UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

Facultatea de Informatică

Lucrare de Licenţă

Smart Remote

Propusă de

Mitocariu Ioan-Emilian

Sesiunea: Februarie, 2018

Coordonator Ştiinţific

Lector dr. Anca Ignat

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

Facultatea de Informatică

Smart Remote

Mitocariu Ioan-Emilian

Sesiunea: Februarie 2018

Coordonator Ştiinţific

Lector dr. Anca Ignat

DECLARAŢIE 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 instituţie de învăţământ superior din ţară 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 deţin referinţa precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
- codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
- rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi,

Absolvent Mitocariu Ioan-Emilian

_____

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „Smart Remote”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii 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 licenţă.

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. Motivaţie

Multe lucruri din ziua de astăzi, cum ar fi becuri, aerul condiţionat, prize şi multe altele, pot fi controlate prin telecomenzi care folosesc frecvenţe radio pentru a transmite diferite comenzi. Prin simpla apăsare a unui buton poţi închide un bec sau porni aerul condiţionat fără a fi nevoit sa te mişti atât timp cât ai telecomanda la îndemâna.

Dar în momentul în care ai mai multe astfel de dispozitive în casa, care pot fi controlate de la distanţa, vei avea de asemenea şi mai multe telecomenzi. Acest lucru duce la incomoditatea de a trebui să căram mai multe telecomenzi.

Deoarece trăim într-o lume în care aproape toţi oamenii au un smartphone pe care îl poartă peste tot cu ei, ar fi mult mai uşor şi comod să putem controla anumite dispozitive din jurul nostru de pe telefonul personal.

Am ales această temă fiindcă doresc să aduc o soluţie prin care să putem înlocui telecomenzile fizice cu telecomenzi virtuale ce sunt mult mai la îndemâna.

2. Context

Una dintre primele utilizări a undelor radio pentru controlul de la distanţa a fost făcută de către Nikola Tesla în 1898[1], care a controlat o barcă în miniatura. Aceasta avea o antenă prin care recepta comenzi, iar ele erau trimise printr-o cutie ce permitea controlarea vitezei şi a direcţiei.

După anii 1930-1940 telecomenzile pe bază de frecvenţe radio au început uşor să fie folosite pentru a controla dispozitive din jurul casei, unul dintre exemple fiind uşa de la garaj.

---

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

Telecomenzile pe bază de unde radio încă sunt folosite şi astăzi, ele devenind mai mici şi mai accesibile oamenilor. Din dorinţa de a controla diferite dispozitive de la distanţa prin telefon, au început sa fie folosite tehnologii asemănătoare, cum ar fi Bluetooth şi Wi-Fi care sunt suportate de către telefoane în mod direct.

În acest context apare problema în care vrem să controlăm un dispozitiv ce primeşte comenzi prin frecvenţe radio, dar un smartphone obişnuit nu are posibilitatea de a trimite astfel de comenzi.

3. Descrierea sumară a soluţiei

Aplicaţia dezvoltată de mine oferă şansa de a porta telecomenzile fizice, ce controlează obiectele din jurul casei, pe telefon. În plus aplicaţia oferă şi posibilitatea de a programa comenzi ce vor fi trimise periodic la anumite ore. Acest lucru duce la o controlare mult mai uşoară a obiectelor din casă, toate comenzile fiind centralizate pe telefonul personal.

Din moment ce un smartphone obişnuit nu poate trimite comenzi prin frecvenţe radio, este nevoie de un obiect intermediar ce poate comunica atât cu dispozitive ce folosesc frecvenţe radio cât şi cu un telefon. După cum se poate vedea în Figura 1, soluţia este de a folosi un Raspberry Pi ce răspunde la request-uri HTTP de la aplicaţia android şi comunică cu celelalte dispozitive prin semnale radio.

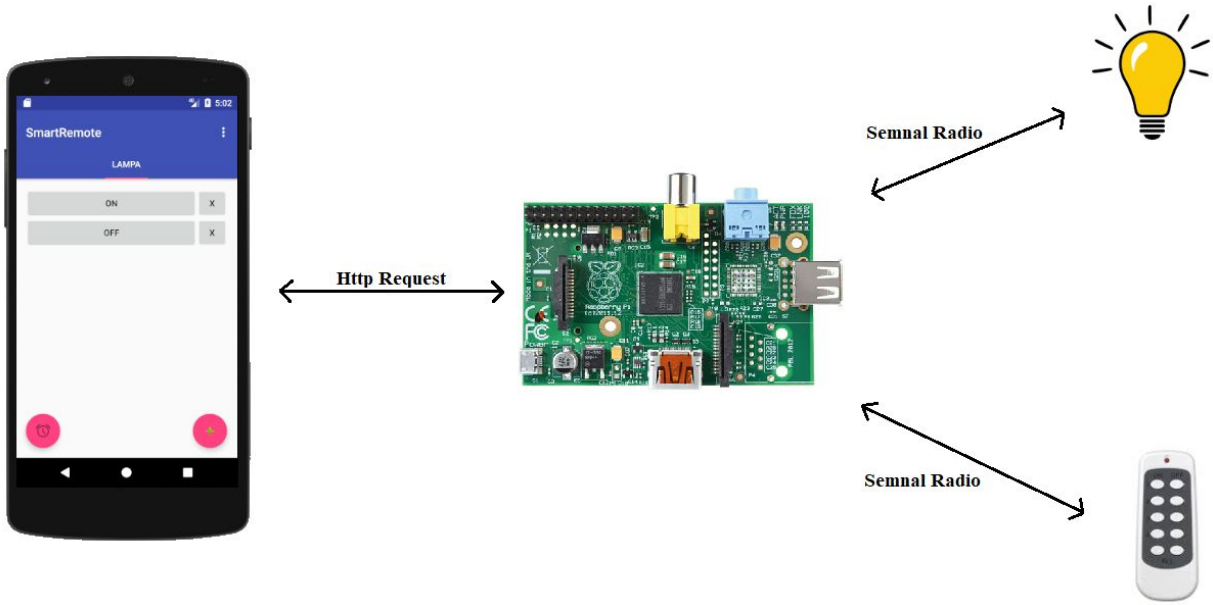

Figura 1: Diagrama aplicaţiei

Pentru această soluţie Raspberry Pi-ul necesită folosirea următoarelor:

- un receptor radio de frecvenţă 433MHz pentru copierea comenzilor de pe telecomandă
- un transmiţător radio de frecvenţă 433MHz pentru trimiterea comenzilor către dispozitive
- un adaptor USB pentru Wi-Fi

Controlarea pinilor **GPIO** de pe Raspberry Pi se face cu ajutorul pachetului **WiringPi**. La aceşti pini se conectează receptorul şi transmiţătorul radio care la rândul lor sunt controlaţi printr-un proiect open source **433Utils**[2] de la **NinjaBlock**.

