôd metode koja se poziva prilikom klika na dugme *CONTACTS*.

```
private void contactsBtnActionPerformed(java.awt.event.ActionEvent evt) {
	cardLayout.show(tabsPanel, "contactsCard");

	if (!tableIsEmpty(contactsTable)){
		return;
	}

	String command = "getContactsList";
	sendStringMessageToAndroidDevice(command);
	waitContactsFromAndroidDevice();
}
```

Ispis 3-4: Definicija metode contactsBtnActionPerformed

Prvo se u fokus stavlja prozor u kojem se nalazi tablica u kojoj će biti spremljeni primljeni kontakti, potom se provjerava je li kojim slučajem tablica sa kontaktima već popunjena, ako jest neće se ponovno slati zahtjev prema poslužitelju jer to nepotrebno uzima vrijeme dok se poziv pošalje, obradi i vrati natrag, već ako je tablica popunjena samo će se prikazati već dostupni podaci, odnosno podaci pristigli u pozivu kad je prvi put poslan zahtjev za dohvaćanjem kontakta.

Potom slijedi formiranje naredbe, koja je u ovom slučaju proizvoljno odabrana da bude naziva *getContactsList* i onda se ta naredba prosljeđuje već opisanoj metodu *sendStringMessageToAndroidDevice* (Ispis 3-4).

Na kraju klijent prelazi u fazu slušanja, odnosno čekanja na odgovor poslužitelja. Prelazak u fazu slušanja i obrada pristiglih podataka je prikazan u ispisu Ispis 3-5.

```
private void waitContactsFromAndroidDevice() {
	try {
		ServerSocket ss = new ServerSocket(7800);
		Socket s = ss.accept();
		InputStreamReader isr = new InputStreamReader(
							s.getInputStream());
		BufferedReader br = new BufferedReader(isr);
		DefaultTableModel model = (DefaultTableModel)
						contactsTable.getModel();
		String message = "";

		while ((message = br.readLine()) != null) {
			String newMessage = message.substring(1);

			if (newMessage.isEmpty() == true) {
				break;
```

```
                    }

                    String[] arrOfStr = newMessage.split(" - ");
                    if (arrOfStr.length == 2) {
                          arrOfStr[0] = arrOfStr[0].trim();
                          arrOfStr[1] = arrOfStr[1].trim();

                          if (!existsInTable(contactsTable, arrOfStr)) {
                                model.addRow(arrOfStr);
                          }
                    }
              }

              isr.close();
              br.close();
              s.close();
              ss.close();
        } catch (IOException e) {
               e.printStackTrace();
        }
}
```

Ispis 3-5: Definicija metode waitContactsFromAndroidDevice

Računalna aplikacija ulazi u fazu slušanja na ulazu 7800, te kada poruka od poslužitelja stigne, slijedi njeno raščlanjivanje jer kontakti u poruci su složeni na način prikazan u ispisu Ispis 3-6:

```
IME - 555333
IME2 - 123456
IME3 - 965856
IME4 - 0998568568
...
```

Ispis 3-6: Primjer zapisa kontakta u poruci sa poslužitelja prema klijentu

Stoga se ide red po red te se poruka rastavlja na dva dijela: Ime i broj. Nakon provjere jesu li broj i ime već dodani u tablicu, slijedi dodavanje u *contactsTable* koja predstavlja tablicu koja prikazuje sva imena i brojeve, kao što je prikazano na slici Slika 3-3

Slika 3-3: Prikaz ispisa kontakata

Iznad tablice u desnom kutu se nalazi i dugme *Refresh* koje služi da bi se ponovno poslao zahtjev za dohvaćanjem svih kontakta i prikazali ih tablici *contactsTable*. Ispis 3-7 prikazuje kôd koji se pokreće klikom na dugme *Refresh*. Metoda je gotovo ista kao i kod inicijalnog zatraživanja kontakata, samo se razlikuje po tome što se prvo pobriše sve kontakte iz tablice, kako bi tablica bila čista i spremna za popunjavanje kad stignu novi kontakti.

```
private void contactsRefreshBtnActionPerformed(
                                        java.awt.event.ActionEvent evt) {

        if (!tableIsEmpty(contactsTable)){
               DefaultTableModel dm = (DefaultTableModel)
                                                contactsTable.getModel();
               int rowCount = dm.getRowCount();
               for (int i = rowCount - 1; i >= 0; i--) {
                      dm.removeRow(i);
               }
       }

       String command = "getContactsList";
       sendStringMessageToAndroidDevice(command);
       waitContactsFromAndroidDevice();
}
```

Ispis 3-7: Definicija metode contactsRefreshBtnActionPerformed

3.3. Dohvaćanje imena datoteka sa poslužitelja

Računalna aplikacija dopušta korisniku da dohvati imena svih slika, audio datoteka, videa i svih dokumenata sa android uređaja. Obzirom da je način dohvaćanja i obrade primljene poruke identičan za sva četiri tipa datoteke, u ovom poglavlju će se obraditi samo način dohvaćanja imena slika, uz naznaku šta je to različito kod dohvaćanja imena audio datoteka, videa i dokumenata.

Klikom na dugme *PICTURES* pokreće se kôd definiran u metodi *picturesBtnActionPerformed* čiji kôd je vidljiv u ispisu Ispis 3-8.

```
private void picturesBtnActionPerformed(java.awt.event.ActionEvent evt) {
      cardLayout.show(tabsPanel, "picturesCard");

      if (!tableIsEmpty(picturesTable)){
            return;
      }

      String command = "getImageFiles";
      sendStringMessageToAndroidDevice(command);
      waitFilesFromAndroidDevice("image");
}
```

Ispis 3-8: Definicija metode picturesBtnActionPerformed

U ispisu Ispis 3-8 se može vidjeti da su neke stvari podcrtane. One označuju stvari koje su različite ovisno o tome žele li se dohvatiti imena audio datoteke, videa ili dokumenti. Prva linija kôda u metodi: *cardLayout.show(tabsPanel, "picturesCard");* ima funkciju da u fokus stavi prikaz tablice odnosno zaslona sa tablicom onih datoteka čija imena se želi dohvatiti. Potom slijedi provjera je li tablica možda već popunjena, ako jest neće se ponovno pozivati dohvaćanje imena datoteka, a ako tablica nije popunjena slijedi formiranje naredbe i slanje iste prema poslužitelju.

Naredba za dohvaćanje imena slika je *getImageFiles,* naredba za imena audio datoteka je *getAudioFiles*, za dokumente *getDocumentFiles*, a za vdiea *getVideoFiles.* Za slanje naredbe se koristi već opisana metoda *sendStringMessageToAndroidDevice.*

Kao i kod dohvaćanja kontakata, nakon slanja naredbe, klijent prelazi u fazu slušanja na odgovor od poslužitelja i to je definirano u metodi *waitFilesFromAndroidDevice* čiji kôd je vidljiv u ispisu Ispis 3-9 . Ovisno o tome koji se tip datoteka dohvaća, metodi

waitFilesFromAndroidDevice će se proslijediti jedan od četiri tipa datoteke, kako bi se moglo odrediti koju tablicu treba popuniti pristiglim podacima.

```
private void waitFilesFromAndroidDevice(String type) {
      try {
            ServerSocket ss = new ServerSocket(7800);
            Socket s = ss.accept();
            InputStreamReader isr = new InputStreamReader(
                                                      s.getInputStream());
            BufferedReader br = new BufferedReader(isr);
            String message = "";
            DefaultTableModel model;

            switch (type) {
                case "image":
                    model = (DefaultTableModel) picturesTable.getModel();
                    break;
                case "audio":
                    model = (DefaultTableModel) musicTable.getModel();
                    break;
                case "video":
                    model = (DefaultTableModel) videosTable.getModel();
                    break;
                default:
                    model = (DefaultTableModel) documentsTable.
                                                          getModel();
                    break;
            }

            while ((message = br.readLine()) != null) {
                  String newMessage = message.substring(1);
                  if (newMessage.isEmpty() == true) {
                        break;
                  }

                  String[] arrOfStr = newMessage.split("~, ");

                  for (String file : arrOfStr) {
                        model.insertRow(
                              model.getRowCount(), new String[] {file});
                  }
            }

            isr.close();
            br.close();
            s.close();
            ss.close();
      } catch (IOException e) {
            e.printStackTrace();
      }
}
```

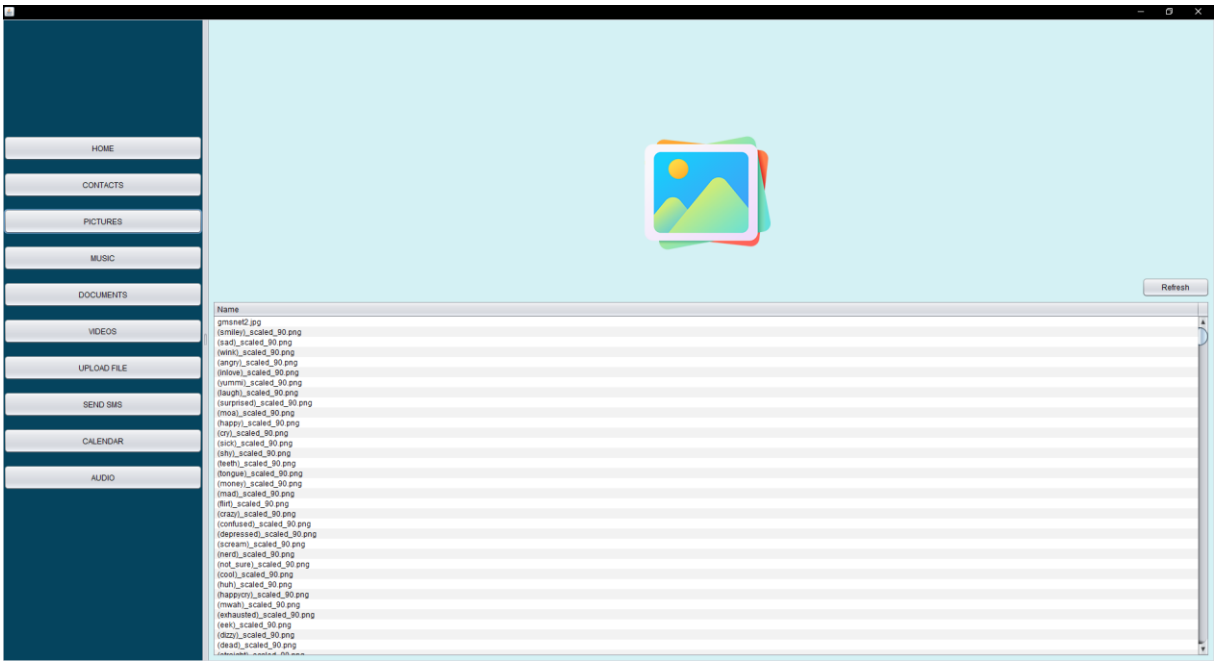
Ispis 3-9: Definicija metode waitFilesFromAndroidDevice

