Санкт-Петербургское Государственное
бюджетное профессиональное образовательное учреждение
«Колледж информационных технологий»

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ

«Разработка приложения с использованием протокола UDP для передачи данных»

МДК 01.03. РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Специальность 09.02.07 Информационные системы и программирование

Выполнил студент
гр. 393: Баланин Э.Н.

Санкт-Петербург 2022

Проектирование интерфейса приложения

Ссылка на репозиторий с программной –

https://github.com/bl1st/DataExchange/tree/master

Интерфейс приложения был спроектирован в приложении DrawIO. Было разработано 2 макета: макет главной страницы и макет страницы настроек. Макеты продемонстрированы на рисунке 1.

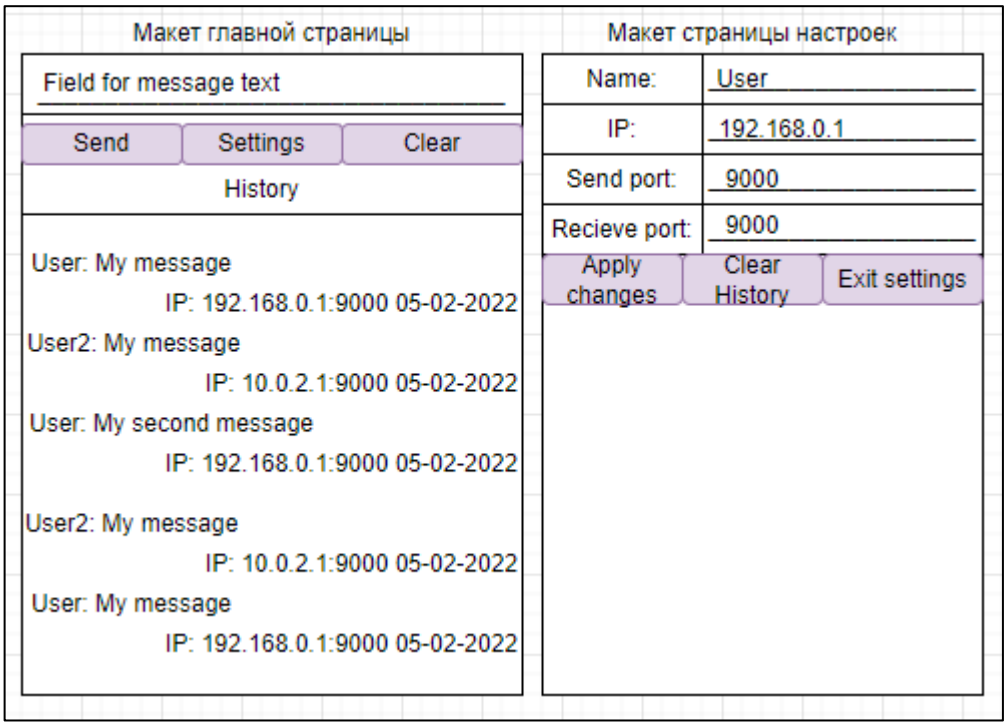


Рисунок 1 – Макеты приложения

Разработка базы данных

Также была разработана база данных на языке SQLite3 для хранения сообщений. ER-диаграмма базы данных представлена на рисунке 2.

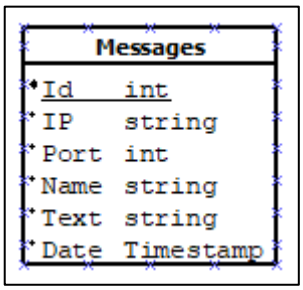


Рисунок 2 – ER-диаграмма базы данных хранения сообщений

В разработанной таблице Messages (рисунок 2) хранятся все отправленные и полученные на устройство сообщения.

Интерфейс приложения

Был разработан интерфейс приложения в соответствии разработанными макетами. На рисунке 3 изображены интерфейсы двух окон приложения.

