UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

Facultatea de Informatică

Lucrare de Licență

Smart Remote

Propusă de

Mitocariu Ioan-Emilian

Sesiunea: Februarie, 2018

Coordonator Ştiinţific

Lector, dr. Anca Ignat

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

Facultatea de Informatică

Smart Remote

Mitocariu Ioan-Emilian

Sesiunea: Februarie 2018

Coordonator Ştiinţific

Lector, dr. Anca Ignat

DECLARAȚIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul "Smart Remote" este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate cu respectarea drepturilor de autor şi dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iaşi,

Absolvent Mitocariu Ioan-Emilian

_____

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „Instant – Web Messenger”, codul sursă al programelor şi celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iaşi,

Absolvent Mitocariu Ioan-Emilian

_____

Acord privind proprietatea dreptului de autor

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, format executabil şi sursă, să aparţină autorului prezentei lucrări, Mitocariu Ioan-Emilian

Încheierea acestui acord este necesară din următoarele motive:

- în eventualitatea continuării proiectului pentru distribuire

Iaşi,

DECAN Adrian Iftene

Mitocariu Ioan-Emilian

_____

Cuprins

Introducere

1. Motivatie

Multe lucruri din ziua de astazi, cum ar fi becuri, aerul conditionat, prize si multe altele, pot fi controlate prin telecomenzi care folosesc frecvente radio pentru a transmite diferite comenzi. Prin simpla apasare a unui buton poti inchide un bec sau porni aerul conditionat fara a fi nevoit sa te misti atat timp cat ai telecomanda la indemana.

Dar in momentul in care ai mai multe astfel de dispozitive in casa, care pot fi controlate de la distanta, vei avea de asemenea si mai multe telecomenzi. Acest lucru duce la incomoditatea de a trebui sa caram mai multe telecomenzi.

Deoarece traim intr-o lume in care aproape toti oamenii au un smartphone pe care il poarta peste tot cu ei, ar fi mult mai usor si comod sa putem controla anumite dispozitive din jurul nostru de pe telefonul personal.

Am ales aceasta tema fiindca doresc sa aduc o solutie prin care sa putem inlocui telecomenzile fizice cu telecomenzi virtuale ce sunt mult mai la indemana.

2. Context

Una dintre primele utilizari a undelor radio pentru controlul de la distanta a fost facuta de catre Nikola Tesla in 1898[1], care a controlat o barca in miniatura. Aceasta avea o antena prin care recepta comenzi, iar ele erau trimise printr-o cutie ce permitea controlarea vitezei si a directiei.

Dupa anii 1930-1940 telecomenzile pe baza de frecvente radio au inceput usor sa fie folosite pentru a controla dispozitive din jurul casei, unul dintre exemple fiind usa de la garaj.

---

[1] https://science.howstuffworks.com/innovation/repurposed-inventions/history-of-remote-control.htm

Telecomenzile pe baza de unde radio inca sunt folosite si astazi, ele devenind mai mici si mai accesibile oamenilor. Din dorinta de a controla diferite dispozitive de la distanta prin telefon, au inceput sa fie folosite tehnologii asemanatoare, cum ar fi Bluetooth si Wi-Fi care sunt suportate de catre telefoane in mod direct.

In acest context apare problema in care vrem sa controlam un dispozitiv ce primeste comenzi prin frecvente radio, dar un smartphone obisnuit nu are posibilitatea de a trimite astfel de comenzi.

3. Descrierea sumara a solutiei

Din moment ce un smartphone obisnuit nu poate trimite comenzi prin frecvente radio, este nevoie de un obiect intermediar ce poate comunica atat cu dispozitive ce folosesc frecvente radio cat si cu un telefon.

Solutia oferita de mine necesita folosirea unui **Raspberry Pi** la care se conecteaza:

- un receptor radio de frecventa 433MHz pentru copierea comenzilor de pe telecomanda
- un transmitator radio de frecventa 433MHz pentru trimiterea comenzilor catre dispozitive
- un adaptor USB pentru Wi-Fi

Controlarea pinilor **GPIO** de pe Raspberry Pi se face cu ajutorul pachetului **WiringPi**. La acesti pini se conecteaza receptorul si transmitatorul radio care la randul lor sunt controlati printr-un proiect open source **433Utils**[2] de la **NinjaBlock**.

Pentru ca Raspberry Pi sa poata comunica cu telefonul, ei se conecteaza prin Wi-Fi la un router local. Pe Raspberry Pi am folosit **Python** impreuna cu microframework-ul **Flask** pentru a crea un server ce raspunde la request-uri **HTTP**.

In final am dezvoltat o aplicatie de telefon pentru sistemul de operare Android. In acest caz am ales **Java**, fiind limbajul oficial si cel mai folosit pentru dezvoltare de aplicatii Android. Am ales sa lucrez in **Android Studio** pentru faptul ca suporta Java si este de asemenea IDE-ul oficial pentru Android. Pentru trimiterea de requesturi catre serverul de pe Raspberry Pi am folosit libraria **Volley**.

---

[2] https://github.com/ninjablocks/433Utils

Ca sistem de administrare a bazelor de date relationale am folosit SQLite, principalul sistem nativ oferit de Android. Este folosit pentru stocarea de informatii, cum ar fi lista de telecomenzi, butoane, codurile frecventelor radio atribuite fiecarui buton si comenzile programate pentru a fi trimise periodic.

4. Structura lucrarii

Lucrarea de licenta va avea trei capitole in care se va descrie in detaliu procesul de dezvoltare al aplicatiei si cum functioneaza:

- Configurarea Raspberry Pi-ului
- Descrierea aplicatiei Android

Primul capitol descrie pasii necesari pentru configurarea Raspberry Pi-ului. De la instalarea sistemului de operare si controlarea receptorului/transmitatorului radio pana la crearea server-ului in Python ce raspunde la comenzi.