Ca Raspberry Pi sa poată comunica cu telefonul, el se conectează prin Wi-Fi la un router local. Pe Raspberry Pi am folosit **Python** împreună cu microframework-ul **Flask** pentru a crea un server ce răspunde la request-uri **HTTP**.

În final am dezvoltat o aplicaţie de telefon pentru sistemul de operare Android. În acest caz am ales **Java**, fiind limbajul oficial şi cel mai folosit pentru dezvoltare de aplicaţii Android. Am ales să lucrez în **Android Studio** pentru faptul că suporta Java şi este de asemenea IDE-ul oficial pentru Android. Pentru trimiterea de request-uri către serverul de pe Raspberry Pi am folosit librăria **Volley**.

Ca sistem de administrare a bazelor de date relaţionale am folosit SQLite, principalul sistem nativ oferit de Android. El este folosit pentru stocarea de informaţii, cum ar fi lista de telecomenzi, butoane, codurile frecvenţelor radio atribuite fiecărui buton şi comenzile programate pentru a fi trimise periodic.

4. Structura lucrării

Lucrarea de licenţă va avea două capitole în care se va descrie în detaliu procesul de dezvoltare al aplicaţiei şi cum funcţionează:

- Configurarea Raspberry Pi-ului
- Descrierea aplicaţiei Android

Primul capitol descrie paşii necesari pentru configurarea Raspberry Pi-ului. De la

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

instalarea sistemului de operare şi controlarea receptorului/transmiţătorului radio până la crearea server-ului în Python ce răspunde la comenzi.

Cel de-al doilea capitol va descrie tehnologiile folosite şi va explica cum funcţionează aplicaţia SmartRemote, folosind capturi de ecran cu blocuri de cod şi pagini din aplicaţie.

1. Contribuţii

Contribuţiile mele pentru această lucrare încep de la căutarea de informaţii despre cum funcţionează Raspberry Pi-ul, cum se configurează unul şi cum pot controla transmiţătorul/receptorul conectaţi prin pinii GPIO. Aceste informaţii vor fi disponibile la sfârşitul lucrării în cadrul Bibliografiei.

După ce informaţiile necesare au fost găsite şi acumulate, am început să le pun în practică prin punerea în funcţiune a Raspberry Pi-ului. După ce am reuşit să controlez receptorul şi transmiţătorul radio cu ajutorul unor pachete open source am creat un server ce răspunde la request-urile făcute de către un client, în cazul meu aplicaţia Android.

Ultimul pas din contribuţiile mele aduse acestui proiect este crearea unei aplicaţii android. Aceasta oferă posibilitatea de a înregistra semnalele radio trimise de telecomenzi, cu ajutorul Raspberry Pi-ului, ce sunt apoi stocate şi oferite utilizatorului sub formă de butoane virtuale. Prin aplicaţie se pot de asemenea programa semnale radio care să fie trimise automat la o anumită oră.

Sumar al contribuţiilor mele în cadrul acestei lucrări:

- configurarea Raspberry Pi-ului
- implementarea server-ului de pe Raspberry Pi
- dezvoltarea aplicaţiei 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 în United Kingdom şi a fost proiectat pentru învăţarea noţiunilor de bază ale informaticii. A devenit mult mai popular decât a fost anticipat şi este adesea folosit în proiecte de robotica şi home automation (ca şi această lucrare).

În acest proiect voi folosi un Raspberry Pi Model A. Pentru a fi pus în funcţiune 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 distribuţie de Linux bazată pe Debian. Pentru a instala sistemul de operare am formatat cardul SD, am descărcat imaginea pentru Raspbian şi am folosind un cititor de carduri SD împreună cu Etcher.io am "ars" imaginea pe card. Acum Raspberry Pi-ul este gata pentru a fi pornit. Pentru ca IP-ul să fie static şi să se conecteze la router am modificat fişierul **/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 2: Configurarea unui IP static

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

Unul dintre atuurile unui Raspberry Pi sunt pinii **GPIO** care pot fi folosiţi pentru a conecta senzori, transmiţători, receptori, lumini şi alte dispozitive. Pe lângă pinii GPIO se mai găsesc pinii de alimentare cu 3.3V respectiv 5V şi pinii de împământare. Pentru modelul folosit de mine ei sunt aranjaţi astfel:

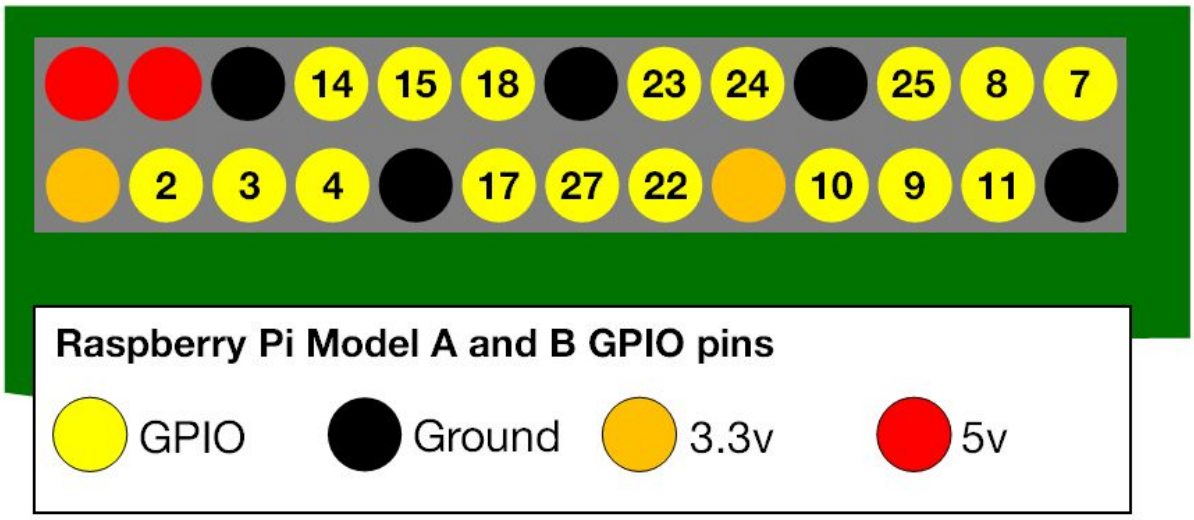

Figura 3: Pinii de pe Raspberry Pi model A

Acum e timpul să conectăm receptorul şi transmiţătorul de frecvenţa radio 433MHz la Raspberry Pi. Amândoi vor avea nevoie să se conecteze la 3 pini:

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

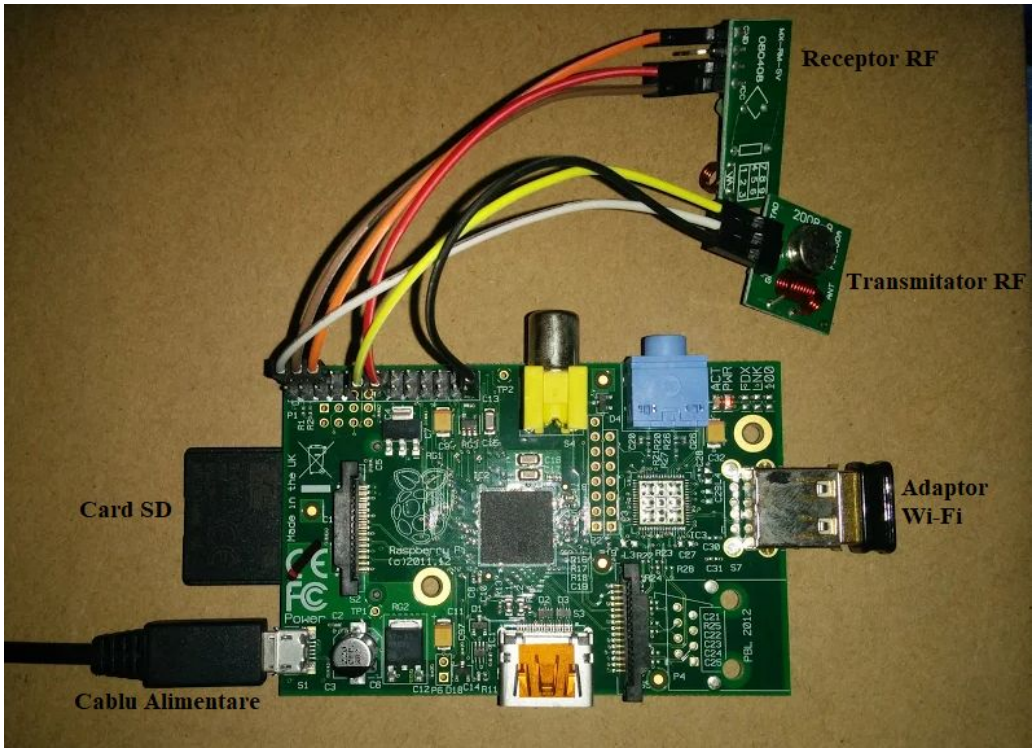

Figura 4: Raspberry Pi cu toate componentele conectate

Pentru a putea controla pinii GPIO am folosit librăria **WiringPi** care este scrisă în limbajul C. Pentru a controla receptorul şi transmiţătorul am folosit un proiect open source numit **433Utils** de la NinjaBlocks care îmi permite să ascult sau să trimit semnale radio.

2.2 HTTP Server

Am ales să implementez serverul de pe Raspberry Pi care va răspunde la request-uri HTTP în **Python** folosind microframework-ul **Flask**[4]. Acest framework este bazat pe Werkzeug si Jinja2. Câteva aplicaţii mai cunoscute care folosesc Flask sunt Pinterest şi LinkedIn. Am ales acest framework deoarece oferă 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** → porneşte receptorul radio, copie primul semnal primit şi il returnează 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 5: Request pentru a înregistra un semnal

- **sendcode** → trimite un cod ce formează un semnal radio cu ajutorul transmiţătorului 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 6: Request pentru a trimite un semnal