Imena datoteka (neovisno o tipu) sa poslužitelja stižu u formatu prikazanom i ispisu Ispis 3-10:

```
ime_datoteke~, ime_datoteke2~, ime_datoteke3~, ...
```

Ispis 3-10: Primjer formata poruke u kojem stižu imena datoteka

Stoga kada imena stignu potrebno ih je raščlaniti tako da se znakovi „~, “ izbace, a da se samo imena datoteka stavljaju u niz *arrOfStr* kroz koji se kasnije iterira i svako ime se zapisuje u predviđenu tablicu. Izgled popunjene tablice može se vidjeti na slici Slika 3-4.

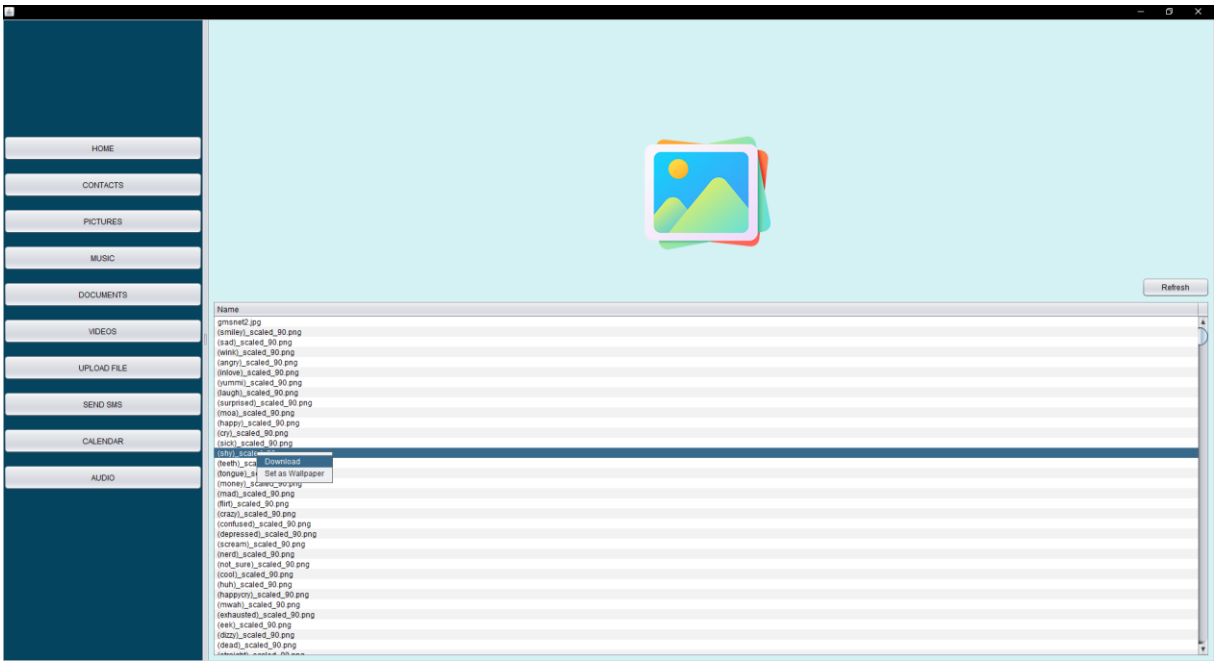


Slika 3-4: Prikaz dohvaćenih imena slika

Kao i kod funkcionalnosti za dohvaćanje liste kontakta, i kod dohvaćanja imena datoteka postoji mogućnost osvježavanja tablice, klikom na dugme *Refresh* u gornjem desnom kutu iznad same tablice. Kôd je gotovo isti kao i kod klika na dugme *Refresh* (Ispis 3-7) za osvježiti listu kontakata, jedina razlika je što za ulazak u fazu slušanja se ne koristi metoda *waitContactsFromAndroidDevice* već metoda *waitFilesFromAndroidDevice.*

3.3.1. Preuzimanje datoteke

Kada je tablica imena bilo kojeg tipa podataka popunjena, korisnik ima mogućnost desnim klikom miša na pojedino ime datoteke iz padajućeg izbornika odabrati opciju *Download.* Tom funkcionalnošću moguće je preuzeti bilo koju sliku, video, dokument ili audio zapis sa android uređaja na računalo. Prikaz padajućeg izbornika sa opcijom *Download* prikazan je na slici Slika 3-5 .



Slika 3-5: Prikaz opcije preuzimanja slike

Kôd koji se izvršava klikom na dugme *Download* iz padajućeg izbornika je prikazan u ispisu Ispis 3-11.

```
private void picturesOptionActionPerformed(
                                              java.awt.event.ActionEvent evt) {
       int row = picturesTable.getSelectedRow();
       if (row != -1) {
             String name = picturesTable.getValueAt(row, 0).toString();
             downloadFile(name, "imageFile");
       }
}
```