Cel de-al doilea capitol va descrie tehnologiile folosite si va explica cum functioneaza aplicatia SmartRemote, folosind capturi de ecran cu blocuri de cod si pagini din aplicatie.

1. Contributii

Contributiile mele pentru aceasta lucrare incep de la cautarea de informatii despre cum functioneaza Raspberry Pi-ul, cum se configureaza unul si cum pot controla transmitatorul si emitatorul conectati prin pinii GPIO. Aceste informatii vor fi disponibile la sfasitul lucrarii in cadrul Bibliografiei.

Dupa ce informatiile necesare au fost gasite si acumulate, am inceput sa le pun in practica prin punerea in functiune a Raspberry Pi-ului. Dupa ce am reusit sa controlez receptorul si transmitatorul radio cu ajutorul unor pachete open source am creat un server ce raspunde la request-urile facute de catre un client, in cazul meu aplicatia Android.

Ultimul pas din contributiile mele aduse acestui proiect a fost crearea aplicatiei android ce va fi descrisa in detaliu in capitolele urmatoare.

Sumar al contributiilor mele in cadrul acestei lucrari:

- configurarea Raspberry Pi-ului
- implementarea server-ului de pe Raspberry Pi
- dezvoltarea aplicatiei Android

2. Configurarea Raspberry Pi-ului

2.1 Raspberry Pi

Raspberry Pi[3] este un “Single-Board Computer” de dimensiunea unui card de credit. El este fabricat in United Kingdom si a fost proiectat pentru invatarea notiunilor de baza ale informaticii. A devenit mult mai popular decat a fost anticipat si este adesea folosit in proiecte de robotica si home automation (ca si aceasta lucrare).

In acest proiect voi folosi un Raspberry Pi Model A. Pentru a fi pus in functiune el are nevoie de:

- un cablu de alimentare
- un card SD pe care se va instala sistemul de operare
- un adaptor USB pentru Wi-Fi

Principalul sistem de operare folosit pentru Raspberry Pi este **Raspbian** care este o distributie de linux bazat pe Debian. Pentru a instala sistemul de operare am formatat cardul SD, am descarcat imaginea pentru Raspbian si folosind un cititor de carduri SD impreuna cu Etcher.io am “ars” imaginea pe card. Acum Raspberry Pi-ul este gata pentru a fi pornit. Pentru ca IP-ul sa fie static si sa se conecteze la router am modificat fisierul **/etc/network/interfaces**:

```
auto wlan0
iface wlan0 inet static
        address 192.168.0.140
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        wpa-ap-scan 1
        wpa-scan-ssid 1
        wpa-ssid MyWiFiName
        wpa-psk MyWiFiPassword
```

Figura 1: Configurarea unui IP static

[3] https://en.wikipedia.org/wiki/Raspberry_Pi

Unul dintre atuurile unui Raspberry Pi sunt pinii **GPIO** care pot fi folositi pentru a connecta senzori, transmitatori, receptori, lumini si alte dispozitive. Pe langa pinii GPIO se mai gasesc pini de alimentare cu 3.3V respectiv 5V si pini de impamantare. Pentru modelul folosit de mine ei sunt aranjati astfel:

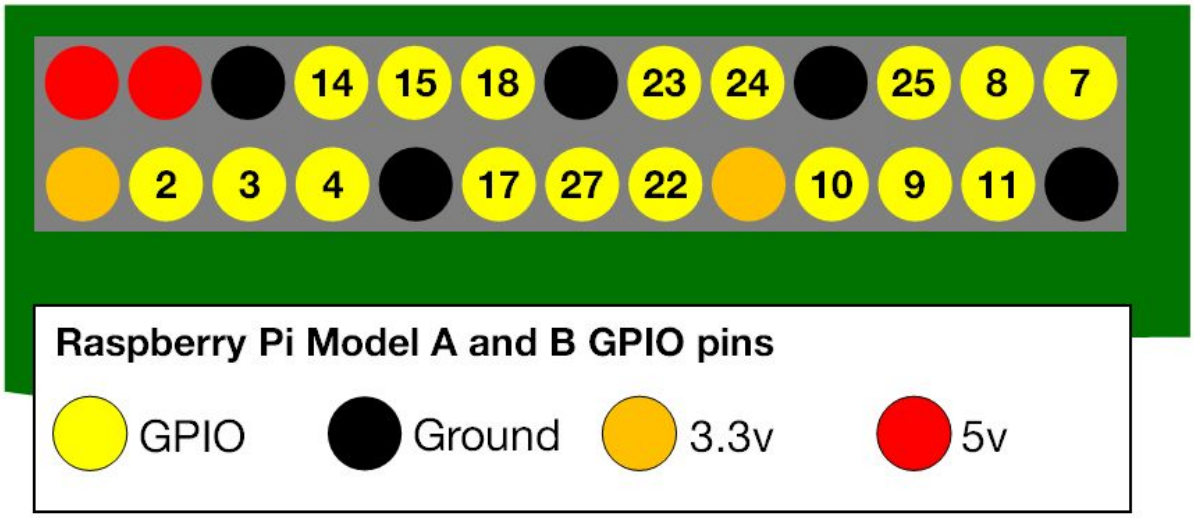

Figura 2: Raspberry Pi model A pins

Acum e timpul sa conectam receptorul si transmitatorul de frecventa radio 433MHz la Raspberry Pi. Amandoi vor avea nevoie sa se conecteze la 3 pini:

- un pin de alimentare cu 5V
- un pin de impamantare
- unul din pinii GPIO

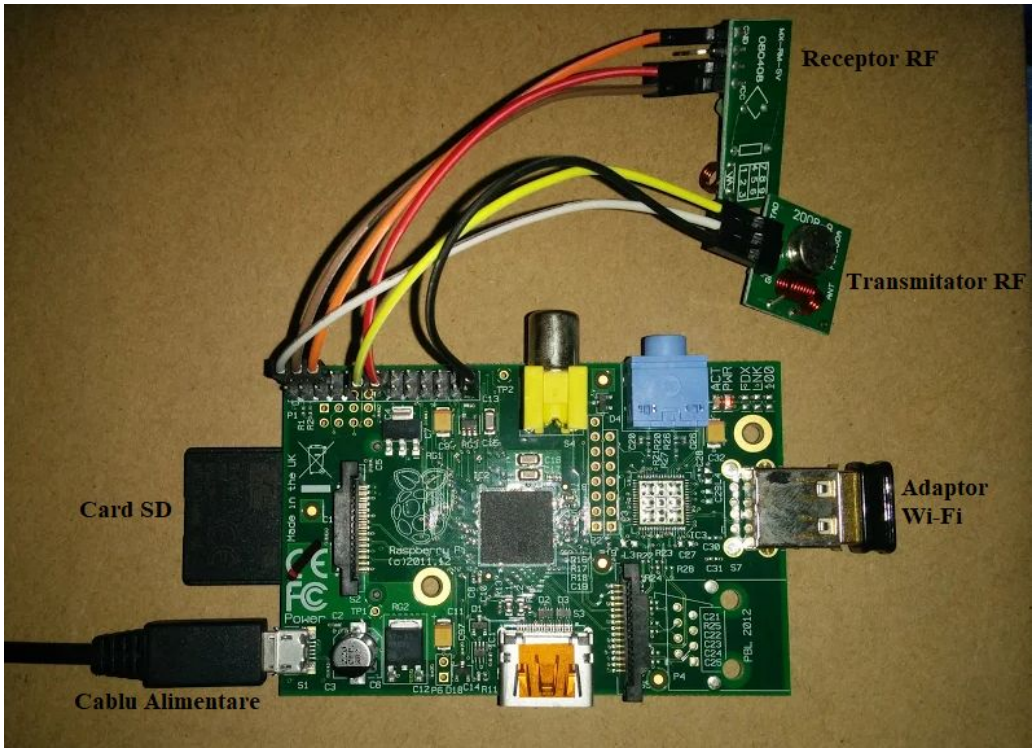

Figura 3: Raspberry Pi cu toate componentele connectate

Pentru a putea controla pinii GPIO am folosit libraria **WiringPi** care este scrisa in limbajul C. Pentru a controla receptorul si transmitatorul am folosit un proiect open source numit 433Utils de la NinjaBlocks care imi permite sa ascult sau sa trimit semnale radio.

2.2 HTTP Server

Am ales sa implementez serverul de pe Raspberry Pi care va raspunde la request-uri HTTP in **Python** folosind micro framework-ul **Flask**[4]. Acest framework este bazat pe Werkzeug si Jinja2. Cateva aplicatii mai cunoscute care folosesc Flask sunt Pinterest si LinkedIn. Am ales acest framework deoarece ofera tot ce am nevoie pentru serverul meu, este simplu de implementat, iar python este un limbaj destul de popular cu o comunitate mare.

Server-ul are 4 tipuri de request-uri implementate:

- **getcode** → porneste receptorul radio, copie primul semnal primit si il returneaza sub forma unui cod de 6 cifre

[4] https://en.wikipedia.org/wiki/Flask_(web_framework)

```
@app.route('/getcode', methods=['GET'])
def getcode():
    code = subprocess.check_output('sudo /home/emilian/433Utils/RPi_utils/RFSniffer', shell=True)
    return code
```

Figura 4: Request pentru a inregistra un semnal

- **sendcode** → trimite un code ce formeaza un semnal radio cu ajutorul transmitatorului radio

```
@app.route('/sendcode', methods=['POST'])
def sendcode():
    code = request.args.get('code')
    os.system('sudo /home/emilian/433Utils/RPi_utils/codesend ' + code)
    return ''
```

Figura 5: Request pentru a trimite un semnal

- **setautocode** → programeaza cu crontab ca un semnal radio sa fie trimis la o anumita ora

```
@app.route('/setautocode', methods=['POST'])
def setautocode():
    code = request.args.get('code')
    minute = request.args.get('minute')
    hour = request.args.get('hour')
    os.system("sudo crontab -u root -l | { cat; echo '" + minute + " " + hour +
              "* * * /home/emilian/433Utils/RPi_utils/codesend " + code + "'; } | sudo crontab -u root -")
    return ''
```

Figura 6: Request pentru setarea unui semnal automat

- **deleteautocode** → sterge un semnal radio din crontab pentru a nu mai fi trimis periodic

```
@app.route('/deleteautocode', methods=['POST'])
def deleteautocode():
    code = request.args.get('code')
    minute = request.args.get('minute')
    hour = request.args.get('hour')
    os.system("sudo crontab -u root -l | grep -v '" + minute + " " + hour + ".*" + code + "' | sudo crontab -u root -")
    return ''
```

Figura 7: Request pentru setarea unui semnal automat

Pentru ca serverul sa porneasca automat cand Raspberry Pi-ul este pornit, am adaugat aceasta comanda in fisierul **/etc/rc.local**

```
python3 /home/emilian/server/server.py >> /var/log/smartremote.log
```

Figura 7: Comanda pentru pornirea automata a serverului

3. **Descrierea aplicatiei Android**

3.1 SQLite

Pentru a stoca datele din aplicatie am folosit baza de date SQLite. Printre alte metode[5] de a salva date se mai numara:

- fisiere interne
- memorii externe
- date primitive cu pereche cheie - valoare

Am ales SQLite fiindca aceasta baza de date este bazate pe standardul SQL cu care am mai lucrat, iar interogarile prin SQL ofera destul de multe optiuni.