- **setautocode** → programează cu crontab un semnal radio care să fie trimis la o anumită oră

```
@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 7: Request pentru setarea unui semnal automat

- **deleteautocode** → șterge 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 8: Request pentru setarea unui semnal automat

Pentru ca serverul să pornească automat când Raspberry Pi-ul este pornit, am adăugat această comandă în fișierul **/etc/rc.local**

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

Figura 9: Comanda pentru pornirea automată a serverului

3. Descrierea aplicaţiei Android

3.1 SQLite

Pentru a stoca datele din aplicaţie am folosit baza de date SQLite. Am ales SQLite fiindcă această bază de date este bazată pe standardul SQL cu care sunt familiarizat, iar interogările prin SQL oferă destul de multe opţiuni.

Printre alte metode[5] de a salva date se mai număra:

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

În cadrul acestui proiect am folosit trei tabele:

1. **Tabs** → stochează informaţiile despre telecomenzile adăugate în aplicaţie. Conţine următoarele câmpuri:

a. position → câmpul este cheie privată de tip integer, poziţia tabului (se actualizează în cazul ştergerii unui tab)

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

2. **Buttons** → stochează informaţiile despre butoanele adăugate în aplicaţie. Conţine următoarele câmpuri:

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

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

c. code → câmp de tip integer, stochează un cod format din 6 cifre ce conţine un semnal radio

d. tabnumber → câmp de tip integer, numărul tabului pe care se află butoanele

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

3. Schedule → stochează informaţiile despre semnale radio programate să fie trimise periodic la anumite ore. Conţine următoarele câmpuri:

a. id → câmpul este cheie privată de tip integer cu autoincrement, id-ul unui schedule

b. name → câmp de tip text, nume schedule

c. code → câmp de tip integer, stochează un cod format din 6 cifre ce conţine un semnal radio

d. hour → câmp de tip integer, ora la care să fie trimis semnalul radio

e. minute → câmp de tip integer, minutul la care să fie trimis semnalul radio

f. state → câmp de tip integer, stochează 0 dacă acest schedule e dezactivat şi 1 dacă este activat, la momentul creării câmpul primeşte valoarea 1

Pentru folosirea bazei de date am creat o clasă ce extinde "SQLiteOpenHelper". În cadrul acestei clase creez baza de date cu tabelele descrise mai sus şi am implementat următoarele metode pentru a reuşi să adaug, modific şi selectez datele necesare:

1. **addTab** → retrage numărul de taburi existente şi crează unul nou ce primeşte următoarea poziţie şi 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 10: Metoda addTab

2. **deleteTab** → funcţia primeşte ca argument o poziţie, iar după ce tabul de pe poziţia respectivă este şters, taburile de pe poziţiile mai mari sunt mutate mai la stânga cu o poziţie; de asemenea se şterg butoanele necesare iar celelalte sunt actualizate

```
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 11: Metoda deleteTab

3. **getTabName** → primeşte poziţia unui tab, caută acel tab în tabel şi returnează 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 12: Metoda getTabName

4. **getTabCount** → interoghează baza de date pentru numărul 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 13: Metoda getTabCount

5. **addButton** → primeşte ca argument un buton (clasa Mybutton creată de mine) pe care îl adaugă în 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 14: Metoda addButton

6. **deleteButton** → primeşte ca argument un id şi şterge butonul cu id-ul respectiv

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

Figura 15: Metoda deleteButton

7. **getButtonByTab** → primeşte ca argument poziţia tabului curent, crează o listă cu butoanele ce se găsesc pe tabul respectiv şi returnează această listă

```
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 16: Metoda getButtonsByTab

8. **addSchedule** → primeşte ca argument un schedule (clasa MySchedule creată de mine) şi îl adaugă în 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 17: Metoda addSchedule

9. **deleteSchedule** → primeşte ca argument un id şi şterge 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 18: Metoda deleteSchedule

10. **getSchedule** → interoghează baza de date pentru toate semnalele programate să fie trimise automat, le pune într-o listă și o returnează

```
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 19: Metoda getSchedule

11. **setScheduleState** → primeşte un id, o stare (true sau false) şi modifică 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 20: Metoda setScheduleState

12. **getScheduleById** → returnează 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 21: Metoda getScheduleById

13. **setScheduleNameAndTime** → schimbă numele şi 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 22: Metoda setScheduleNameAndTime

3.2 Volley

Volley[6] este o librărie care ajută la trimiterea de request-uri HTTP. Volley oferă următoarele beneficii:

- programare automată a request-urilor
- multiple conexiuni concurente
- mânuirea transparentă a discului şi memoriei cu coerenta standard a cache-ului HTTP
- uşor de modificat proprietăţile request-ului (ex. durata timeout-ului)
- suport pentru prioritizarea request-urilor
- unelte pentru debugging şi tracing

Pentru a folosi Volley în acest proiect am adăugat librăria ca dependentă în fişierul **build.grable**: “compile 'com.android.volley:volley:1.1.0”. Un simplu request HTTP cu ajutorul lui Volley arată 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 23: Volley HTTP Request

Pentru a seta adresa la care se face request-ul, variabila **url** de tip String primește adresa iar aceasta este dată ca parametru al request-ului prin variabila **stringRequest**. Un alt parametru primit este tipul request-ului în cazul de mai sus acesta fiind de tip **GET**, atribuit prin "Request.Method.GET". În continuare se definește metoda **onResponse** care este apelată în momentul în care se primește un răspuns de la request. În caz de eroare este apelată metoda **onErrorResponse**.

Apoi, proprietățile request-ului se poate seta prin setRetryPolicy:

1. durata așteptării unui răspuns înainte de a arunca timeout, în cazul de sus 10 secunde prin primul argument cu valoarea de "10000"

2. numărul de încercări al request-ului dacă acesta eșuează, definit prin al doilea argument cu valoarea "1"

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

3.3 Pagina Principală

Pagina principală a aplicaţiei este o activitate cu taburi (telecomenzi) ce sunt adăugate de către utilizator pe parcurs. Când aplicaţia este deschisă pentru prima dată, ea arată astfel:

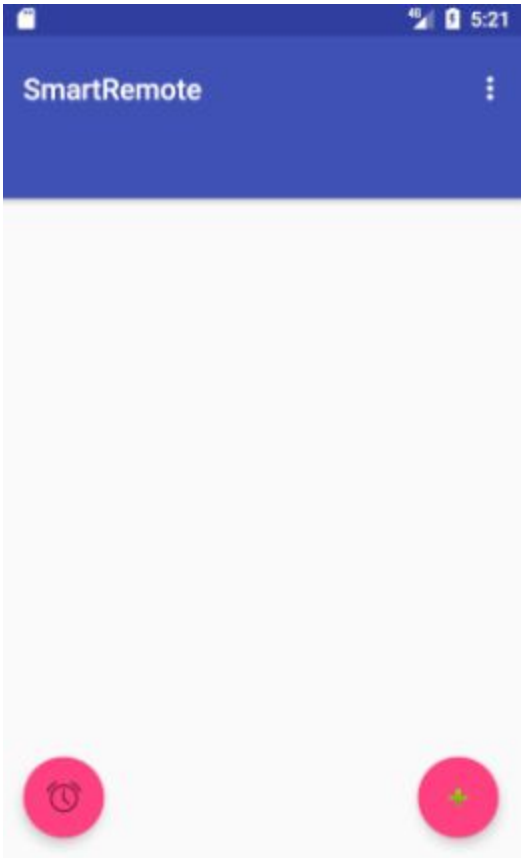

Figura 24: Pagina principală goală

În partea de jos se pot observa două butoane, cel din stânga duce la pagina de schedule iar cel din dreapta adaugă un buton nou. Prin apăsarea celor 3 puncte din coltul dreapta sus se deschide un meniu cu două opţiuni:

- Add a tab → deschide o pagină nouă pentru a denumi noul tab
- Delete this tab → şterge tabul selectat curent

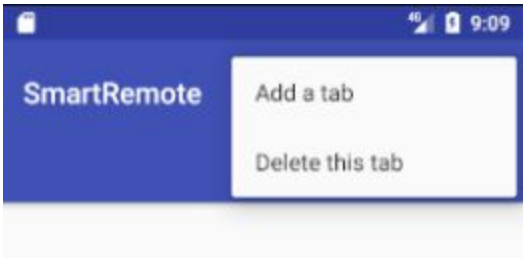

Figura 25: Meniul paginii principale

Acţiunile 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) {
        if(dbhandler.getTabCount() == 0){
            Toast.makeText(getApplicationContext(), "There is no tab", Toast.LENGTH_SHORT).show();
        } else {
            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 26: onOptionsItemSelected

Când unul din butoane este apăsat, variabila **id** primeşte id-ul elementului apăsat iar codul din interiorul if-ului corespunzător este apelat.

În cazul în care butonul “Add a tab” este selectat, aplicaţia deschide o nouă pagină pentru adăugarea tabului. Această pagină este descrisă în următorul subcapitol.

În cazul în care butonul “Delete this tab” este selectat aplicaţia verifică mai întâi dacă există taburi. Dacă nu există nici un tab atunci utilizatorul este notificat printr-un mesaj. Dacă

există cel puţin un tab, atunci variabila **position** primeşte poziţia tabului curent, se apelează metoda **deleteTab** care şterge tabul din baza de date şi pagina este reîncărcata pentru a se observa schimbarea.

3.4 Adăugarea unui tab

Selectând "Add a tab" din meniul prezentat mai sus, se deschide o nouă pagină ce permite setarea numelui.

Figura 27: Adăugarea unui tab

În momentul apăsării butonului "ADD", se apelează funcţia din Figura 28. Aceasta verifică dacă un nume a fost setat, iar în cazul în care utilizatorul nu a introdus nici un nume, el este notificat să facă acest lucru.

```
public void addButtonClicked(View view){
    String tabName = namefield.getText().toString();
    if (tabName.isEmpty()){
        Toast.makeText(getApplicationContext(), "Provide a name", Toast.LENGTH_SHORT).show();
    }else {
        dbhandler.addTab(namefield.getText().toString());
        startActivity(new Intent(AddTab.this, MainActivity.class));
    }
}
```

Figura 28: Codul pentru adăugarea unui tab

În cazul afirmativ, un nou tab este adăugat în baza de date apelând **addTab**. După adăugarea tabului în baza de date, pagina principală este redeschisă prin apelarea **startActivity** şi noul tab poate fi observat.

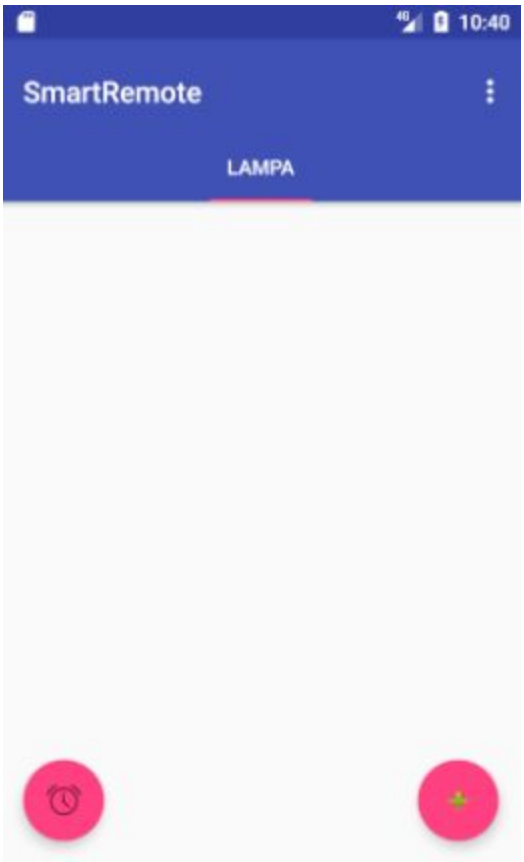

Figura 29: Pagina principala

3.5 Adăugarea unui buton

Acum că a fost adăugat un tab, se poate adaugă primul buton prin apăsarea butonul din dreapta jos de pe pagina principală. Acest lucru duce la încărcarea unei noi pagini ce permite copierea unui semnal radio prin Raspberry Pi şi setarea numelui pentru buton.