Ispis 3-11: Definicija metode koja se pokreće klikom na Download

U metodi *picturesOptionActionPerformed* se iz označenog retka tablice izvlači ime datoteke koju korisnik želi preuzeti, te se to ime skupa sa tipom datoteke (u ovom slučaju za slike je *imageFile*) prosljeđuje u metodu *downloadFile* koja potom kreira naredbu i šalje ju poslužitelju te ponovno ulazi u fazu slušanja na odgovor. Definicija metode *downloadFile* vidljiva je u ispisu Ispis 3-12.

```
public void downloadFile(String nameofFile, String fileType) {
	String command = "download~" + fileType + "~" + nameofFile;
	sendStringMessageToAndroidDevice(command);

	Socket clientSocket;
	ServerSocket serverSocket = null;
	int bytesRead;

	try {
		serverSocket = new ServerSocket(7800);
		clientSocket = serverSocket.accept();
		InputStream in = clientSocket.getInputStream();

		DataInputStream clientData = new DataInputStream(in);

		String fileName = clientData.readUTF();
		OutputStream output = new FileOutputStream(fileName);
		long size = clientData.readLong();

		byte[] buffer = new byte[1024];
		while (size > 0 && (bytesRead = clientData.read(
			buffer, 0, (int)Math.min(buffer.length, size))) != -1) {
				output.write(buffer, 0, bytesRead);
				size -= bytesRead;
		}

		output.close();
		in.close();
		clientData.close();
		clientSocket.close();
		serverSocket.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

Ispis 3-12: Definicija metode downloadFile [4]

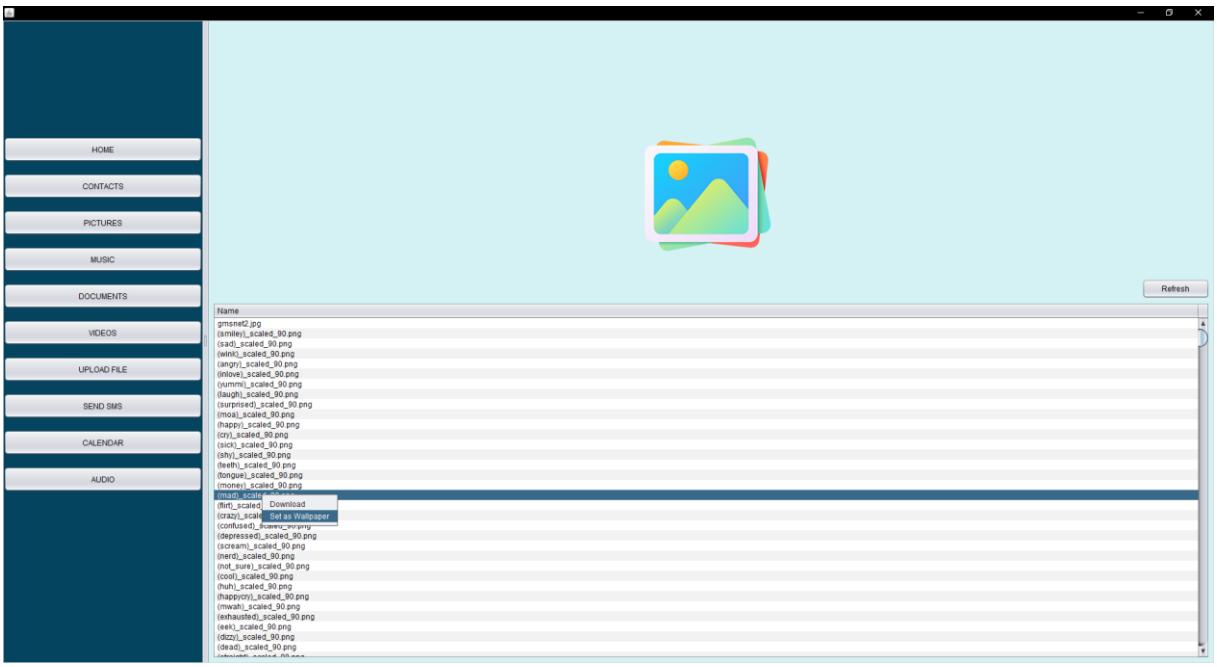
Razlog ovakvog izgleda naredbe je taj da bi se uspostavio vid standarda između klijenta i poslužitelja, u kojem znak ~ služi za razdvajanje dijelova poruke. Kad čitava poruka dođe na poslužiteljsku stranu u obliku samo jedne tekstualne poruke, dio poruke do prvog pojavljivanja znaka ~ predstavlja klijentsku naredbu, a onda svaki idući dio poruke do sljedećeg pojavljivanja znaka ~ predstavlja neki od dodatnih argumenata potrebnih za izvršavanje naredbe.

Faza slušanja na odgovor poslužitelja u ovom slučaju je neznatno drugačija od faze slušanja u dosadašnjim slučajevima (npr. čekanje na odgovor kod zahtjeva za kontaktima ili imenima datoteka). U ovom slučaju prvo se pročita naziv datoteke koja se prima iz toka (engl. *stream*) podataka, potom se čita veličina datoteke, a onda čitaju bajtovi same datoteke i spremaju na računalo.

3.3.2. Postavljanje slike zaslona android uređaja

Jedna dodatna mogućnost (pored mogućnosti preuzimanja datoteke) koju tablica imena svih slika nudi jest mogućnost postavljanja bilo koje slike kao slike zaslona android uređaja.

Desnim klikom na bilo koje ime slike u tablici imena svih slika, otvara se padajući izbornik u kojem se nudi opcija *Set as Wallpaper* što je vidljivo na slici Slika 3-6.



Slika 3-6: Prikaz mogućnosti postavljanja slike zaslona

Kôd koji se pokreće klikom na opciju *Set as Wallpaper* prikazan je u ispisu Ispis 3-13

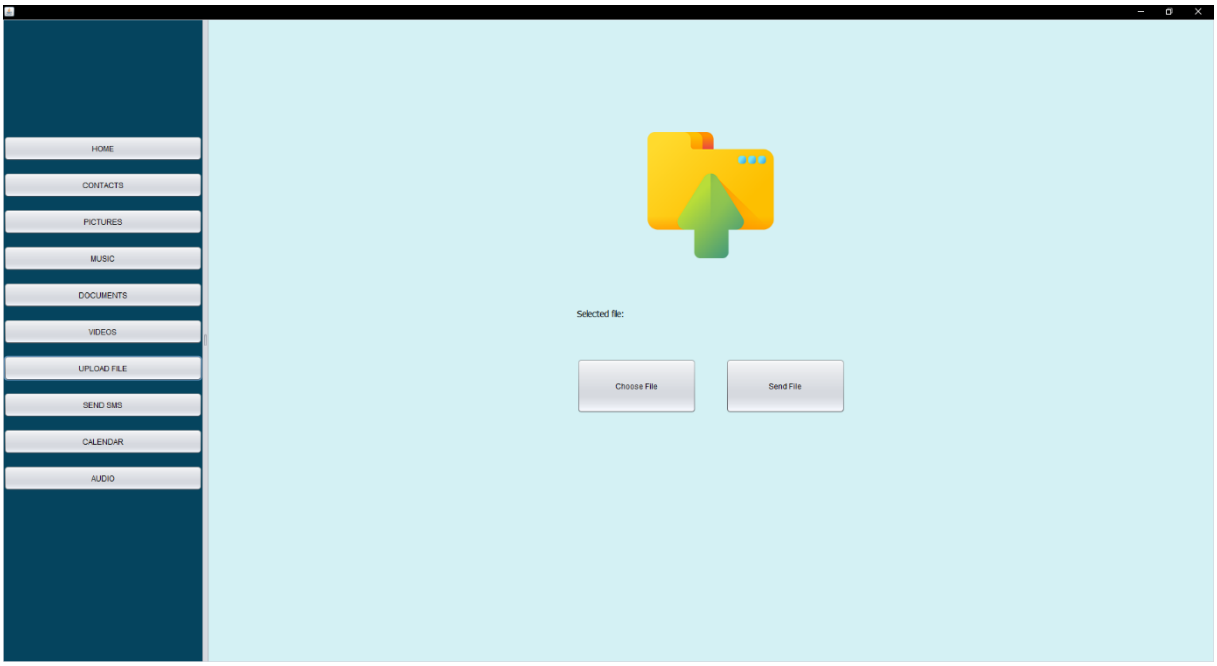
```
private void setAsWallpaperOptionActionPerformed(
                                          java.awt.event.ActionEvent evt) {
       int row = picturesTable.getSelectedRow();
       if (row != -1) {
              String name = picturesTable.getValueAt(row, 0).toString();
              String command = "setWallpaper~" + name;
              sendStringMessageToAndroidDevice(command);
       }
}
```

Ispis 3-13: Definicija metode setAsWallpaperOptionActionPerformed

Najprije se dohvati redak koji je označen, potom se iz označenog retka izvlači ime same datoteke i formira naredba koja glasi: setWallpaper~ime_datoteke te se koristi već opisana metoda *sendStringMessageToAndroidDevice* kako bi se naredba poslala poslužitelju.

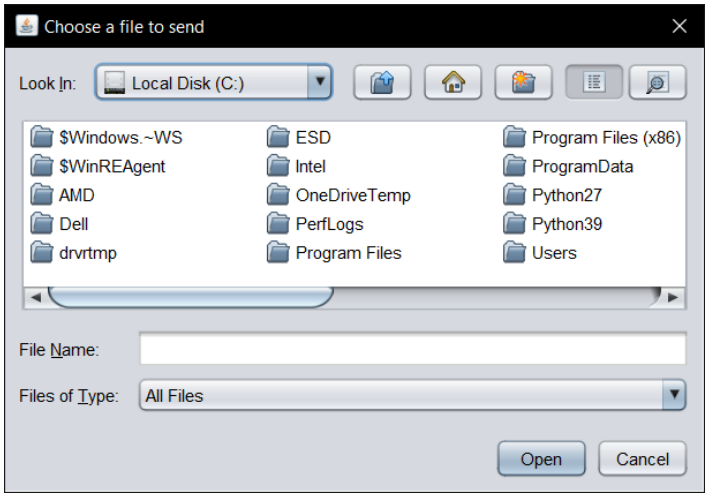
3.4. Slanje datoteke na poslužitelj

Pored opcije preuzimanja datoteka, računalna aplikacija korisniku nudi opciju da pošalje bilo koju datoteku sa računala na android uređaj. Zaslon koji se prikazuje korisniku klikom na dugme *UPLOAD FILE* iz izbornika, prikazan je na slici Slika 3-7.



Slika 3-7: Prikaz zaslona za slanje datoteke na android uređaj

Na zaslonu za slanje datoteka nalaze se dva dugmeta. Klikom na dugme *Choose File* korisniku se otvara skočni prozor koji korisniku nudi da izabere bilo koju datoteku sa svog računala koju želi poslati na poslužitelja. Izgled skočnog prozora je na slici Slika 3-8.



Slika 3-8: Prikaz skočnog prozora za odabir datoteke za slanje

Kôd koji se pokreće prilikom klika na dugme *Choose File* prikazan je u ispisu Ispis 3-14.

```
private void chooseFileBtnMouseReleased(java.awt.event.MouseEvent evt) {
      JFileChooser jFileChooser = new JFileChooser();
      jFileChooser.setDialogTitle("Choose a file to send");

      if (jFileChooser.showOpenDialog(null) ==
                                          jFileChooser.APPROVE_OPTION) {
               fileToSend[0] = jFileChooser.getSelectedFile();
               nameOfTheFileLabel.setText(fileToSend[0].getName());
      }
}
```

Ispis 3-14: Definicija metode chooseFileBtnMouseReleased

U slučaju da se neka datoteka odabere u skočnom prozoru, ime datoteke se zapisuje na zaslonu pored teksta *Selected file:* kako bi korisniku bilo jasno naznačeno koja je datoteka odabrana. Potom je potrebno kliknuti dugme *Send* kako bi se datoteka poslala na poslužitelj. Kôd koji vrši slanje datoteke na poslužitelj prikazan je u ispisu Ispis 3-15

```
private void sendFileBtnMouseReleased(java.awt.event.MouseEvent evt) {
      if (fileToSend[0] == null) {
            nameOfTheFileLabel.setText("You have to select the file!");
            return;
      }

      // send first message so android device
      // will know that new file will be sent
      String command = "newFile";
      sendStringMessageToAndroidDevice(command);

      // send File
      Socket s;
      try {
            s = new Socket(IP, 7801);
            FileInputStream fileInputStream = new FileInputStream(
                                        fileToSend[0].getAbsoluteFile());
            DataOutputStream dataOutputStream = new DataOutputStream(
                                        s.getOutputStream());

            String fileName = fileToSend[0].getName();
            byte[] fileNameBytes = fileName.getBytes();

            byte[] fileContentBytes = new byte[
                                       (int)fileToSend[0].length()];
            fileInputStream.read(fileContentBytes);

            dataOutputStream.writeInt(fileNameBytes.length);
            dataOutputStream.write(fileNameBytes);

            dataOutputStream.writeInt(fileContentBytes.length);
            dataOutputStream.write(fileContentBytes);

            fileInputStream.close();
            dataOutputStream.close();
            s.close();

            // set everything to default
            fileToSend[0] = null;
            nameOfTheFileLabel.setText("");
            sentFileResultLabel.setText("FILE SENT SUCCESSFULLY");