In cadrul acestui proiect am folosit trei tabele:

1. **Tabs** → stocheaza informatiile despre telecomenzile adaugate in aplicatie. Contine urmatoarele campuri:

   a. position → campul este cheie privata de tip integer, pozitia tabului (se actualizeaza in cazul stergerii unui tab)

   b. tabname → camp de tip text, numele tabului (telecomenzii)

2. **Buttons** → stocheaza informatiile despre butoanele adaugate in aplicatie. Contine urmatoarele campuri:

   a. id → campul este cheie privata de tip integer cu autoincrement, id-ul unui buton

   b. name → camp de tip text, numele care apare pe buton

   c. code → camp de tip integer, stocheaza un cod format din 6 cifre ce contine un semnal radio

   d. tabnumber → camp de tip integer, numarul tabului pe care se afla butoanele

[5] https://developer.android.com/guide/topics/data/data-storage.html

3. Schedule → stocheaza informatiile despre semnale radio programate sa fie trimise periodic la anumite ore. Contine urmatoarele campuri:

a. id → campul este cheie privata de tip integer cu autoincrement, id-ul unui shedule

b. name → camp de tip text, nume schedule

c. code → camp de tip integer, stocheaza un cod format din 6 cifre ce contine un semnal radio

d. hour → camp de tip integer, ora la care sa fie trimis semnalul radio

e. minute → camp de tip integer, minutul la care sa fie trimis semnalul radio

f. state → camp de tip integer, stocheaza 0 daca acest schedule e dezactivat si 1 daca este activat, la momentul crearii campul primeste valoare 1

Pentru folosirea bazei de date am creat o clasa ce extinde "SQLiteOpenHelper", creaza o baza de date cu tabelele descrise mai sus si am implementat urmatoarele metode pentru a reusi sa adaug, modific si selectez datele necesare:

1. **addTab** → retrage numarul de taburi existente si creaza unul nou ce primeste urmatoarea pozitie si numele primit ca argument

```
public void addTab(String tabName){
    SQLiteDatabase db = getWritableDatabase();
    int position;
    String query = "SELECT COUNT(*) FROM " + TABLE_TAB + ";";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    position = cursor.getInt(0);

    ContentValues values = new ContentValues();
    values.put(TAB_POSITION, position);
    values.put(TAB_TABNAME, tabName);
    db.insertOrThrow(TABLE_TAB, null, values);
    db.close();
}
```

Figura 8: Metoda addTab

2. **deleteTab** → functia primeste ca argument o positie, iar dupa ce tabul de pe pozitia respectiva este sters, taburile de pe pozitiile mai mari sunt mutate mai la stanga cu o pozitie; la fel se procedeaza si cu butoanele de pe tabul respectiv.

```
public void deleteTab(int position){
    SQLiteDatabase db = getWritableDatabase();
    db.execSQL("DELETE FROM " + TABLE_TAB + " WHERE " + TAB_POSITION + "=" + position + ";");
    db.execSQL("UPDATE " + TABLE_TAB + " SET " + TAB_POSITION + "=" + TAB_POSITION + " - 1 WHERE " +
            TAB_POSITION + " > " + position + ";");

    db.execSQL("DELETE FROM " + TABLE_BUTTON + " WHERE " + BUTTON_TABNUMBER + "=" + position + ";");
    db.execSQL("UPDATE " + TABLE_BUTTON + " SET " + BUTTON_TABNUMBER + "=" + BUTTON_TABNUMBER + " - 1 WHERE " +
            BUTTON_TABNUMBER + " > " + position + ";");
    db.close();
}
```

Figura 9: Metoda deleteTab

3. **getTabName** → primeste pozitia unui tab, cauta acel tab in tabel si returneaza numele

```
public String getTabName(int position){
    String tabname;
    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_TAB + " WHERE " + TAB_POSITION + "=" + position + ";";

    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    tabname = cursor.getString(cursor.getColumnIndex(TAB_TABNAME));
    db.close();
    return tabname;
}
```

Figura 10: Metoda getTabName

4. **getTabCount** → interogheaza baza de date pentru numarul de taburi existente

```
public int getTabCount(){
    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT COUNT(*) FROM " + TABLE_TAB + ";";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    return cursor.getInt(0);
}
```

Figura 11: Metoda getTabCount

5. **addButton** → primeste ca argument un buton (clasa creata de mine) pe care il adauga in tabelul de butoane

```
public void addButton(Mybutton button){
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(BUTTON_NAME, button.getName());
    values.put(BUTTON_CODE, button.getCode());
    values.put(BUTTON_TABNUMBER, button.getTabnumber());
    db.insertOrThrow(TABLE_BUTTON, null, values);
    db.close();
}
```

Figura 12: Metoda addButton

6. **deleteButton** → primeste ca argument un id si sterge butonul cu id-ul respetiv

```
public void deleteButton(int id){
    SQLiteDatabase db = getWritableDatabase();
    db.execSQL("DELETE FROM " + TABLE_BUTTON + " WHERE " + BUTTON_ID + "=\"" + id + "\";");
    db.close();
}
```