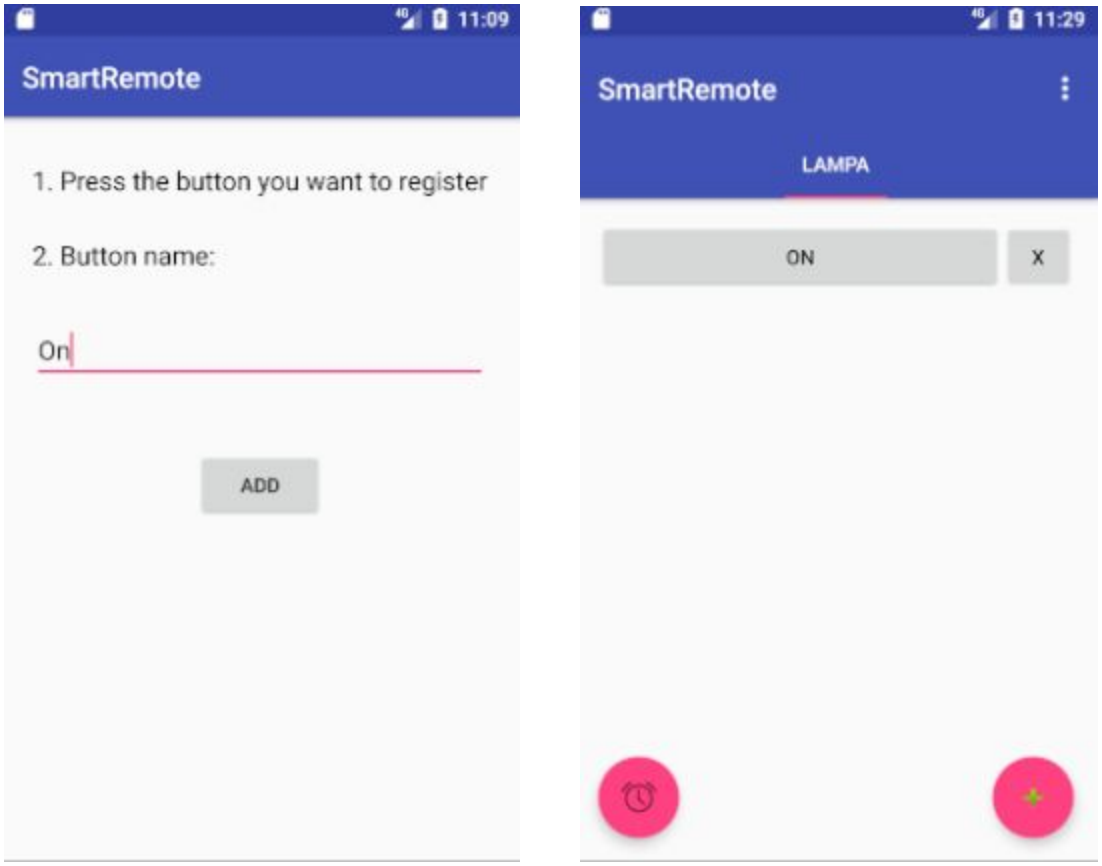

Figura 30: Adăugarea unui buton

La încărcarea acestei pagini aplicaţia trimite un request către Raspberry Pi pentru a asculta după un semnal radio.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_button);
    namefield = (EditText) findViewById(R.id.namefield);
    dbhandler = new MyDBHandler(this, null, null, 1);

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

    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));
    queue.add(stringRequest);
}
```

Figura 31: Codul pentru adăugarea unui buton

În acest moment se poate îndrepta telecomanda spre Raspberry Pi şi prin apăsarea butonului, receptorul va copia semnalul radio, îl va trimite aplicaţiei Android, iar aceasta va semnala utilizatorul cu un mesaj că semnalul a fost înregistrat. În cazul în care request-ul primeşte o eroare, utilizatorul este notificat şi redirecţionat înapoi la pagina principală pentru a exclude posibilitatea adăugării unui buton fără un cod/semnal atribuit.

După introducerea numelui şi apăsarea butonului de “Add”, se apelează metoda addButton a clasei ce administrează baza de date şi pagina principală este reîncărcata.

Pentru a adaugă butoanele în pagină dinamic, am folosit un TableLayout în care adaug două butoane pe fiecare rând:

- unul care face request la Raspberry Pi pentru a trimite semnalul radio
- unul care să şteargă 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 32: Crearea dinamica a butoanelor

Secvenţa de cod de mai sus este folosită pentru adăugarea dinamică a butoanelor pe ecran. Se declară un TableLayout ce va cuprinde butoanele. În continuare se interoghează baza de date, prin metoda **getButtonsByTab** descrisă în subcapitolul 3.1, pentru a primi lista de butoane ce aparţin tabului curent. Folosind un iterator, am parcurs lista şi la fiecare pas am creat un nou rând pentru tabel. După aceasta am definit un buton ce are ca rol executarea unui request către Raspberry Pi pentru a trimite semnalul radio corespunzător. Următorul buton definit este cel care şterge un semnal din baza de date prin metoda **deleteButton** descrisă în subcapitolul 3.1. Aceste două butoane sunt adăugate pe un TableRow , iar în final rândul este adăugat în TableLayout.

3.6 Pagina cu semnale automate

Această pagină, denumita **Schedule** în codul aplicaţiei, afişează toate semnalele ce au fost înregistrare pentru a fi trimise periodic la o anumită oră. Pentru a ajunge pe această pagină trebuie apăsat butonul din stânga jos de pe pagina principală.

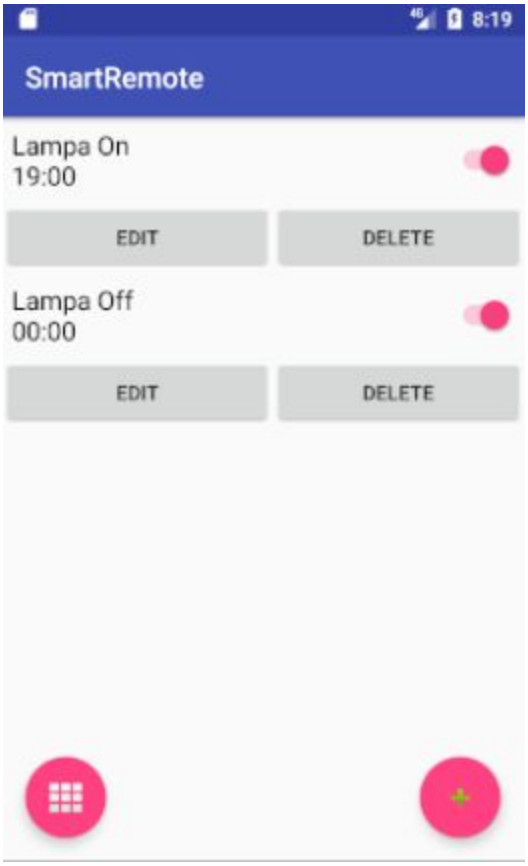

Figura 33: Pagina cu semnale automate

Pe această pagină utilizatorul poate să vadă ce semnale au fost programate, identificate printr-un nume, şi la ce oră vor fi trimise. Pentru a afişa datele pe ecran am folosit, la fel ca pe pagina principală, un **TableLayout** în care adaug pentru fiecare semnal:

1. Un switch ce conţine numele, ora la care va fi trimis semnalul şi un întrerupător ce permite activarea şi dezactivarea semnalului. La apăsarea întrerupătorului se verifică starea lui şi dacă aceasta devine **true** atunci se actualizează starea lui în baza de date şi se trimite un request către Raspberry Pi pentru a seta un crontab corespunzător, iar în cazul

în care devine **false** atunci se trimite un request ce şterge crontab-ul. Motivul pentru care semnalele automate sunt setate să ruleze de pe Raspberry Pi şi nu de pe telefon este că astfel ele vor fi rulate indiferent dacă telefonul este conectat la router sau chiar dacă el este închis.