      } catch (IOException ex) {
            sentFileResultLabel.setText("FILE SENT UNSUCCESSFULLY");
               Logger.getLogger(NewJFrame.class.getName()).
                                          log(Level.SEVERE, null, ex);
      }
}
```

Ispis 3-15: Definicija metode sendFileBtnMouseReleased

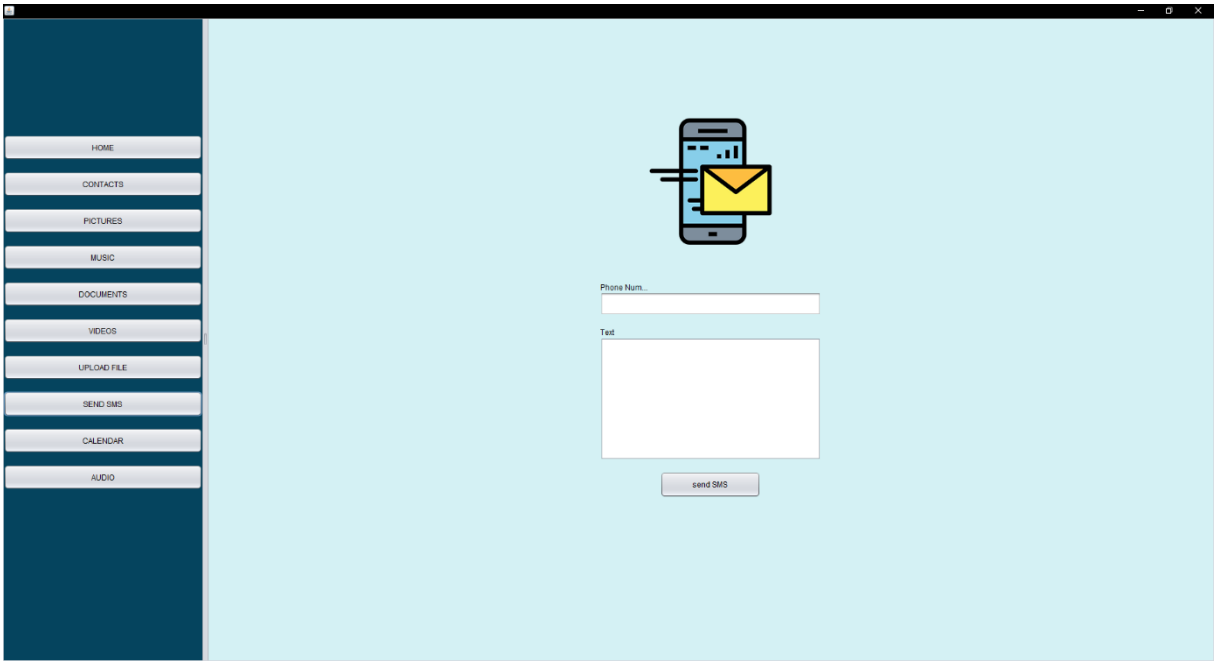
Ova metoda se također sastoji od dva dijela, no u ovom slučaju za razliku od prethodnih, u drugom dijelu se ne prelazi u fazu slušanja na odgovor. Prvi dio je identičan, formiranje naredbe koja glasi *newFile* i slanje na poslužitelj, a potom slijedi slanje odabrane datoteke na poslužitelj. Razlog tog načina pristupa leži u tome što na poslužitelju drugačije izgleda proces primanja samo obične poruke tekstualnog oblika od načina primanja datoteke, stoga sa prvom

porukom u kojoj se samo šalje tekstualna poruka *newFile* poslužitelju se daje do znanja da će iduća poruka biti ustvari datoteka te da za primanje sljedeće poruke inicijalizira druge varijable potrebne za prihvaćanje datoteke, a ne tekstualne poruke.

Slanje datoteke se odvija u segmentima tako da se prvo klijent spoji na utičnicu poslužitelja preko IP adrese poslužitelja i ulaza na kojem poslužitelj čeka poruke (7801). Potom se otvori tok podataka u kojeg se upisuje ime datoteke koja se šalje, veličina imena datoteke, veličina datoteke te sama datoteka. Na kraju kad je sve poslano ka poslužitelju, tok podataka skupa sa utičnicom se zatvara, a tekstualno polje u kojem bude zapisano ime datoteke koja se šalje, postavlja se na zadanu vrijednost (prazno polje) i ispisuje poruka da je datoteka uspješno poslana.

3.5. Slanje SMS poruke

Funkcionalnosti slanja SMS poruke moguće je pristupiti na dva načina. Prvi način je klikom na dugme *SEND SMS* iz izbornika, a drugi način je desnim klikom na ime ili broj nekog kontakta spremljenog u tablicu kontakata koja je prikazana kada se klikne dugme *CONTACTS* i odabirom opcije *Send SMS* iz padajućeg izbornika. Kojim god načinom se pokrene ova funkcionalnost biti će prikazan zaslon za slanje SMS poruke, zaslon izgleda kao što je na slici Slika 3-9.



Slika 3-9: Prikaz zaslona za slanje SMS poruke

Sastoji se od dva obavezna polja: *Phone Number* i *Text* te dugmeta *send SMS*. Ako se u ovaj prikaz dođe preko prikaza *CONTACTS* i označenog kontakta, onda će polje *Phone Number* biti već popunjeno sa brojem za kojeg se odabere opcija *Send SMS*. Nakon popunjavanja svih polja i klika dugmeta *send SMS* poziva se metoda *sendSMSBtnMouseReleased* čiji kôd se može vidjeti u ispisu Ispis 3-16.

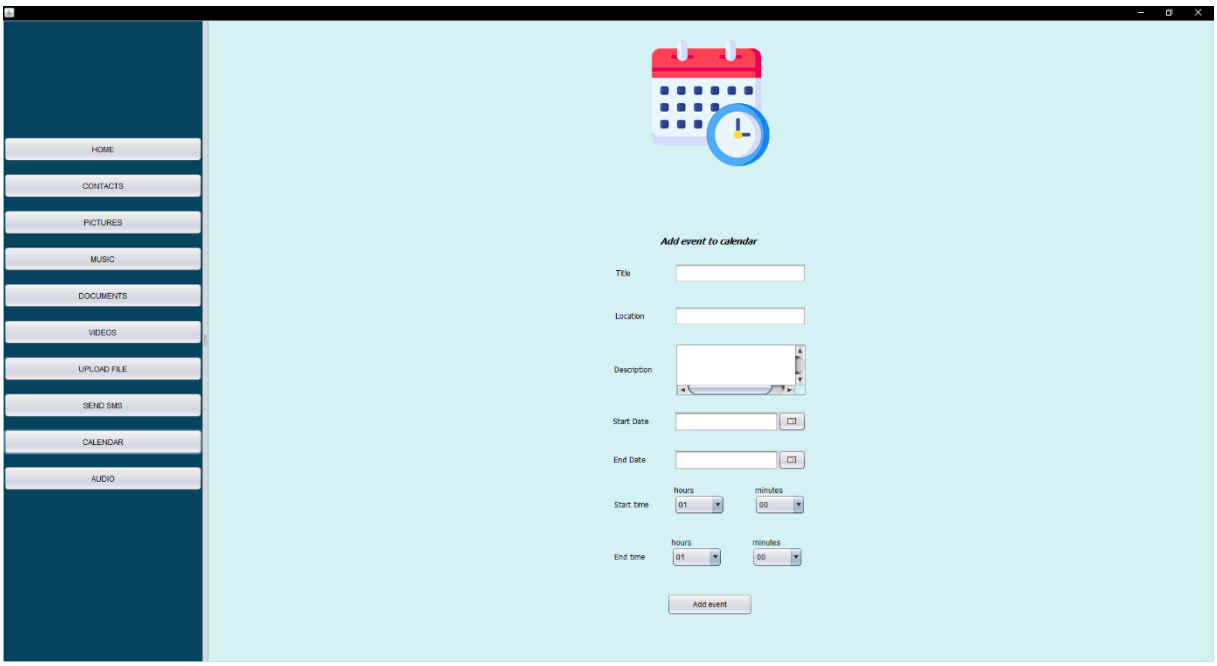
```
private void sendSMSBtnMouseReleased(java.awt.event.MouseEvent evt) {
      String command = "sendSMS";
      if (sendSMSNumber.getText().isEmpty()) {
            sendSMSNumber.setText("Number is required");
            return;
      } else if (sendSMSTextArea.getText().isEmpty()) {
            sendSMSTextArea.setText("Text is required");
            return;
      }

      String messageToSend = command + "~" +
                             sendSMSNumber.getText() + "~" +
                             sendSMSTextArea.getText();

      sendStringMessageToAndroidDevice(messageToSend);
}
```

Ispis 3-16: Definicija metode sendSMSBtnMouseReleased

U metodi *sendSMSBtnMouseReleased* se iščitavaju vrijednosti iz *Phone Number* i *Text* polja, potom se formira naredba oblika: sendSMS~broj_kontakta~text_SMS_poruke i šalje ka poslužitelju.

3.6. Dodavanje događaja u kalendar

Poželi li korisnik dodati novi događaj u svoj kalendar na android uređaju, to je moguće izvesti iz prikaza koji se otvara klikom na dugme *CALENDAR*. Na tom prikazu nalazi se forma koju je potrebno popuniti kako bi se događaj spremio u kalendar. Ponuđena polja za unos podataka su *Title* (naslov događaja), *Location* (lokacija događaja), *Description* (opis), *Start Date* (datum početka događaja), *End Date* (datum završetka događaja), *Start time* (sati i minute početka događaja), *End time* (sati i minute završetka događaja). Polja *Title, Start Date i End Date* su obavezna (polja *Start time* i *End time* su također obavezna, ali ona imaju zadanu vrijednost od 01:00 sati). Izgled forme prikazan je na slici Slika 3-10.



Slika 3-10: Prikaz zaslona za dodavanje novog događaja u kalendar

Klikom na dugme *Add Event* koje se nalazi na dnu forme poziva se metoda *calendarAddEventBtnMouseReleased* čiji kôd se nalazi u ispisu Ispis 3-17.

