Competiția Premii Domenii POLITEHNICA Bucuresti, Sisteme de monitorizare si control al unor dispozitive IoT.

Studenți: Miron Andrei-Auraș, Buga Cătălin, Marcu Răzvan-Daniel.
Facultatea: Electronică Telecomunicații și Tehnologia Informației.
Grupa:434D.
Conducător Științific: S.I. Dr. Ing. Laurențiu BOICESCU.

Mai 2024.

Cuprinsul lucrării:

1.Rezumatul temei, prezentări generale.

În societatea modernă, evoluția tehnologică crescută din ultimii ani, a dus la creșterea cerințelor de control și monitorizare a dispozitivelor inteligente, controlul luminii casei dintr-o aplicație, a unei alarme, camere video, ecran, alt dispozitiv inteligent, toate acestea intrând în sfera IoT (Internet of Things), o ramură nouă a tehnologiei și ingineriei ce crește și prinde popularitate din ce în ce mai mult. Sfera IoT cuprinde elemente de programare Web, Embedded, Networking, Electronică și Criptologie, unind domenii inginerești de interes.
În cadrul acestei lucrări, am realizat un sistem de control și monitorizare a unei rețele de senzori, interconectate la un microcontroller, folosind o aplicație Android printr-o conexiune Wi-Fi la nivel local. Am folosit diferite tehnologii, de la programarea efectivă a aplicației Android în Java, folosind IDE-ul Android Studio la programarea firmware și software a unui microcontroller, specific un ESP32 WROOM, cu modul de Wi-Fi și BLE integrate in arhitectura acestuia. Am folosit diferiți senzori pentru testarea si implementarea prototipului, un senzor de temperatură si umiditate, un ecran LCD pentru a afișa diferite date, un led RGB, led-uri normale de diferite culori și un buzzer. În cadrul proiectului există 2 opțiuni de utilizare a aplicației, de control și monitorizare, acestea fiind discutate și dezvoltate în cadrul lucrării prezentate. De asemenea, interfața pentru aplicația Android trebuie să fie cât mai intuitivă și ușor de folosit de către orice utilizator.

Exemplu de utilizare:
Utilizatorul va citi de pe afișajul LCD conectat la microcontroler ip-ul acestuia și îl va introduce în prima interfață a aplicației. Dacă ip-ul este corect (se va transmite un mesaj TCP către server și un răspuns aferent asteptat), utilizatorul va fi mutat în următoarea interfață/activitate (conexiunea reușită), unde va trebui să aleagă dintre a monitoriza sau controla microcontrolerul (în cazul unei conexiuni eșuate există o altă interfață în care utilizatorul va trebui să reintroducă ip-ul). În modul de monitorizare, utilizatorul va putea printr-un buton de refresh să vadă datele transmise de către server (temperatura, umiditate, starea ledurilor-aprins/stins), iar în funcție de temperatura din cameră se va activa o alarmă de atenționare (led roșu separat, nu cel RGB alături de un advertisment sonor). În meniul de control există diferite opțiuni, aprinderea/stingerea unui led RGB, controlul unui buzzer și afișarea unui anumit mesaj pe ecran.

2.Obiective

Obiectivele propuse în realizarea lucrării sunt reprezentate de implementarea a celor două moduri de funcționare în mod eficient, anume monitorizarea și controlul. Un principal obiectiv ar fi cel de monitorizare, de culegere corectă a datelor primite de la server, să nu existe erori de propagare a semnalului(zgomot) și ca datele citite de senzori să fie cât mai exacte. Să existe un mod de alarmă în cazul în care temperatura depășește un anumit prag, și să putem obține datele într-un mod cât mai ușor și eficient, chiar în cazul în care există mai mulți utilizatori care transmit aceeași cerere. Controlul de asemenea trebuie să aibă loc cât mai repede și eficient, chiar dacă există un număr mai mare de utilizatori ce doresc să controleze dispozitivul în același moment de timp (găsirea unei metode pentru posibila utilizare a unei familii/corporații/mulțime).