```
Switch switchSchedule = new Switch(this);
TableRow.LayoutParams params = new TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT);
params.weight = 1f;
params.setMargins(20,20,20,20);
switchSchedule.setLayoutParams(params);
switchSchedule.setTextSize(18);
switchSchedule.setText(mySchedule.getName() + "\n" + hour + ":" + minute);
switchSchedule.setChecked(mySchedule.isState());
switchSchedule.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked == true){
            dbhandler.setScheduleState(mySchedule.getId(), true);
            RequestQueue queue = Volley.newRequestQueue(Schedule.this);
            String url ="http://192.168.0.140:5000/setautocode?code=" + mySchedule.getCode()
                    + "&minute=" + mySchedule.getMinute() + "&hour=" + mySchedule.getHour();
            StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                    new Response.Listener<String>() {...}, (error) → {
                    Toast.makeText(Schedule.this, "Something went wrong!", Toast.LENGTH_SHORT).show();
            });
            queue.add(stringRequest);
        }else{
            dbhandler.setScheduleState(mySchedule.getId(), false);
            RequestQueue queue = Volley.newRequestQueue(Schedule.this);
            String url ="http://192.168.0.140:5000/deleteautocode?code="
                    + mySchedule.getCode() + "&minute=" + mySchedule.getMinute() + "&hour=" + mySchedule.getHour();
            StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                    new Response.Listener<String>() {...}, (error) → {
                    Toast.makeText(Schedule.this, "Something went wrong!", Toast.LENGTH_SHORT).show();
            });
            queue.add(stringRequest);
        }
    }
});
```

Figura 34: Codul pentru crearea unui switch

2. Un buton ce permite editarea numelui şi a orei.

```
Button editButton = new Button(this);
params = new TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT);
editButton.setLayoutParams(params);
editButton.setText("Edit");
editButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent intent = new Intent(Schedule.this, EditSchedule.class);
        intent.putExtra("SCHEDULE_ID", mySchedule.getId());
        startActivity(intent);
    }
});
```

Figura 35: Codul pentru crearea butonului de edit

3. Un buton pentru a şterge semnalul. În momentul apăsării acestui buton este apelată metoda **deleteSchedule** pentru a şterge programarea din baza de date apoi este trimis un request pentru a şterge a o şterge şi de pe Raspberry Pi.

```
Button deleteButton = new Button(this);
params = new TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT);
deleteButton.setLayoutParams(params);
deleteButton.setText("Delete");
deleteButton.setOnClickListener((view) → {
        dbhandler.deleteSchedule(mySchedule.getId());
        RequestQueue queue = Volley.newRequestQueue(Schedule.this);
        String url ="http://192.168.0.140:5000/deleteautocode?code=" + mySchedule.getCode()
                + "&minute=" + mySchedule.getMinute() + "&hour=" + mySchedule.getHour();
        StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                new Response.Listener<String>() {...}, (error) → {
                Toast.makeText(Schedule.this, "Something went wrong!", Toast.LENGTH_SHORT).show();
        });
        queue.add(stringRequest);
        startActivity(new Intent(Schedule.this, Schedule.class));
});
```

Figura 36: Codul pentru crearea butonului de delete

3.7 Adăugarea unui semnal automat

Pentru a adaugă un semnal automat, identificat ca şi **schedule** în codul aplicatiei, trebuie apăsat butonul din dreapta jos de pe pagina descrisă la subcapitolul anterior. Acest lucru va duce la deschiderea unei pagini noi.

Pe această pagină se poate înregistra semnalul radio, numele semnalului şi ora la care va fi trimis semnalul. Când această pagină se deschide, aplicaţia trimite un request către Raspberry Pi pentru ca acesta să asculte şi să recepţioneze un semnal radio. La fel ca la adăugarea unui buton, pentru înregistrarea unui semnal este nevoie de a îndrepta telecomanda cât mai aproape de Raspberry Pi pentru ca acesta să fie capabil de a recepţiona semnalul. După ce semnalul a fost recepţionat el este trimis la aplicaţia android.

Figura 37:Adăugarea unui schedule

La apăsarea butonului de “ADD”, se verifică dacă semnalul radio a fost recepţionat şi câmpurile cu informaţiile necesare au fost completate. În cazul în care unul din câmpuri nu este completat sau conţine informaţii eronate, utilizatorul este informat printr-un mesaj care îi spune ce câmp nu este corect.