Figura 13: Metoda deleteButton

7. **getButtonByTab** → primeste ca argument pozitia tabului curent, creaza o lista cu butoanele ce se gasesc pe tabul respectiv si returneaza aceasta lista

```
public ArrayList<Mybutton> getButtonsByTab(int position){
    ArrayList<Mybutton> list = new ArrayList<~>();
    String name;
    int id, code, tabNumber;

    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_BUTTON + " WHERE " + BUTTON_TABNUMBER + " = " + position + ";";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();

    while(!cursor.isAfterLast()) {
        id = cursor.getInt(cursor.getColumnIndex(BUTTON_ID));
        name = cursor.getString(cursor.getColumnIndex(BUTTON_NAME));
        code = cursor.getInt(cursor.getColumnIndex(BUTTON_CODE));
        tabNumber = cursor.getInt(cursor.getColumnIndex(BUTTON_TABNUMBER));
        list.add(new Mybutton(id, name, code, tabNumber));
        cursor.moveToNext();
    }

    db.close();
    return list;
```

Figura 14: Metoda getButtonsByTab

8. **addSchedule** → primeste ca argument un schedule (clasa creata de mine) si il adauga in tabelul schedule

```
public void addSchedule(MySchedule schedule){
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(SCHEDULE_NAME, schedule.getName());
    values.put(SCHEDULE_CODE, schedule.getCode());
    values.put(SCHEDULE_HOUR, schedule.getHour());
    values.put(SCHEDULE_MINUTE, schedule.getMinute());
    values.put(SCHEDULE_STATE, 1);
    db.insertOrThrow(TABLE_SCHEDULE, null, values);
    db.close();
}
```

Figura 15: Metoda addSchedule

9. **deleteSchedule** → primeste ca argument un id si sterge din tabel un schedule cu id-ul respectiv

```
public void deleteSchedule(int id){
    SQLiteDatabase db = getWritableDatabase();
    db.execSQL("DELETE FROM " + TABLE_SCHEDULE + " WHERE " + SCHEDULE_ID + "=\"" + id + "\";");
    db.close();
}
```

Figura 16: Metoda deleteSchedule

10. **getSchedule** → interogheaza baza de date pentru toate semnalele programate sa fie trimise automat, le pune intr-o lista si o returneaza

```
public ArrayList<MySchedule> getSchedule(){
    ArrayList<MySchedule> list = new ArrayList<~>();
    String name;
    int id, code, hour, minute;
    boolean state;

    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_SCHEDULE + ";";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();

    while(!cursor.isAfterLast()) {
        id = cursor.getInt(cursor.getColumnIndex(SCHEDULE_ID));
        name = cursor.getString(cursor.getColumnIndex(SCHEDULE_NAME));
        code = cursor.getInt(cursor.getColumnIndex(SCHEDULE_CODE));
        hour = cursor.getInt(cursor.getColumnIndex(SCHEDULE_HOUR));
        minute = cursor.getInt(cursor.getColumnIndex(SCHEDULE_MINUTE));
        if (cursor.getInt(cursor.getColumnIndex(SCHEDULE_STATE)) == 1){
            state = true;
        }else{
            state = false;
        }
        list.add(new MySchedule(id, name, code, hour, minute, state));
        cursor.moveToNext();
    }

    db.close();
    return list;
}
```

Figura 17: Metoda getSchedule

11. **setScheduleState** → primeste un id, o stare (true sau false) si modifica starea pentru schedule-ul cu id-ul respectiv

```
public void setScheduleState(int id, boolean state){
    SQLiteDatabase db = getWritableDatabase();
    int x;

    if (state){
        x = 1;
    }else{
        x = 0;
    }

    db.execSQL("UPDATE " + TABLE_SCHEDULE + " SET " + SCHEDULE_STATE + " = " + x +
                    " WHERE " + SCHEDULE_ID + " = " + id + ";");
    db.close();
}
```

Figura 18: Metoda setScheduleState

12. **getScheduleById** → returneaza schedule-ul cu id-ul primit ca argument

```
public MySchedule getScheduleById(int id){
    SQLiteDatabase db = getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_SCHEDULE + " WHERE " + SCHEDULE_ID + " = " + id + ";";
    Cursor cursor = db.rawQuery(query, null);
    cursor.moveToFirst();
    String name = cursor.getString(cursor.getColumnIndex(SCHEDULE_NAME));
    int code = cursor.getInt(cursor.getColumnIndex(SCHEDULE_CODE));
    int hour = cursor.getInt(cursor.getColumnIndex(SCHEDULE_HOUR));
    int minute = cursor.getInt(cursor.getColumnIndex(SCHEDULE_MINUTE));
    boolean state;
    if (cursor.getInt(cursor.getColumnIndex(SCHEDULE_STATE)) == 1){
        state = true;
    }else{
        state = false;
    }
    db.close();
    return new MySchedule(id, name, code, hour, minute, state);
}
```

Figura 19: Metoda getScheduleById

13. **setScheduleNameAndTime** → schimba numele si timpul pentru un schedule cu id-ul primit

```
public void setScheduleNameAndTime(int id, String name, int hour, int minute){
    SQLiteDatabase db = getWritableDatabase();
    db.execSQL("UPDATE " + TABLE_SCHEDULE + " SET " + SCHEDULE_NAME + " = \"" + name +
            "\", " + SCHEDULE_HOUR + " = " + hour +
            ", " + SCHEDULE_MINUTE + " = " + minute +
            " WHERE " + SCHEDULE_ID + " = " + id + ";");
    db.close();
}
```

Figura 20: Metoda setScheduleNameAndTime

3.2 Volley

Volley[6] este o librarie care ajuta la trimiterea de request-uri HTTP. Volley ofera urmtoarele beneficii:

- programare automata a request-urilor
- multiple conexiuni concurente
- manuirea transparenta a discului si memoriei cu coerenta standard a cache-ului HTTP
- usor de modificat propritatile requestului (ex. durata timeout-ului)
- soport pentru prioritizarea request-urilor
- debugging and tracing tools

Pentru a folosi Volley in acest proiect am adaugat libraria ca dependenta in fisierul **build.grable**: “compile 'com.android.volley:volley:1.1.0'”. Un simplu request HTTP cu ajutorul lui Volley arata astfel:

[6] https://developer.android.com/training/volley/index.html

```
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://192.168.0.140:5000/getcode";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Toast.makeText(getApplicationContext(), "Signal received", Toast.LENGTH_SHORT).show();
                code = response.toString();
            }
        }, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), "Something went wrong!", Toast.LENGTH_SHORT).show();
        startActivity(new Intent(AddButton.this, MainActivity.class));
    }
});
stringRequest.setRetryPolicy(new DefaultRetryPolicy(10000, 1, 1.0f));
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Figura 8: Volley HTTP Request

Pentru a seta adresa la care se face request-ul, variabila **url** de tip String primeste adresa iar aceasta este data ca parametru al request-ului prin variabila **stringRequest**. Un alt parametru primit este tipul request-ului in cazul de mai sus acesta fiind de tip **GET**, atribuit prin “Request.Method.GET”. In continuare se defineste metoda **onResponse** care este apelata in momentul in care se primeste un raspuns de la request. In caz de eroare este apelata metoda **onErrorResponse**.

Dupa definirea request-ului se poate seta prin setRetryPolicy:

1. durata asteptarii unui raspuns inainte de a arunca timeout, in cazul de sus 10 secunde prin primul argument cu valoarea de “10000”

2. numarul de incercari al request-ului daca acesta esueaza, definit prin al doilea argument cu valoarea “1”

3. multiplicator back off, definit prin al treilea argument cu valoarea “1.0f”

3.3 Pagina Principala

Pagina principala a aplicatiei este o activitate cu taburi (telecomenzi) ce sunt adaugate de catre utilizator pe parcurs. Cand aplicatia este deschisa pentru prima data, ea arata astfel:

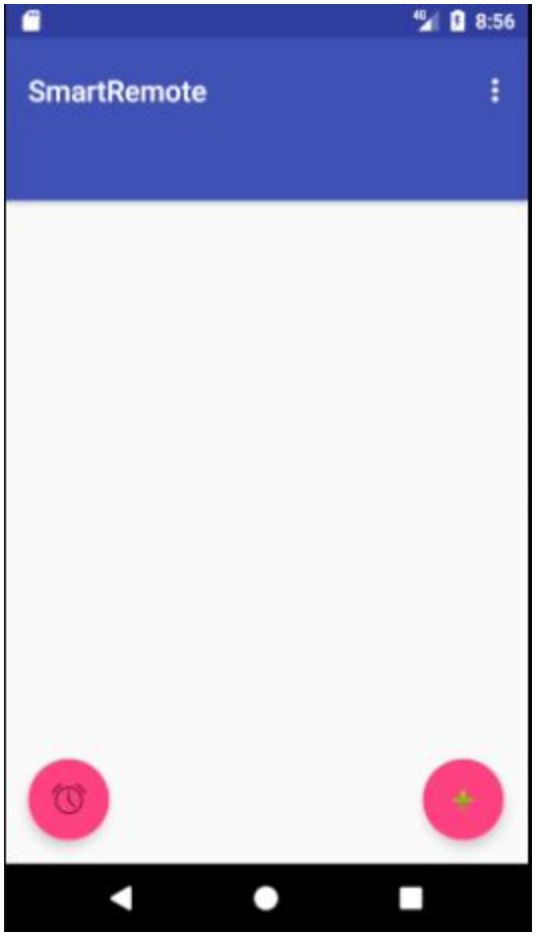

Figura 25: Pagina principala goala

In partea de jos se pot observa doua butoane, cel din stanga duce la pagina de schedule iar cel din dreapta adauga un buton nou. Prin apasarea celor 3 puncte din coltul dreapta sus se deschide un meniu cu doua optiuni:

- Add a tab → deschide o pagina noua pentru a denumi noul tab
- Delete this tab → sterge tabul selectat current

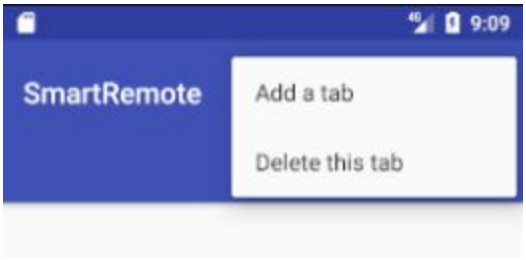

Figura 26: Meniu pagina principala

Actiunile acestui meniu sunt definite prin suprascrierea metodei **onOptionsItemSelected**

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    if (id == R.id.add_tab) {
        startActivity(new Intent(MainActivity.this, AddTab.class));
        return true;
    }

    if (id == R.id.delete_tab) {
        int position = mViewPager.getCurrentItem();
        dbhandler.deleteTab(position);
        startActivity(new Intent(MainActivity.this, MainActivity.class));
        Toast.makeText(getApplicationContext(), "Tab "+ (++position) + " Deleted", Toast.LENGTH_SHORT).show();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Figura 27: onOptionsItemSelected

Cand unul din butoane este apasat, variabila **id** primeste id-ul elementului apasat iar codul din interiorul if-ului corespunzator este apelat.

3.4 Adaugarea unui tab

Selectand “Add a tab” din meniul prezentat mai sus, se deschide o noua pagina ce permite setarea numelui.

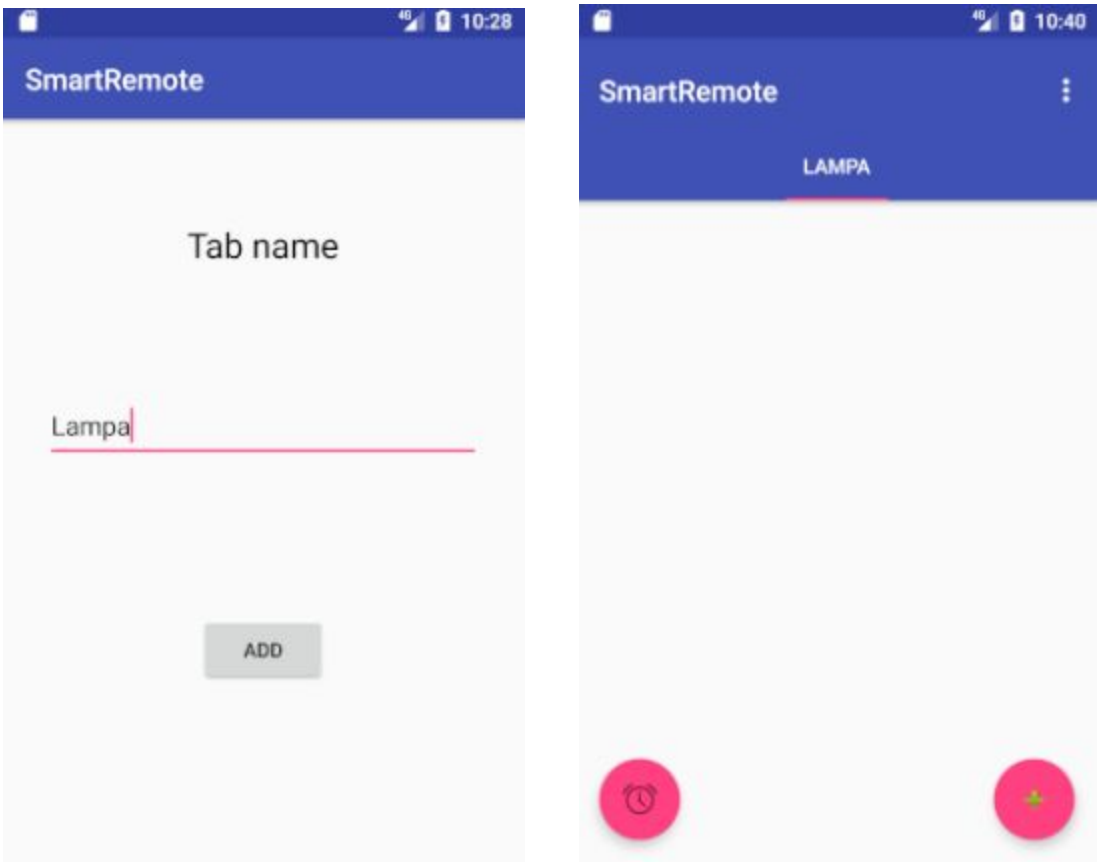

Figura 28: Adaugarea unui tab

La apasarea butonului de Add se executa urmatorul cod ce duce la adaugarea tabului in tabelul de taburi cu ajutorul metodei addTab descrisa mai sus.