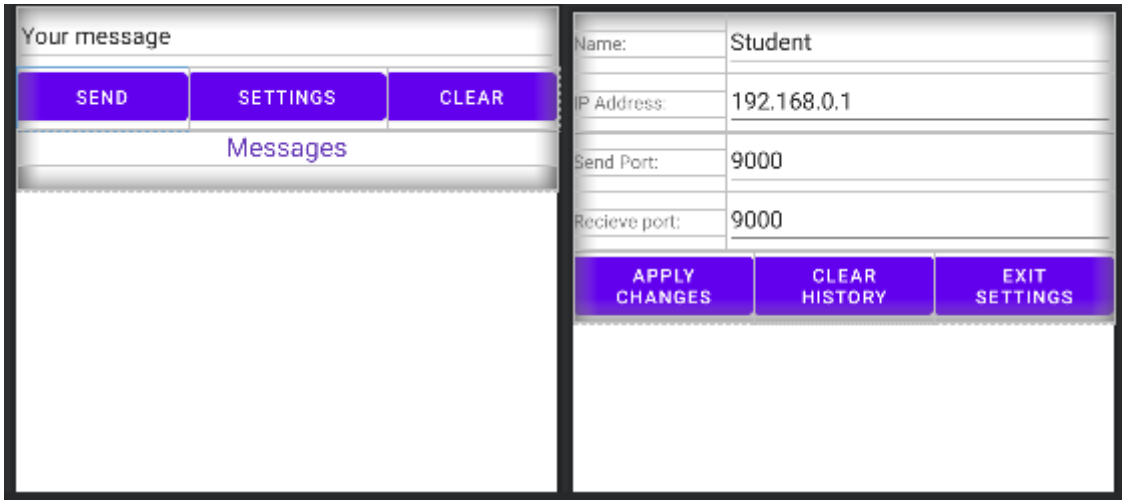


Рисунок 3 – Интерфейсы приложения

На рисунке 3 в левой части продемонстрирован интерфейс главного начального интерфейса приложения. По нажатию на кнопку «Settings» открывается интерфейс продемонстрированный в правой части рисунка 3.

Программный код приложения

Для правильного функционирования приложения был разработан следующий ниже программный код.

Вначале была разработан программный код базы данных в файле DB.java. Программный код продемонстрирован на рисунке 4.

```
package com.example.dataexchange;

import ...

public class DB extends SQLiteOpenHelper {
    public DB(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE messages (IP TEXT,PORT TEXT,NAME TEXT,MESSAGE TEXT, DATE INTEGER);";
        db.execSQL(sql);
    }
    public void SaveMessage(Message m) {
        SQLiteDatabase db = getWritableDatabase();
        String sql = "INSERT INTO messages VALUES ('" + m.IP + "','" + m.Port + "','" + m.Name + "','" + m.Text + "'," + m.DateTime.getTime() + ");";
        db.execSQL(sql);
    }
   public void LoadHistory(ArrayList<Message> lst){
       SQLiteDatabase db = getReadableDatabase();
       String sql = "SELECT * FROM messages";
       Cursor cur = db.rawQuery(sql,  selectionArgs: null);
       if (cur.moveToFirst() == true) {
           do {
               Message m = new Message();
               m.IP = cur.getString( columnIndex: 0);
               m.Port = cur.getString( columnIndex: 1);
               m.Name = cur.getString( columnIndex: 2);
               m.Text = cur.getString( columnIndex: 3);
               m.DateTime = new Date(Long.parseLong(cur.getString( columnIndex: 4)));
               lst.add(m);

           }while (cur.moveToNext() == true);
       }
   }
    public void CLEARALLTABLE(){
        SQLiteDatabase db = getWritableDatabase();
        String sql = "DELETE FROM messages";
        db.execSQL(sql);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
```

Рисунок 4 – Программный код класса DB.java

Также был разработан класс g программный код которого продемонстрирован на рисунке 5. При помощи данного класса происходит инициализация соединения с базой данных.

```
package com.example.dataexchange;
public final class g { //493 balanin
    static DB messages;
}
```