```
if (this.code == null){
    Toast.makeText(getApplicationContext(), "You need to press the button you want to record", Toast.LENGTH_SHORT).show();
} else{
    code = Integer.parseInt(this.code);
    if (scheduleName.isEmpty()){
        Toast.makeText(getApplicationContext(), "Provide a name", Toast.LENGTH_SHORT).show();
    } else if (this.hour.getText().toString().isEmpty()){
        Toast.makeText(getApplicationContext(), "Provide an hour", Toast.LENGTH_SHORT).show();
    } else {
        hour = Integer.parseInt(this.hour.getText().toString());
        if (hour > 23){
            Toast.makeText(getApplicationContext(), "Provide a valid hour", Toast.LENGTH_SHORT).show();
        } else if (this.minute.getText().toString().isEmpty()){
            Toast.makeText(getApplicationContext(), "Provide a minute", Toast.LENGTH_SHORT).show();
        } else{
            minute = Integer.parseInt(this.minute.getText().toString());
            if (minute > 59){
                Toast.makeText(getApplicationContext(), "Provide a valid minute", Toast.LENGTH_SHORT).show();
            } else{
                MySchedule schedule = new MySchedule(scheduleName, code, hour, minute);
                RequestQueue queue = Volley.newRequestQueue(this);
                String url ="http://192.168.0.140:5000/setautocode?code=" + code + "&minute=" + minute + "&hour=" + hour;
                StringRequest stringRequest = new StringRequest(Request.Method.POST, url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {
                            }
                        }, (error) -> {
                        Toast.makeText(getApplicationContext(), "Something went wrong!", Toast.LENGTH_SHORT).show();
                });
                queue.add(stringRequest);
                dbhandler.addSchedule(schedule);
                startActivity(new Intent(AddSchedule.this, Schedule.class));
            }
        }
    }
}
```

Figura 38: Codul pentru adăugarea unui schedule

În cazul în care totul este corect, un request este trimis către Raspberry Pi pentru a crea un crontab ce va trimite semnalul la ora specificată. Datele sunt de asemenea stocate în baza de date pentru ca utilizatorul să poată vedea ce semnale automate există şi pentru a le putea şterge sau edita.

3.8 Editarea unui semnal automat

Utilizatorul are posibilitatea de a modifica numele sau ora unui schedule. Pentru a face asta se apasă butonul “EDIT” de sub numele semnalului. Acesta va deschide o nouă pagină unde utilizatorul poate edita schedule-ul.

Figura 39: Pagina de editare a unui schedule

La deschiderea paginii, câmpurile ce pot fi editate sunt deja completate cu datele curente. Astfel dacă utilizatorul doreşte să modifice doar unul dintre câmpuri, de exemplu ora, el nu este nevoit să introducă şi numele din nou.

```
dbhandler.setScheduleNameAndTime(id, scheduleName, hour, minute);
RequestQueue queue = Volley.newRequestQueue(this);
String url2 ="http://192.168.0.140:5000/setautocode?code=" + schedule.getCode() + "&minute=" + minute + "&hour=" + hour;
StringRequest stringRequest2 = new StringRequest(Request.Method.POST, url2,
        new Response.Listener<String>() {...}, (error) → {
        Toast.makeText(getApplicationContext(), "Something went wrong!", Toast.LENGTH_SHORT).show();
});
queue.add(stringRequest2);
String url1 ="http://192.168.0.140:5000/deleteautocode?code=" + schedule.getCode()
        + "&minute=" + schedule.getMinute() + "&hour=" + schedule.getHour();
StringRequest stringRequest1 = new StringRequest(Request.Method.POST, url1,
        new Response.Listener<String>() {...}, (error) → {
        Toast.makeText(getApplicationContext(), "Something went wrong!", Toast.LENGTH_SHORT).show();
});
queue.add(stringRequest1);
startActivity(new Intent(EditSchedule.this, Schedule.class));
```

Figura 40: Codul pentru editarea unui schedule

La apăsarea butonul de “EDIT” se verifică ca nici un câmp să fie gol şi informaţiile introduse să nu fie eronate. Dacă totul este în regulă, este executat codul din Figura 40 care actualizează informaţiile din baza de date şi trimite doua request-uri către Raspberry Pi pentru a şterge vechiul crontab şi a adaugă unul nou. În final pagina Schedule va fi deschisă.

Concluziile Lucrării

1. Concluzii

Raspberry Pi-ul joacă un rol important în această aplicaţie, făcând posibilă comunicarea între aplicaţia android şi dispozitivele din jurul casei. Pentru a putea să-i conectez receptorul/transmiţătorul radio şi să îi controlez am fost nevoit să fac o aprofundare în acest domeniu.

Pentru aplicaţia android a fost nevoie de asemenea de o aprofundare, aceasta fiind prima mea aplicaţie dezvoltată pentru sistemul de operare android, dar a fost uşor de a începe să lucrez la aplicaţie datorită faptului că ea se bazează pe limbajul de programare Java, pe care l-am învăţat în timpul facultăţii.

Sunt de părere că această aplicaţie oferă o experienta mai plăcuta şi comodă utilizatorului în propria casă prin înlocuirea telecomenzilor cu telefonul. De asemenea mai este loc de îmbunatăţiri, câteva din idei fiind specificate în subcapitolul următor.

Consider că în cadrul acestei lucrări de licenţa am învăţat multe lucruri noi ce îmi vor fi de ajutor în viitor.

2. Aspecte rămase ce ar putea fi rezolvate în viitor

Una din limitările aplicaţiei este că Raspberry Pi-ul poate recepta şi transmite doar semnale radio, dar există şi telecomenzi ce folosesc infraroşu pentru a trimite comenzi la depărtare. Unul dintre lucrurile de făcut în viitor este să adaug suport şi pentru astfel de telecomenzi.

Un alt lucru ce ar putea aduce valoare aplicaţiei este implementarea de comenzi vocale. O soluţie pentru acest lucru ar putea fi crearea unui server propriu ce recunoaşte un set comenzi vocale. O altă soluţie ar fi instalarea unui asistent virtual cum ar fi Google Assistant şi găsita o metodă prin care la anumite comenzi vocale sa trimită semnale radio.

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