```
private void calendarAddEventBtnMouseReleased(
                                        java.awt.event.MouseEvent evt) {
      String command = "addEventInCalendar~";

      String title = calendarTitleTextField.getText();
      String location = calendarLocationTextField.getText();
      String description = calendarDescriptionTextArea.getText();
      String startHours = startHoursComboBox.
                                       getSelectedItem().toString();
      String startMinutes = startMinutesComboBox.
                                       getSelectedItem().toString();
      String endHours = endHoursComboBox.getSelectedItem().toString();
      String endMinutes = endMinutesComboBox.
                                       getSelectedItem().toString();

      if ((title.isEmpty()) ||
          (calendarEndDateChooser.getDate() == null) ||
          (calendarStartDateChooser.getDate() == null)) {
            calendarTitleTextField.setBorder(
                        BorderFactory.createLineBorder(Color.RED, 2));
            calendarStartDateChooser.setBorder(
                        BorderFactory.createLineBorder(Color.RED, 2));
            calendarEndDateChooser.setBorder(
                        BorderFactory.createLineBorder(Color.RED, 2));
            return;
      }


      DateFormat dateFormatStart = new SimpleDateFormat("yyyy/MM/dd");
      String dateStartString = dateFormatStart.format(
                                    calendarStartDateChooser.getDate());
      String dateStart = dateStartString + " " +
                         startHours + ":" + startMinutes;

      DateFormat dateFormatEnd = new SimpleDateFormat("yyyy/MM/dd");
      String dateEndString = dateFormatEnd.format(
                                    calendarEndDateChooser.getDate());
      String dateEnd = dateEndString + " " + endHours + ":" + endMinutes;

      LocalDateTime localDateTimeStart = LocalDateTime.parse(
             dateStart, DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm"));
      LocalDateTime localDateTimeEnd = LocalDateTime.parse(
             dateEnd, DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm"));
      long millisStart = localDateTimeStart.atZone(
             ZoneId.systemDefault()).toInstant().toEpochMilli();
      long millisEnd = localDateTimeEnd.atZone(
             ZoneId.systemDefault()).toInstant().toEpochMilli();
      if (millisEnd < millisStart) {
            jLabel20.setText("End date can't be before start date");
            return;
      }

      if (location.isEmpty()) {
            location = "!";
      }

      if (description.isEmpty()) {
            description = "!";
      }
```

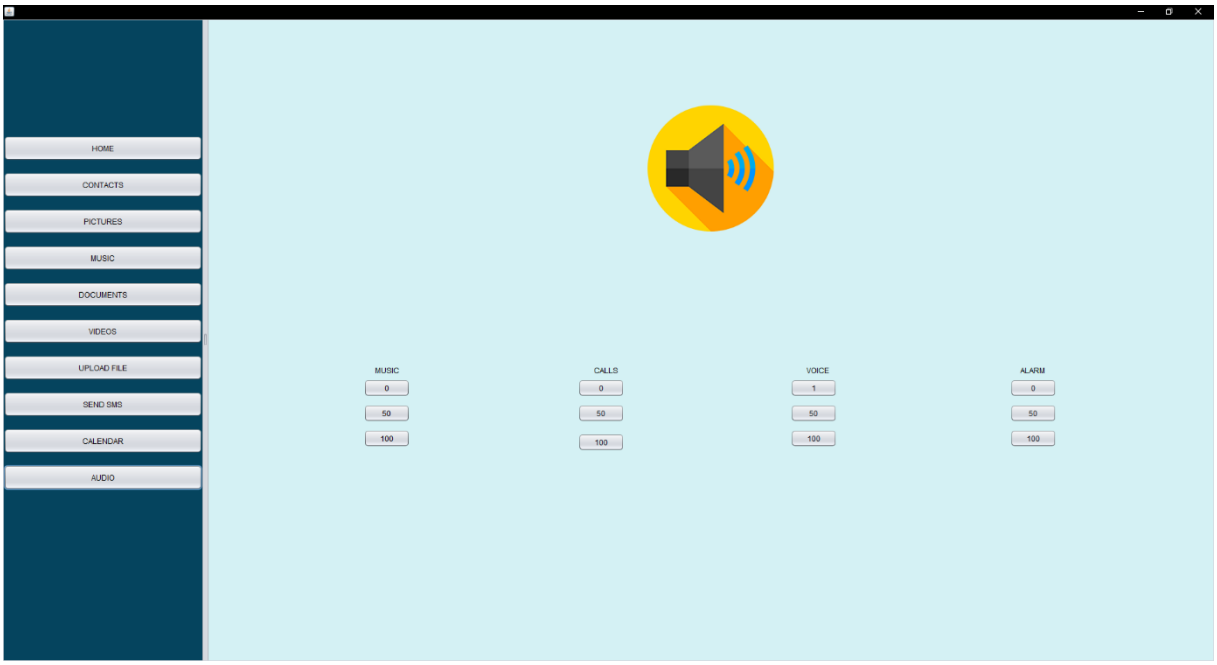
```
        String message = command + title + "~" +
                         location + "~" + description + "~" +
                         millisStart + "~" + millisEnd;

        sendStringMessageToAndroidDevice(message);
        jLabel20.setText("Event added to calendar");
}
```

Ispis 3-17: Definicija metode calendarAddEventBtnMouseReleased

Osim povlačenja vrijednosti koje su unesene u polja za unos podataka na samom prikazu, prije formiranja poruke koja će biti poslana poslužitelju rade se dodatno provjere jesu li obavezna polja (*Title, Start Date, End Date*) popunjena te se zatim datum, sati i minute spajaju i pretvaraju u sekunde (kao startna točka je odabran datum 1. siječnja 1970., 24:00) jer taj format zahtjeva poslužitelj kako bi događaj bio dodan u kalendar.

3.7. Kontrola zvuka

Korisniku se klikom na dugme *AUDIO* u izborniku otvara prikaz u kojem može kontrolirati glasnoću glazbe, alarma, glasnoću obavijesti dolaznog poziva i drugih obavijesti te glasnoću zvuka glasa sugovornika u pozivu. Izgled prikaza se vidi na slici Slika 3-11.



Slika 3-11: Prikaz zaslona za kontrolu glasnoće

Svaki tip glasnoće ima ponuđene opcije da se stiša odnosno pojača na razinu 0, 50% maksimalne glasnoće ili 100% maksimalne glasnoće. Glasnoća zvuka sugovornika u toku

poziva se ne može stišati do nule, stoga za tu glasnoću minimalna vrijednost je 1% maksimalne glasnoće. Kôd za bilo koju od opcija je gotovo isti, samo je razlika u dodatnim parametrima koji se šalju uz glavnu naredbu. U ispisu Ispis 3-18 je vidljiv kôd za stišavanje glasnoće glazbe na 0.

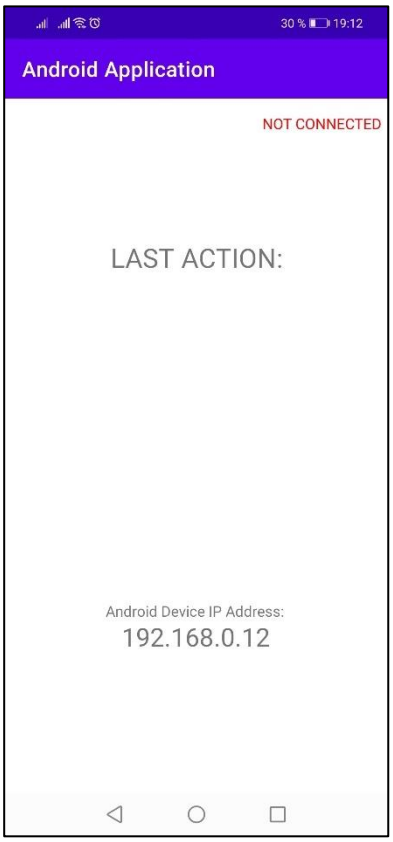
```
private void music0BtnMouseReleased(java.awt.event.MouseEvent evt) {
      String command = "adjustVolume~AUDIO~0";
      sendStringMessageToAndroidDevice(command);
}
```

Ispis 3-18: Definicija metode music0BtnMouseReleased

Podcrtani dio u ispisu je ono što se mijenja od parametara ovisno koji tip glasnoće se želi modulirati i na koliki iznos. Za glazbu je drugi parametar *AUDIO*, za glasnoću sugovornika u pozivu je *VOICE*, za alarm je *ALARM*, a za glasnoću obavijesti je *CALLS*.

4. Android aplikacija

Android aplikacija u ovom radu predstavlja poslužiteljsku aplikaciju, odnosno aplikaciju koja izvršava naredbe stigle od klijenta (računalne aplikacije) i ovisno koja je naredba u pitanju, šalje odgovor natrag prema klijentu. Obzirom da se sve operacije izvršavaju bez potrebe interakcije sa korisnikom, grafičko sučelje je prilagođeno tako da se na dnu zaslona aplikacije prikazuje IP adresa android uređaja, a u desnom gornjem kutu je trenutni status: *CONNECTED* ili *NOT CONNECTED* ovisno je li konekcija sa klijentom uspostavljena. U fokusu zaslona je tekstualni prikaz zadnje akcije koja je izvršena nad android uređajem. Prikaz zaslona po ulasku u aplikaciju prikazan je na slici Slika 4-1.



Slika 4-1: Prikaz početnog zaslona android aplikacije

4.1. Potrebne dozvole

Kako bi naredbe koje stižu sa klijentske strane bile realizirane, za većinu akcija je potrebna neka od dozvola pristupa podacima (npr. pristup kontaktima, galeriji, kalendaru, itd.). Sve potrebne dozvole su deklarirane u AndroidManifest.xml datoteci [5], a vidljive su u ispisu Ispis 4-1.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_CALENDAR"/>
<uses-permission android:name="android.permission.SET_WALLPAPER" />
```