3.Schema bloc aplicație-dispozitiv IoT

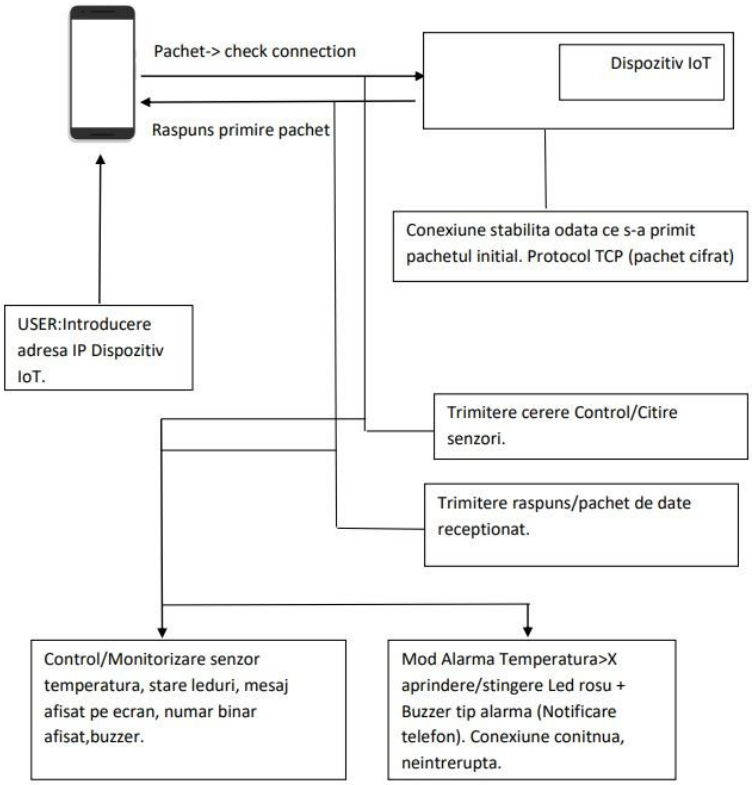


Figura1

4.Tehnologii folosite

4.1 Tehnologii Java-POO

În cadrul dezvoltării software a aplicației Android, am folosit IDE-ul Android Studio, ce oferă numeroase instrumente și tehnologii pentru dezvoltarea aplicațiilor specifice acestui sistem de operare mobil. Limbajul de programare folosit a fost Java, acesta folosește paradigma de programare orientată pe obiecte, fiind foarte util în realizarea funcționalităților interfețelor. Pentru realizarea design-ului interfețelor am utilizat editorul intern ce folosește fișiere XML pentru controlul și stilizarea acestora, încercând să obținem interfețe cât mai simple și intuitive.

4.2 Tehnologii Arduino/Embedded&IoT

Microcontrolerul ESP32 a jucat un rol crucial în relizarea proiectului. Fiind foarte ieftin și versatil, integrând un modul Wi-Fi și mulți pini pentru conectivitate hardware, cât și o arhitectură dual core, alături de posibilitatea implementării Software în Arduino IDE, am reușit să integrăm toate funcționalitățile firmware, software și hardware necesare. Limbajul de programare utilizat este C++, unde am folosit diferite librării precum Wifi.h pentru implementarea conexiunii server-client dintre microcontroler(server) și aplicația Java (clientul).

4.3 Networking-Protocolul TCP

Un alt element important în cadrul lucrării a fost alegerea unui protocol de comunicație dintre cele 2 dispozitive. Am optat pentru o conexiune la nivel TCP (Transfer Control Protocol) deoarece asigură o conexiune sigură, știind cu certitudine că pentru fiecare pachet de date transmis există un răspuns. Protocolul a fost implementat manual, folosind ca bază doar librării și API-uri ce doar creează serverul și trimit/așteaptă pachete de date, arhitectura protocolului fiind realizată în partea de Software și detaliată în secțiunea de interconectivitate. Pe scurt pe ESP32 se creează un server la nivel local unde se așteaptă conexiune cu un client într-un loop infinit, iar odată ce clientul s-a conectat se va citi ce mesaj transmite acesta, fiind de tipul control sau de monitorizare, întotdeauna trimițând un mesaj de confirmare primirii pachetului de date.

4.4 Criptologie-Criptarea datelor

Este foarte importantă implementarea unui algoritm de cifrare/codare a informației. În cadrul proiectului, nu am folosit funcții predefinite ci am creat propriul algoritm de criptare. Am folosit un simplu algoritm de cifrare a informației prin permutări simple de tip Caesar, anume am luat un alfabet comun pentru cele 2 dispozitive în funcție de care se va face criptarea și decriptarea, folosind o cheie comună ce se schimbă în funcție de anumiți parametrii (cheia este (modulo 27)-1, reprezentând pașii de parcurgere a alfabetului de la cuvântul respectiv până la cel cu care va fi înlocuit).