```
public void addButtonClicked(View view){
    //Tab tab = new Tab(namefield.getText().toString());
    dbhandler.addTab(namefield.getText().toString());
    startActivity(new Intent(AddTab.this, MainActivity.class));
}
```

Figura 29: Codul pentru adaugarea unui tab

Dupa adaugarea tabului in baza de date, pagina principala este redeschisa prin apelarea **startActivity** si noul tab poate fi observat.

3.5 Adaugarea unui buton

Acum ca a fost adaugat un tab, se poate adauga un buton prin apasarea butonul din dreapta jos. Acest lucru duce la incarcarea unei noi pagini ce permite copierea unui semnal radio si setarea numelui pentru buton.

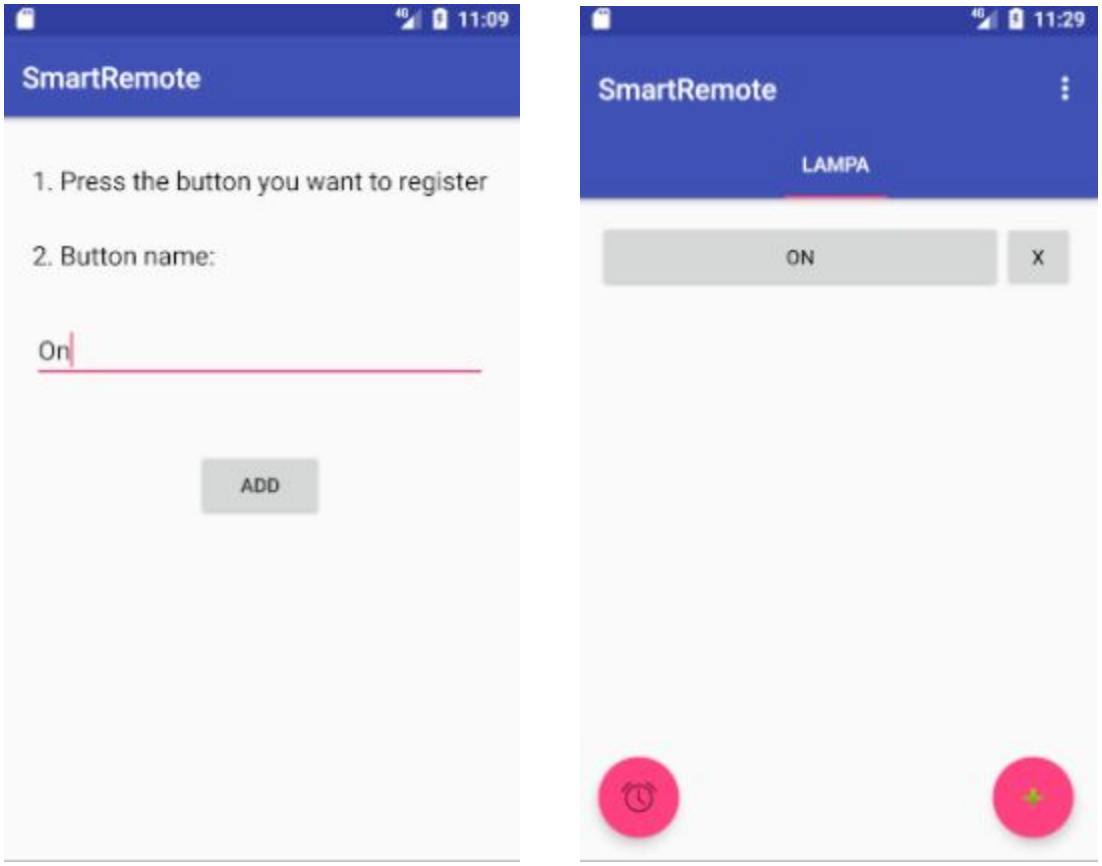

Figura 29: Adaugarea unui buton

La incarcarea acestei pagini aplicatia trimite un request catre Raspberry Pi pentru a asculta dupa un semnal radio. In acest moment se poate indrepta telecomanda spre Raspberry Pi si prin apasarea butonului, receptorul va copia semnalul, il va trimite aplicatiei Android, iar aceasta va semnala utilizatorul cu un mesaj ca semnalul a fost inregistrat. Dupa introducerea

numelui si apasarea butonului de Add, se apeleaza metoda addButton a clasei ce administreaza baza de date si pagina principala este reincarcata.

Pentru a adauga butoanele in pagina dinamic, am folosit un TableLayout in care adaug doua butoane pe fiecare rand:

- unul care face request la Raspberry Pi pentru a trimite semnalul radio
- unul care sa stearga butonul