Ispis 4-1: Dozvole defnirane u AndroidManifest.xml

Neke od gore navedenih dozvola je potrebno eksplicitno zatražiti da ih korisnik odobri, stoga je u *MainActivity* klasu (klasu u kojoj se nalazi *main* metoda i iz koje kreće izvršavanje aplikacije) u metodu *onCreate* (metodu koja se izvršava prilikom pokretanja aplikacije) dodan dio kôda iz ispisa Ispis 4-2.

```
int PERMISSION_ALL = 1;
String[] PERMISSIONS = {
      android.Manifest.permission.READ_CONTACTS,
      android.Manifest.permission.SEND_SMS,
      android.Manifest.permission.READ_EXTERNAL_STORAGE,
      android.Manifest.permission.WRITE_EXTERNAL_STORAGE,
      android.Manifest.permission.WRITE_CALENDAR,
      android.Manifest.permission.SET_WALLPAPER,
};

if (!hasPermissions(this, PERMISSIONS)) {
      ActivityCompat.requestPermissions(
                           this, PERMISSIONS, PERMISSION_ALL);
}

...

private static boolean hasPermissions(
                                 Context context, String... permissions) {
      if (context != null && permissions != null) {
            for (String permission : permissions) {
                  if (ActivityCompat.checkSelfPermission(
             context, permission) != PackageManager.PERMISSION_GRANTED) {
                  return false;
                  }
            }
      }
      return true;
}
```

Ispis 4-2: Dio kôda zadužen za traženje dozvoli od korisnika

4.2. Koncept izvršavanja android aplikacije

Pored spomenute *MainActivity* klase, postoje još 3 klase kako bi komunikacija između klijenta i poslužitelja radila pravilno. Te tri klase su: *FileSender*, *MessageSender* i *MyServerThread*. Funckija klase *FileSender* je slati datoteke sa poslužitelja prema klijentu, dok je funkcija *MessageSender* klase slati jednostavnu tekstualnu poruku sa poslužitelja prema klijentu.

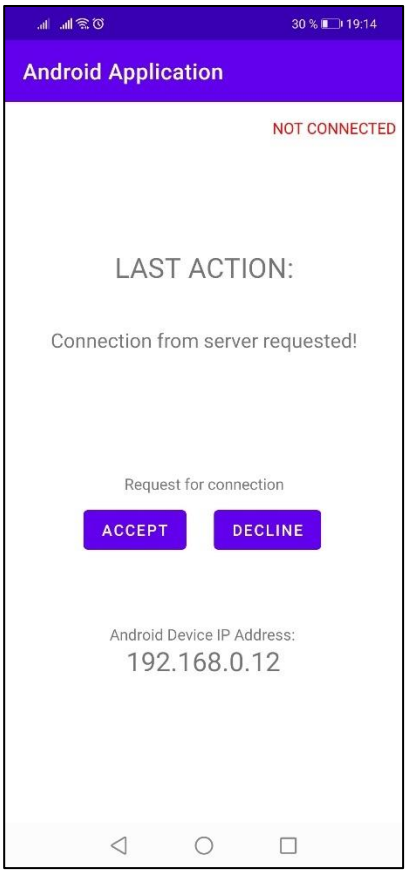
MyServerThread je klasa zadužena za primanje i obradu pristiglih naredbi sa klijenta. Radi na principu stavljanja poslužitelja u konstantnu fazu slušanja na ulaz 7801. Ako je stigla poruka od klijenta, razdvaja ju i uzima prvi element koji predstavlja naredbu koju klijent želi provesti. Ovisno koja naredba je u pitanju, određena metoda se poziva i metodi se prosljeđuju ostali parametri nastali nakon raščlanjivanja poruke (ako pozvana metoda te parametre zahtjeva). Glavna metoda *MyServerThread* klase, koja služi za razdvajanje pristigle poruke i određivanje koju metodu pozvati ovisno o iščitanoj naredbi, prikazana je u ispisu Ispis 4-3.

```
public void run() {
      String[] arr = message.split("~");
      if (!(arr.length > 0)) {
            Toast.makeText(MainActivity.this, "Received NO command
                              from server", Toast.LENGTH_SHORT).show();
            return;
      }

            String command = arr[0];
            if(command.equals("newFile")){
                  fileWillBeSent = true;
            } else if (command.equals("connect")){
                  connectRequest(arr);
            } else if(command.equals("getContactsList")){
                  sendContacts();
            }  else if (command.equals("sendSMS")){
                  sendSMS(arr);
            } else if (command.equals("getAudioFiles")){
                  String type = "audio";
                  sendFiles(type);
            } else if (command.equals("getImageFiles")){
                  String type = "image";
                  sendFiles(type);
            } else if (command.equals("getVideoFiles")){
                  String type = "video";
                  sendFiles(type);
            } else if (command.equals("getDocumentFiles")){
                  String type = "document";
                  sendFiles(type);
            } else if (command.equals("addEventInCalendar")){
                  addEventInCalendar(arr);
            } else if (command.equals("setWallpaper")){
                  setWallpaper(arr);
            } else if (command.equals("adjustVolume")){
                  adjustVolume(arr);
            } else if (command.equals("download")){
                  sendWantedFile(arr);
            }
}
```

Ispis 4-3: Definicija metode koja odlučuje koju metodu pozvat ovisno o klijentskoj naredbi

4.3. Prihvaćanje konekcije

Jedina interakcija sa korisnikom koja se zahtjeva na strani android aplikacije jest da kada klijentska strana zatraži uspostavu konekcije, korisniku se ponudi opcija prihvaćanja ili odbijanja konekcije u vidu dva dugmeta *ACCEPT* (ako želi prihvatiti konekciju) i *DECLINE* (ako želi odbiti konekciju). Zaslon koji se prikazuje korisniku kada stigne zahtjev za konekcijom prikazan je na slici Slika 4-2.



Slika 4-2: Prikaz zaslona za prihvaćanje ili odbijanje konekcije

Iz ispisa Ispis 4-3 može se vidjeti da kada stigne naredba *connect* pokreće se kôd definiran u metodi *connectRequest*, a čiji je kôd naveden u ispisu Ispis 4-4.

```
public void connectRequest(String[] serverMessage) {
      if (serverMessage.length != 2) {
            Toast.makeText(this, "Connection invalid,
                        there is no client IP", Toast.LENGTH_SHORT).show();
            return;
      }

      clientIP = serverMessage[1];

      lastActionTextView.setText("Connection from server requested!");
      connectionRequestTextView.setVisibility(View.VISIBLE);
      acceptBtn.setVisibility(View.VISIBLE);
      declineBtn.setVisibility(View.VISIBLE);
}
```

Ispis 4-4: Definicija metode connectRequest

Pored naredbe *connect* još jedan od parametara poruke je i IP adresa klijenta koja se sprema i prosljeđuje *MessageSender* klasi u momentu kada se šalje odgovor klijentu. Kad korisnik odabere jednu od ponuđenih opcija *ACCEPT* ili *DECLINE*, poziva se metoda

connectAccept (ako je odabrana opcija *ACCEPT)* ili *connectDecline* (ako je odabrana opcija *DECLINE*). Obje metode se identične, jedina je razlika u sadržaju poruke koji se šalje natrag klijentu. Poruka *accept* ili *decline* seodljeđuje se *MessageSender* klasi koja poruku šalje klijentu, a kôd slanja tekstualne poruke je prikazan u ispisu Ispis 4-5.

```
protected Void doInBackground(String... voids) {
      String message = voids[0];
      String IP = voids[1];

      try {
            s = new Socket(IP, 7800);
            pw = new PrintWriter(s.getOutputStream());
            pw.write(message);
            pw.flush();
            pw.close();
            s.close();
      } catch (IOException e) {}
      return null;
}
```

Ispis 4-5: Kôd koji ima zadaću poslati tekstualnu poruku klijentu

Najprije se pomoću spremljene IP adrese klijenta vrši spajanje na klijentsku utičnicu i ulaz 7800 na kojem klijent čeka odgovor te se potom poruka zapisuje u tok podataka.

4.4. Slanje liste kontakata

Kada sa klijentske strane stigne naredba *getContactsList*, poziva se metode *sendContacts*, čija je zadaća prikupiti listu kontakta spremljenih bilo na sam uređaj ili na SIM karticu te ih u obliku tekstualne poruke poslati klijentu. Kôd koji obavlja ovu funkcionalnost nalazi se u ispisu Ispis 4-6.

```
public void sendContacts() {
      Cursor cursor = getContentResolver().query(
                      ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                      null, null, null, null);
      ArrayList<String> listOfContacts = new ArrayList<>();

      while(cursor.moveToNext()){
            String name = cursor.getString(cursor.getColumnIndex(
                  ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));
            String mobile = cursor.getString(cursor.getColumnIndex(
                  ContactsContract.CommonDataKinds.Phone.NUMBER));

            listOfContacts.add(name + " - " + mobile + "\n");
      }

      MessageSender messageSender = new MessageSender();
      messageSender.execute(listOfContacts.toString(), clientIP);
      lastActionTextView.setText("List of contacts sent to server");
}
```

Ispis 4-6: Definicija metode sendContacts

Pokazivačem (engl. *cursor*) se pristupa bazi spremljenih kontakata te potom iterira pokazivačem i sprema kontakte u niz u formatu: „IME – BROJ“ te se na kraju taj niz šalje klasi *MessageSender* da u obliku tekstualne poruke pošalje odgovor klijentu.