Рисунок 5 – Программный код класса g

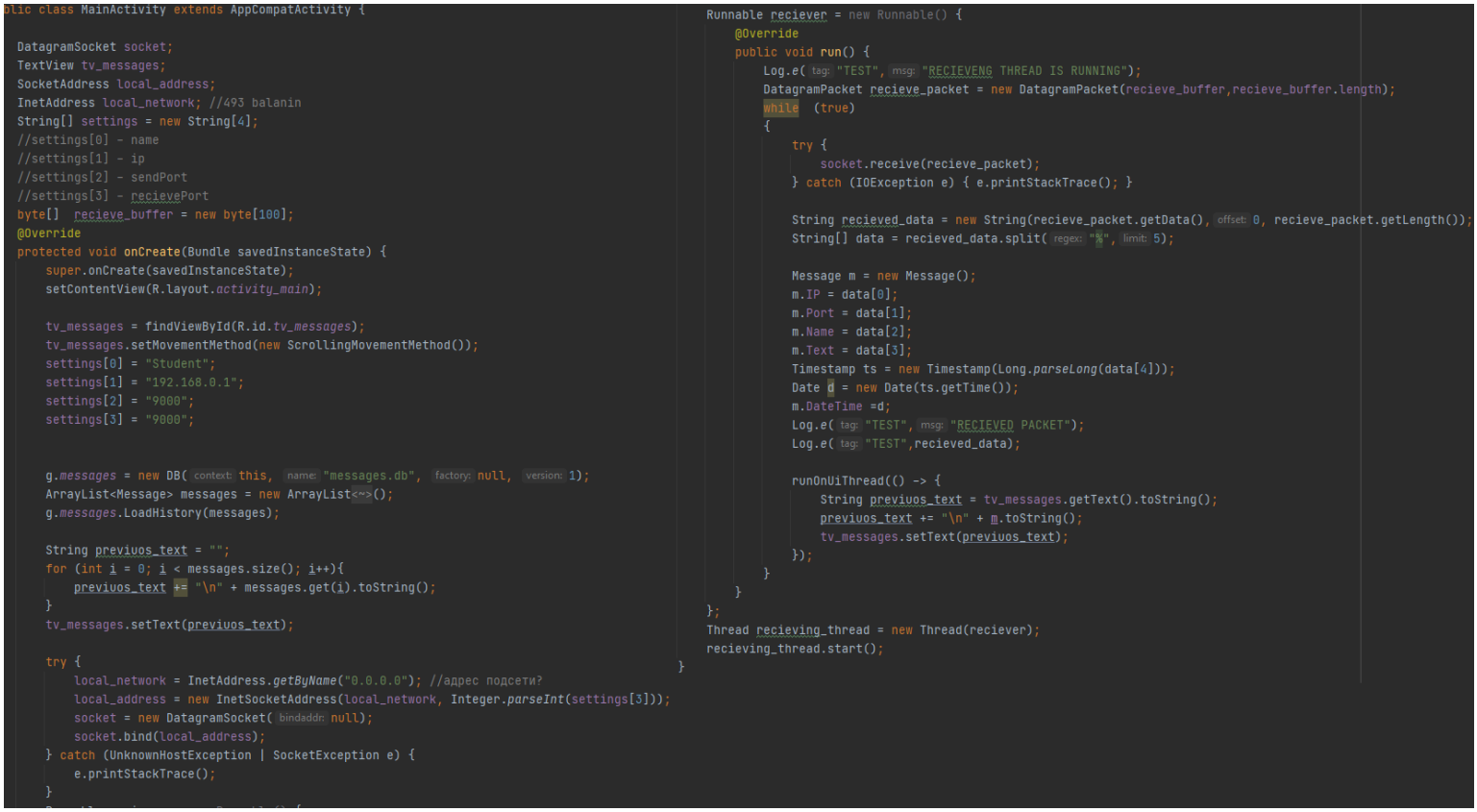
Для упрощенной работы с структурой данных сообщений в данном приложении был разработан класс Message который хранит в себе всю информацию определенного сообщения. На рисунке 6 продемонстрирован программный код данного класса. В данном классе определены 2 метода: метод BufferMessage (конвертирует все переменные класса в одну строку, разделяя каждую переменную процентом, а затем переводит данную строку в массив типа byte) и метод toString.

```
package com.example.dataexchange;
import ...
public class Message {
    public String IP;  //493 balanin
    public String Port;
    public String Name;
    public String Text;
    public Date DateTime;

    public byte[] BufferMessage(){
        int timeStamp = (int) (this.DateTime.getTime() / 1000);
        byte[] buffer = (this.IP + "%" + this.Port + "%" + this.Name + "%" +this.Text + "%" + timeStamp).getBytes(StandardCharsets.UTF_8);
        return buffer;
    }
    public String toString() {
        return this.Name + ": " + this.Text + "\n" + "IP: "+ this.IP + ":"+this.Port  +"   Date: " + this.DateTime.toString();
    }
}
```