5.Proiectarea

La nivel de proiectare, am împărțit dezvoltarea aplicației în subcategorii. Am început de la dezvoltarea aplicației Android, unde implementarea se împarte în partea de design și partea de funcționalitate, partea Embedded care se împarte în hardware/firmware și software și separat partea de interconectivitate dintre server și client (aplicația Android și microcontroler).

5.1 Proiectarea Aplicației Android-Java

Organigrama aplicației Android:

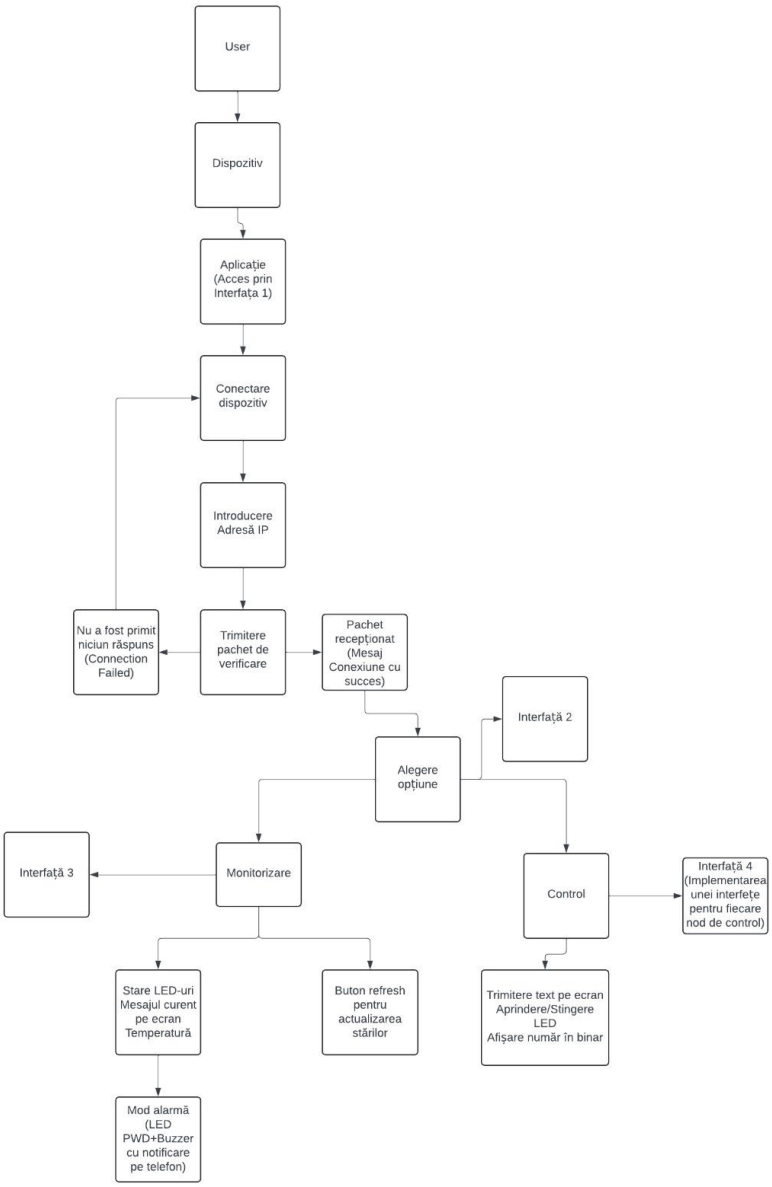


Figura2

5.1.1 Designul aplicației

Design-ul aplicației a fost realizat prin folosirea limbajului de marcare XML (eXtensible Markup Language) , care permite specificarea structurii și a atributelor elementelor UI. Un fișier de layout XML pentru Android are o structură ierarhică. Elementul rădăcină este de obicei un LinearLayout, RelativeLayout, ConstraintLayout, sau alt layout container, iar în interiorul acestuia sunt incluse alte elemente, cum ar fi butoane, text, imagini