4.5. Slanje SMS poruke

Na pristiglu klijentsku naredbu *sendSMS* poziva se metoda *sendSMS* kojoj se prosljeđuju i drugi parametri pristigle poruke, a to su broj na koji treba poslati poruku i tekst poruke. Zatim se u metodi *sendSMS* uz pomoć *SmsManager* klase (predefinirana android klasa) šalje SMS poruka na željeni broj. Kôd *sendSMS* metode je u ispisu Ispis 4-7.

```
public void sendSMS(String[] serverMessage) {
      if (serverMessage.length != 3) {
            Toast.makeText(this, "Can't send SMS, not enough
                              parameters!", Toast.LENGTH_SHORT).show();
            return;
      }

      String phoneNo = serverMessage[1];
      String message = serverMessage[2];

      try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(phoneNo, null,
                                    message, null, null);
            lastActionTextView.setText("Sent SMS to " + phoneNo);
      } catch (Exception e) {}
}
```

Ispis 4-7: Defincija metode sendSMS

4.6. Slanje imena datoteka

Neovisno o tipu datoteke, kada stigne klijentski zahtjev za imenima svih datoteka određenog tipa, poziva se ista metoda *sendNamesOfFiles*. Metodi se proslijedi dodatni parametar koji je stigao u poruci, a on označava tip datoteke za koji klijent želi popis imena. Definicija metode *sendNamesOfFiles* nalazi se u ispisu Ispis 4-8.

```
public void sendNamesOfFiles(String type) {
      List<File> files;
      ArrayList<String> nameOfFiles= new ArrayList<>();

      if (type.equals("image")){
            files = getListOfImageFiles(new File(Environment.
                        getExternalStorageDirectory().toString()));
      } else if (type.equals("audio")) {
            files = getListOfAudioFiles(new File(Environment.
                        getExternalStorageDirectory().toString()));
      } else if (type.equals("video")) {
            files = getListOfVideoFiles(new File(Environment.
                        getExternalStorageDirectory().toString()));
      } else {
            files = getListOfDocumentFiles(new File(Environment.
                        getExternalStorageDirectory().toString()));
      }

      for (int i = 0; i < files.size(); i++) {
           nameOfFiles.add(files.get(i).getName()+"~");
      }

      sendMessageToServer(nameOfFiles.toString(), clientIP);

      if (type.equals("image")){
           lastActionTextView.setText("Names of image
                                       files sent to server");
      } else if (type.equals("audio")) {
           lastActionTextView.setText("Names of audio
                                       files sent to server");
      } else if (type.equals("video")) {
           lastActionTextView.setText("Names of video
                                       files sent to server");
      } else {
           lastActionTextView.setText("Names of document
                                       files sent to server");
      }

}
```

Ispis 4-8: Definicija metode sendNamesOfFiles

Ovisno o kojem tipu datoteke se radi, pozvat će se metoda zadužena za dohvaćanje svih datoteka tog tipa. Potom se iz niza koji sadrži sve datoteke traženog tipa formira novi niz koji sadrži samo imena svih tih datoteka prvog niza i potom se taj niz šalje klasi *MessageSender* koja imena šalje klijentu. U ispisu Ispis 4-9 je prikazana metoda koja dohvaća sve datoteka tipa slika. Dijelovi koji su podcrtani označavaju razlika u odnosu na metode koje dohvaćaju druge tipove datoteka. Pa tako za slike se gleda završava li ime datoteke sa *.jpg* ili *.png*, za audio datoteke *.mp3*, za videa *.mp4*, a za dokumente: *.txt, .pdf, .docx, .ppt, .cvs*.

```
private List<File> getListOfImageFiles(File parentDir) {
      ArrayList<File> inFiles = new ArrayList<File>();
      File[] files = parentDir.listFiles();
      for (File file : files) {
            if (file.isDirectory()) {
                  inFiles.addAll(getListOfImageFiles(file));
            } else {
                  if((file.getName().endsWith(".jpg")) ||
                     (file.getName().endsWith(".png"))){
                           inFiles.add(file);
                  }
           }
      }
      return inFiles;
}
```

Ispis 4-9: Metoda koja dohvaća sve datoteke tipa slika

Metoda je napisana rekurzivno, tako da ulazi u svaku mapu na uređaju za koju ima dopuštenje ući (na uređaju i memorijskoj kartici, ako se ona nalazi u uređaju), te sprema u niz sve datoteke koje odgovaraju uvjetu da im imena završavaju traženim nastavkom i na kraju taj niz vraća u metodu *sendNamesOfFiles*.

4.7. Dodavanje događaja u kalendar

Ako korisnik pošalje naredbu *addEventInCalendar* sa pravilnim parametrima, pozvat će se metoda *addEventInCalendar*, čiji je kôd prikazan u ispisu Ispis 4-10.

```
public void addEventInCalendar(String[] serverMessage) {
      if (serverMessage.length != 6) {
            Toast.makeText(this, "Can't add event in calendar,
                     not enough parameters!", Toast.LENGTH_SHORT).show();
            return;
      }

      ContentResolver cr = this.getContentResolver();
      ContentValues cv = new ContentValues();

      cv.put(CalendarContract.Events.TITLE, serverMessage[1]);

      if (!serverMessage[3].equals("!")) {
            cv.put(CalendarContract.Events.DESCRIPTION,
                                                      serverMessage[3]);
      }

      if (!serverMessage[2].equals("!")) {
            cv.put(CalendarContract.Events.EVENT_LOCATION,
                                                       serverMessage[2]);
      }

      long start = Long.parseLong(serverMessage[4]);
      long end = Long.parseLong(serverMessage[5]);
      cv.put(CalendarContract.Events.DTSTART, start);
      cv.put(CalendarContract.Events.DTEND, end);

      cv.put(CalendarContract.Events.CALENDAR_ID, 1);
      cv.put(CalendarContract.Events.EVENT_TIMEZONE,
                           Calendar.getInstance().getTimeZone().getID());
      Uri uri = cr.insert(CalendarContract.Events.CONTENT_URI, cv);

      lastActionTextView.setText("Event added in calendar");
}
```

Ispis 4-10: Definicija metode addEventInCalendar

Koristeći se već definiranim klasama *ContentResolver* i *ContentValues* (definiranima od strane Androida), zapisuju se primljeni parametri događaja u klasu *ContentValues,* koja se na kraju preko *ContentResolver*-a, spaja na sam kalendar.

4.8. Postavljanje slike zaslona

Ako korisnik poželi promijeniti sliku zaslona na android uređaju i pošalje poruku sa klijentske aplikacije koja kao naredbu ima *setWallpaper,* a kao dodatni parametar ime slike koju želi postaviti za sliku zaslona, na poslužiteljskoj strani će se pokrenuti metoda *setWallpaper*, čiji kôd je prikazan u ispisu Ispis 4-11.

```
public void setWallpaper(String[] serverMessage) {
      if (serverMessage.length != 2) {
            Toast.makeText(this, "Can't set wallpaper,
                      not enough parameters!", Toast.LENGTH_SHORT).show();
            return;
      }

      String imageName = serverMessage[1];
      List<File> imageFiles = getListOfImageFiles(new File(Environment.
                              getExternalStorageDirectory().toString()));
      File picture = getFileFromList(imageFiles, imageName);
      Bitmap bitmap = BitmapFactory.
                                    decodeFile(picture.getAbsolutePath());
      WallpaperManager myWallpaperManager = WallpaperManager.
                                    getInstance(getApplicationContext());
      try {
            myWallpaperManager.setBitmap(bitmap);
            lastActionTextView.setText("Wallpaper changed");
      } catch (IOException e) {}
}
```

Ispis 4-11: Definicija metode setWallpaper

Najprije se iterira kroz memoriju poslužitelja i u nizu zapisuju sve datoteke čije ime završava sa nastavkom *.jpg* ili *.png*, a potom se onda izdvaja datoteka, odnosno slika sa imenom kojeg je klijent poslao. Koristeći predefiniranu klasu *WallpaperManager* pronađena slika se postavlja kao slika zaslona.

4.9. Prilagodba glasnoće

Kada poslužitelj primi poruku, čija naredba glasi *adjustVolume,* pozvat će se metoda *adjustVolume* kojoj će biti proslijeđeni ostali parametri poruke, a to su tip glasnoće koji se želi regulirati i razina na koju se glasnoća želi postaviti. Unutar *adjustVolume* metode ovisno o tipu glasnoće u parametru pristigle poruke, pozvat će se metoda *setVolumeForType* i proslijediti tip i vrijednost na koju se glasnoća treba postaviti. Kôd metode *setVolumeForType* je prikazan u ispisu Ispis 4-12.

```
private void setVolumeForType(int type, String value) {
      if (value.equals("0")) {
            if (type == AudioManager.STREAM_VOICE_CALL) {
                  audioManager.setStreamVolume(type,
                              audioManager.getStreamMaxVolume(type) /
                              audioManager.getStreamMaxVolume(type), 0);
            } else {
                  audioManager.setStreamVolume(type, 0, 0);
            }
     } else if (value.equals("50")) {
           audioManager.setStreamVolume(type,
                        audioManager.getStreamMaxVolume(type) / 2, 0);
     } else {
           audioManager.setStreamVolume(type,
                        audioManager.getStreamMaxVolume(type), 0);
    }
}
```

Ispis 4-12: Definicija metode setVolumeForType

Koristeći predefiniranu klasu *AudioManager* postavlja će određena razina glasnoće za željeni tip.