Рисунок 6 – Код класса Message

Программный код главной Activity продемонстрирован на рисунке 7. На данном рисунке продемонстрирован лишь программный код метода OnCreate().



```
blic class MainActivity extends AppCompatActivity {

    DatagramSocket socket;
    TextView tv_messages;
    SocketAddress local_address;
    InetAddress local_network; //493 balanin
    String[] settings = new String[4];
    //settings[0] - name
    //settings[1] - ip
    //settings[2] - sendPort
    //settings[3] - recievePort
    byte[]  recieve_buffer = new byte[100];
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tv_messages = findViewById(R.id.tv_messages);
        tv_messages.setMovementMethod(new ScrollingMovementMethod());
        settings[0] = "Student";
        settings[1] = "192.168.0.1";
        settings[2] = "9000";
        settings[3] = "9000";


        g.messages = new DB( context: this,  name: "messages.db",  factory: null,  version: 1);
        ArrayList<Message> messages = new ArrayList<~>();
        g.messages.LoadHistory(messages);

        String previuos_text = "";
        for (int i = 0; i < messages.size(); i++){
            previuos_text += "\n" + messages.get(i).toString();
        }
        tv_messages.setText(previuos_text);

        try {
            local_network = InetAddress.getByName("0.0.0.0"); //адрес подсети?
            local_address = new InetSocketAddress(local_network, Integer.parseInt(settings[3]));
            socket = new DatagramSocket( bindaddr: null);
            socket.bind(local_address);
        } catch (UnknownHostException | SocketException e) {
            e.printStackTrace();
        }
    Runnable reciever = new Runnable() {
        @Override
        public void run() {
            Log.e( tag: "TEST", msg: "RECIEVENG THREAD IS RUNNING");
            DatagramPacket recieve_packet = new DatagramPacket(recieve_buffer,recieve_buffer.length);
            while  (true)
            {
                try {
                    socket.receive(recieve_packet);
                } catch (IOException e) { e.printStackTrace(); }

                String recieved_data = new String(recieve_packet.getData(), offset: 0, recieve_packet.getLength());
                String[] data = recieved_data.split( regex: "%", limit: 5);

                Message m = new Message();
                m.IP = data[0];
                m.Port = data[1];
                m.Name = data[2];
                m.Text = data[3];
                Timestamp ts = new Timestamp(Long.parseLong(data[4]));
                Date d = new Date(ts.getTime());
                m.DateTime =d;
                Log.e( tag: "TEST", msg: "RECIEVED PACKET");
                Log.e( tag: "TEST",recieved_data);

                runOnUiThread(() -> {
                    String previuos_text = tv_messages.getText().toString();
                    previuos_text += "\n" + m.toString();
                    tv_messages.setText(previuos_text);
                    });
                }
            }
        };
        Thread recieving_thread = new Thread(reciever);
        recieving_thread.start();
    }
```

Рисунок 7 – метод OnCreate() главной Activity

Данный метод привязывает элементы интерфейса к переменным соответствующих классов. Далее происходит инициализация полей массива settings (данный массив содержит в себе настройки для передачи данных: имя пользователя, Ip-адрес, порт отправителя и получателя). В данном методе запущен новый поток в котором функционирует бесконечный цикл ожидающий получения сообщения на проинициализированный сокет. При получении сообщения все данные оборачиваются в объект класса Message после чего это сообщение переносится на интерфейс главной Activity и записывается в базу данных.