```
TableLayout tl = (TableLayout) view.findViewById(R.id.tableLayout);
tl.setStretchAllColumns(true);
ArrayList<Mybutton> listButton = dbhandler.getButtonsByTab(getArguments().getInt(ARG_SECTION_NUMBER) - 1);
Iterator itr = listButton.iterator();

while(itr.hasNext()){
    TableRow tr = new TableRow(getActivity());
    tr.setLayoutParams(new TableLayout.LayoutParams(TableLayout.LayoutParams.MATCH_PARENT, TableLayout.LayoutParams.WRAP_CONTENT));
    tr.setWeightSum(1f);
    final Mybutton myButton = (Mybutton) itr.next();

    Button button1 = new Button(getActivity());
    TableRow.LayoutParams params = new TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT);
    params.weight = 0.85f;
    button1.setLayoutParams(params);
    button1.setText(myButton.getName());
    button1.setOnClickListener((view) -> {
            RequestQueue queue = Volley.newRequestQueue(getActivity().getApplicationContext());
            String url ="http://192.168.0.140:5000/sendcode?code=" + myButton.getCode();
            StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                    new Response.Listener<String>() {
                        @Override
                        public void onResponse(String response) {
                        }
                    }, (error) -> {
                    Toast.makeText(getActivity().getApplicationContext(), "Something went wrong!", Toast.LENGTH_SHORT).show();
            });
            queue.add(stringRequest);
    });
    tr.addView(button1);

    Button button2 = new Button(getActivity());
    params = new TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT);
    params.weight = 0.15f;
    button2.setLayoutParams(params);
    button2.setText("X");
    button2.setOnClickListener((view) -> {
            dbhandler.deleteButton(myButton.getId());
            startActivity(new Intent(getActivity(), MainActivity.class));
            Toast.makeText(getActivity().getApplicationContext(), "Button Deleted", Toast.LENGTH_SHORT).show();
    });
    tr.addView(button2);
    tl.addView(tr, new TableLayout.LayoutParams(TableLayout.LayoutParams.MATCH_PARENT, TableLayout.LayoutParams.WRAP_CONTENT));
}
```

Figura 30: Crearea dinamica a butoanelor

Secventa de cod de mai sunt este folosita pentru adaugarea dinamica a butoanelor pe ecran. Se declara un TableLayout ce va cuprinde butoanele. In continuare se interogheaza baza de date, prin metoda **getButtonsByTab** descrisa in subcapitolul 3.1, pentru a primi lista de butoane ce apartin tabului curent. Folosind un iterator, am parcurs lista si la fiecare pas am creat un nou rand pentru tabel. Dupa aceasta am definit un buton ce are ca rol executarea unui request

catre Raspberry Pi pentru a trimite semnalului radio corespunzator. Urmatorul buton definit este cel care sterge un semnal din baza de date prin metoda **deleteButton** descrisa in subcapitolul 3.1. Aceste doua butoane sunt adaugate pe un TableRow , iar in final randul este adaugat in TableLayout.

Bibliografie

Raspberry Pi

https://www.raspberrypi.org/learning/software-guide/quickstart/

https://www.princetronics.com/how-to-read-433-mhz-codes-w-raspberry-pi-433-mhz-receiver/

Python

http://flask.pocoo.org/

Android

https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl

https://developer.android.com/training/volley/simple.html

https://stackoverflow.com/questions/8700427/add-buttons-to-tablerow-dynamically

https://stackoverflow.com/questions/10576307/android-how-do-i-correctly-get-the-value-from-a-switch