4.10. Slanje datoteke

U poruci sa klijentske strane, čija naredba glasi *download*, nalaze se dva dodatna parametra, jedan je tip datoteke, a drugi ime datoteke. Ta dva parametra se prosljeđuju metodi *sendWantedFile* čiji je prvi dio identičan metodi *sendNamesOfFiles* (Ispis 4-8), što znači da se ovisno o tipu datoteke kreira niz i u spremaju sve datoteke željenog tipa. Potom se iz tog niza pronalazi datoteka sa primljenim imenom i na kraju se ta datoteka prosljeđuje klasi *ImageSender* kako bi se poslala natrag klijentu, a kôd metode koja izvršava slanje prema klijentu, prikazan je u ispisu Ispis 4-13.

```
protected Void doInBackground(File... voids) {
     File file = voids[0];
     byte[] byteArray = new byte[(int) file.length()];

     try {
          s = new Socket(IP, 7800);
          FileInputStream fis = new FileInputStream(file);
          BufferedInputStream bis = new BufferedInputStream(fis);

          DataInputStream dis = new DataInputStream(bis);
          dis.readFully(byteArray, 0, byteArray.length);

          OutputStream os = s.getOutputStream();

          DataOutputStream dos = new DataOutputStream(os);
          dos.writeUTF(file.getName());
          dos.writeLong(byteArray.length);
          dos.write(byteArray, 0, byteArray.length);
          dos.flush();

          s.close();
        } catch (IOException e) {}
     return null;
}
```

Ispis 4-13: Definicija metode koja šalje datoteku klijentu

Slično kao i kod slanja datoteke sa klijenta prema poslužitelju. Najprije se vrši spajanje na utičnicu preko IP adrese klijenta i ulaz na kojem klijent čeka odgovor. Otvara se tok podataka u koji se onda zapisuje ime, veličinu i u konačnici samu datoteku, koja je prethodno konvertirana u niz bajtova.

4.11. Primanje datoteke

Ako unutar poruke sa klijenta stigne naredba *newFile,* to je znak klijentu da iza te poruke može očekivati datoteku, a ne tekstualnu poruku kao i obično, stoga je potrebno otvoriti tokove podataka na nešto drugačiji način. Kôd za primanje i spremanje datoteku na poslužitelj prikazan je u ispisu Ispis 4-14.

```
DataInputStream dataInputStream = new DataInputStream(
                                                s.getInputStream());
int fileNameLength = dataInputStream.readInt();

if(fileNameLength > 0) {
     byte[] fileNameBytes = new byte[fileNameLength];
     dataInputStream.readFully(fileNameBytes, 0, fileNameBytes.length);
     String fileName = new String(fileNameBytes);
     int fileContentLength = dataInputStream.readInt();

     if (fileContentLength > 0) {
          byte[] fileContentBytes = new byte[fileContentLength];
          dataInputStream.readFully(
                         fileContentBytes, 0, fileContentBytes.length);
          File root = Environment.getExternalStorageDirectory();
          File dir = new File(root + File.separator +"server_app");
          if (!dir.exists()) dir.mkdir();

               //Create file..
               File file = new File(dir + File.separator + fileName);
               file.createNewFile();

               FileOutputStream out = new FileOutputStream(file);
               out.write(fileContentBytes);
               out.close();
               dataInputStream.close();
               continue;
          }
     }
```

Ispis 4-14: Dio kôda zaslužan za primanje i spremanje datoteke na poslužitelj

Datoteka sa klijenta na poslužitelja stiže u nizu bajtova, a čitaju se isti podaci koji su i upisani sa poslužiteljske strane: ime datoteke, njenu veličinu, te datoteku kao niz bajtova. Jednom kad je datoteka pročitana kao niz bajtova, kreira se sama datoteka iz tog niza i sprema u mapu server_*app*.

5. Zaključak

Kroz ovaj rad su korištene brojne funkcionalnosti koje se uvijek mogu iskoristiti u nekim drugim projektima, poput pretraživanja datoteka, slanja i primanja datoteka između dva uređaja, klijent-poslužitelj arhitektura.

Pored svih mogućnosti koje su opisane i realizirane u ovom radu: dohvaćanje i prikazivanje kontakta, slanje SMS poruka, preuzimanje i slanje datoteka na mobilni uređaj, upravljanjem glasnoćom, postavljanje slike zaslona te dodavanje događaja u kalendar, moguće je implementirati i druge funkcionalnosti kao nadogradnje na postojeće aplikacije. Jedna od nadogradnji može biti dodavanja i uređivanje kontakata, druga nadogradnja može biti ispis zadnjih poziva, treća pak može biti ispis zadnjih SMS poruka i brojne druge.

Nažalost neke planirane funkcionalnosti nije bilo moguće realizirati zbog Androidovih ograničenja. Primjer takve funkcionalnosti je postavljanje proizvoljne audio datoteke za zvuk zvona dolaznog poziva. Za takvu funkcionalnost potrebno je imati određena prava koja imaju samo službene android aplikacije. Stoga iako postoje brojne mogućnosti za nadogradnju ovih aplikacije, treba imati na umu da ipak postoje i neka ograničenja u mogućnosti izvedbe određenih funkcionalnosti.

Literatura

[1] Java, osnovni podaci, https://en.wikipedia.org/wiki/Java_(programming_language)
datum zadnjeg pristupa: 19.08.2021.

[2] Android studio,osnovni podaci, https://en.wikipedia.org/wiki/Android_Studio
datum zadnjeg pristupa: 19.08.2021.

[3] Baeldung, Client-Server Architecture, https://www.baeldung.com/a-guide-to-java-sockets
datum zadnjeg pristupa: 19.08.2021.

[4] Stackoverflow, https://stackoverflow.com/
datum zadnjeg pristupa: 18.08.2021.

[5] Android Developer Documentation, https://developer.android.com/guide
datum zadnjeg pristupa: 19.08.2021.

[6] Flaticon, grafički elementi, https://www.flaticon.com/
datum zadnjeg pristupa: 18.08.2021.

Popis slika

Slika 2-1: Shematski prikaz načina komunikacije android aplikacije i računalne aplikacije 7

Slika 3-1: Prikaz početnog zaslona prije uspostavljene konekcije 9

Slika 3-2: Prikaz početnog zaslona nakon uspostavljene konekcije 10

Slika 3-3: Prikaz ispisa kontakata 15

Slika 3-4: Prikaz dohvaćenih imena slika 18

Slika 3-5: Prikaz opcije preuzimanja slike 19

Slika 3-6: Prikaz mogućnosti postavljanja slike zaslona 21

Slika 3-7: Prikaz zaslona za slanje datoteke na android uređaj 22

Slika 3-8: Prikaz skočnog prozora za odabir datoteke za slanje 22

Slika 3-9: Prikaz zaslona za slanje SMS poruke 25

Slika 3-10: Prikaz zaslona za dodavanje novog događaja u kalendar 27

Slika 3-11: Prikaz zaslona za kontrolu glasnoće 29

Slika 4-1: Prikaz početnog zaslona android aplikacije 31

Slika 4-2: Prikaz zaslona za prihvaćanje ili odbijanje konekcije 35

Popis ispisa

Ispis 2-1: Dohvaćanje unesene IP adrese, formiranje poruke i slanje iste poslužitelju 11

Ispis 2-2: Definicija metode sendStringMessageToAndroidDevice 11

Ispis 2-3: Dio metode za konekciju prema poslužitelju, zadužen za slušanje odgovora od poslužitelja 12

Ispis 2-4: Definicija metode contactsBtnActionPerformed 13

Ispis 2-5: Definicija metode waitContactsFromAndroidDevice 14

Ispis 2-6: Primjer zapisa kontakta u poruci sa poslužitelja prema klijentu 14

Ispis 2-7: Definicija metode contactsRefreshBtnActionPerformed 15

Ispis 2-8: Definicija metode picturesBtnActionPerformed 16

Ispis 2-9: Definicija metode waitFilesFromAndroidDevice 17

Ispis 2-10: Primjer formata poruke u kojem stižu imena datoteka 18

Ispis 2-11: Definicija metode koja se pokreće klikom na Download 19

Ispis 2-12: Definicija metode downloadFile [4] 20

Ispis 2-13: Definicija metode setAsWallpaperOptionActionPerformed 21

Ispis 2-14: Definicija metode chooseFileBtnMouseReleased 23

Ispis 2-15: Definicija metode sendFileBtnMouseReleased 24

Ispis 2-16: Definicija metode sendSMSBtnMouseReleased 26

Ispis 2-17: Definicija metode calendarAddEventBtnMouseReleased 29

Ispis 2-18: Definicija metode music0BtnMouseReleased 30

Ispis 3-1: Dozvole defnirane u AndroidManifest.xml 32

Ispis 3-2: Dio kôda zadužen za traženje dozvoli od korisnika 32

Ispis 3-3: Definicija metode koja odlučuje koju metodu pozvat ovisno o klijentskoj naredbi 34

Ispis 3-4: Definicija metode connectRequest 35

Ispis 3-5: Kôd koji ima zadaću poslati tekstualnu poruku klijentu 36

Ispis 3-6: Definicija metode sendContacts 37

Ispis 3-7: Defincija metode sendSMS 38

Ispis 3-8: Definicija metode sendNamesOfFiles 39

Ispis 3-9: Metoda koja dohvaća sve datoteke tipa slika 40

Ispis 3-10: Definicija metode addEventInCalendar 41

Ispis 3-11: Definicija metode setWallpaper 42

Ispis 3-12: Definicija metode setVolumeForType 43

Ispis 3-13: Definicija metode koja šalje datoteku klijentu 44

Ispis 3-14: Dio kôda zaslužan za primanje i spremanje datoteke na poslužitelj 45