На рисунке 8 изображен программный код отправки сообщения из главной Activity. При нажатии на кнопку «Отправить» создается новый объект класса Message который наполняется данными в соответствии с данными массива settings и поля с текстом сообщения. Далее происходит инициализация пакета отправки и сокета, при помощи которого и происходит оправка пакета.

```
DatagramPacket send_packet;
public void onButtonSendMessage_Click(View v){
    EditText et_msg = findViewById(R.id.et_message);
    String msg = et_msg.getText().toString();

    Message m = new Message();
    m.IP = settings[1];
    m.Port = settings[2];
    m.Name = settings[0];
    m.Text = msg;
    m.DateTime = new Date();

    byte[] send_buffer = m.BufferMessage();

    try {
        InetAddress remote_address = InetAddress.getByName(settings[1]);
        send_packet = new DatagramPacket(send_buffer,send_buffer.length,remote_address,Integer.parseInt(settings[2]));
    } catch ( UnknownHostException e) { e.printStackTrace(); }

    Runnable r = new Runnable() {
        @Override
        public void run() {
            Log.e( tag: "TEST", msg: "SENDING THREAD IS RUNNING");
            try {
                socket.send(send_packet);
                Log.e( tag: "TEST", msg: "PACKET SENT");
                g.messages.SaveMessage(m);
                String previuos_text = tv_messages.getText().toString();
                previuos_text += "\n" + m.toString();
                String finalPreviuos_text = previuos_text;

                runOnUiThread(() ->{
                    tv_messages.setText(finalPreviuos_text);
                });
            } catch (IOException e) { e.printStackTrace(); }
        }
    };
    Thread send_thread = new Thread(r);
    send_thread.start();
}
```

Рисунок 8 – Программный код нажатия на кнопку «Отправить»

На рисунке 9 изображен программный код нажатия на кнопку «Настройки» и получения данных из второй Activity (Activity с настройками).

Метод OnButtonSettings_Click инициализирует объект класса Intent для запуска и передачи данных во вторую Activity с настройками. Метод OnActivityResult получает данные после завершения работы второй Activity. Метод OnButtonClearChat_Click очищает TextView со всеми сообщениями (не удаляя записи из БД).

```
public void onButtonSettings_Click(View v) {
    Intent i = new Intent( packageContext: this, SettingsActivity.class);
    i.putExtra( name: "ip", settings[1]);
    i.putExtra( name: "sendPort", settings[2]);
    i.putExtra( name: "recievePort",settings[3]);
    i.putExtra( name: "name",settings[0]);
    startActivityForResult(i, requestCode: 555); }

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    Log.e( tag: "TEST",  msg: " Method onActivity result initializing");
    if (requestCode == 555) {
        if (data !=null) {
            settings[0]  = data.getStringExtra( name: "name");
            settings[1]  = data.getStringExtra( name: "ip");
            settings[2]  = data.getStringExtra( name: "sendPort");
            settings[3]  = data.getStringExtra( name: "recievePort");
            ChangeData();
            Log.e( tag: "TEST", msg: "RECIEVED DATA FROM SETTINGS ACTIVITY");
            Log.e( tag: "TEST", msg: settings[0] + settings[1] + settings[2] + settings[3]);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
private void ChangeData() {
    local_address = new InetSocketAddress(local_network, Integer.parseInt(settings[3]));
}
public void onButtonClearChat_Click(View v){
    tv_messages.setText(null);
}
```

Рисунок 9 – Оставшийся программный код главной Activity

На рисунке 10 изображен программный код Activity с настройками соединения. В методе OnCreate принимаются данные из первой Acitivty и ими заполняются поля данной Activity. Функция onButtonApply_Click() инициализирует новый Intent, наполняет его данными из полей Activity и отправляет эти данные в первую Activity. OnButtonExit_Click закрывает Activity без сохранения изменений настроек.

```
public class SettingsActivity extends AppCompatActivity {
    //493 balanin
    EditText ip,send_port,recieve_port,name;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        ip = findViewById(R.id.et_ip);
        send_port = findViewById(R.id.et_sendPort);
        recieve_port = findViewById(R.id.et_recievePort);
        name = findViewById(R.id.et_name);

        Intent i = getIntent();
        name.setText(i.getStringExtra( name: "name"));
        ip.setText(i.getStringExtra( name: "ip"));
        send_port.setText(i.getStringExtra( name: "sendPort"));
        recieve_port.setText(i.getStringExtra( name: "recievePort"));

    }

    public void onButtonApply_Click(View v){

        Intent i = new Intent();
        i.putExtra( name: "ip", ip.getText().toString());
        i.putExtra( name: "sendPort", send_port.getText().toString());
        i.putExtra( name: "recievePort",recieve_port.getText().toString());
        i.putExtra( name: "name",name.getText().toString());
        Log.e( tag: "TEST", msg: "FINISHING SETTINGS ACTIVITY");
        setResult(RESULT_OK,i);
        finish();
    }

    public void onButtonClearHistory_Click(View v){

       g.messages.CLEARALLTABLE();
        Toast.makeText( context: this,  text: "Successfully cleared history", Toast.LENGTH_SHORT).show();
    }

    public void onButtonExit_Click(View v){

        setResult(RESULT_CANCELED);
        finish();
    }
}
```

Рисунок 10 – Программный код второй Activity

Проверка работоспособности приложения

На рисунке 11 изображена работа приложения.

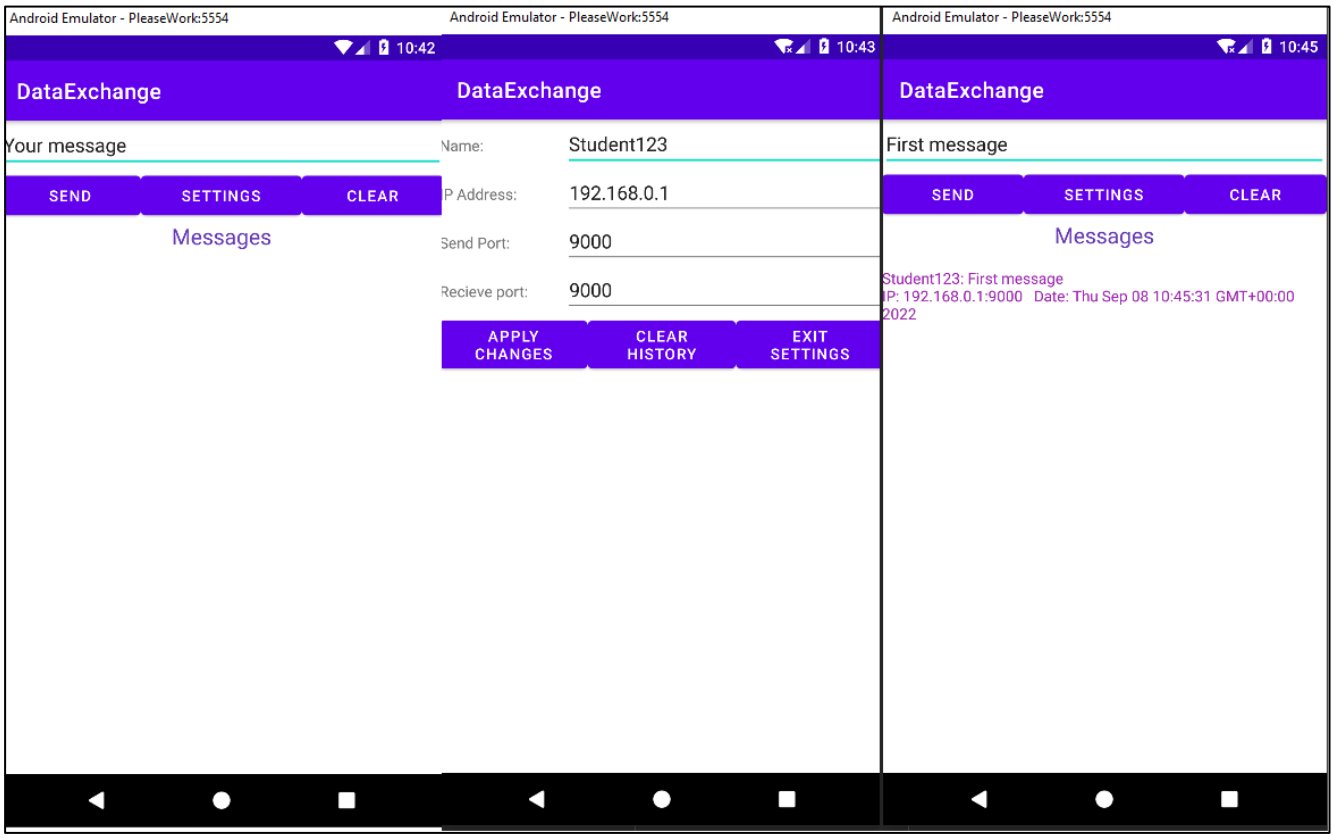


Рисунок 11 – Работа приложения