sau alte containere. Din meniul "Palette" putem alege diferite elemente de adăugat care vor realiza interacțiunea cu utilizatorul. Fiecare element adăugat are un anumit id, pe care îl putem folosi mai apoi în partea de implementare Java pentru a identifica obiectul respectiv și să extragem date sau să modificăm butoane, texte, imagini. De asemenea în secțiunea Attributes regăsim numeroase atribute ce pot fi implementate. De exemplu, odată ce utilizatorul a apăsat pe un buton putem folosi o metodă specifică "OnClick" prin suprascriere (sau să denumim altfel metoda) pentru a realiza anumite funcționalități (deschiderea unei noi activități, preluarea datelor dintr-o căsuță de text etc). Fiecare element pus pe interfață trebuie legat spațial prin niște "Constraints", astfel încât pe orice dispozitiv, indiferent de rezoluția acestuia, să se respecte poziționarea grafică.

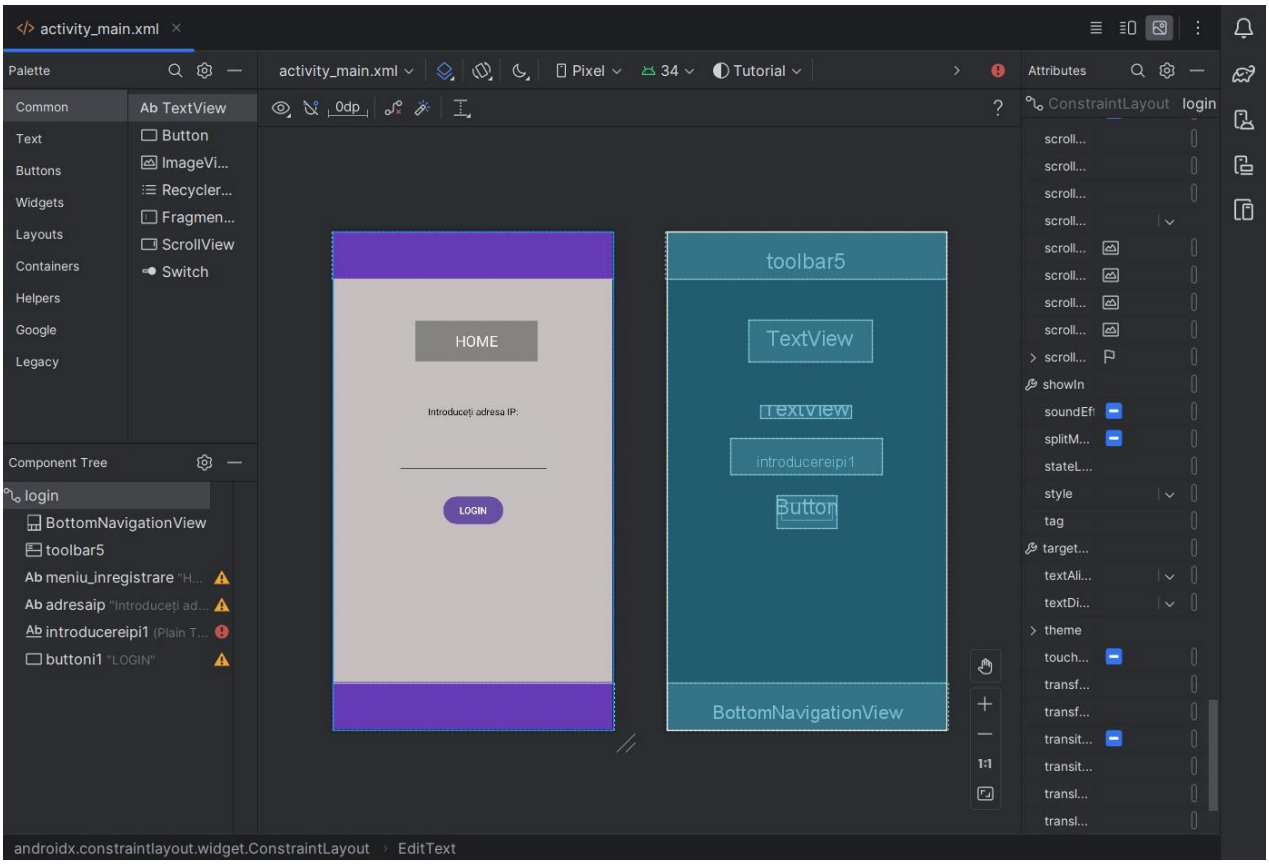


Figura3

În general, interfețele XML sunt un element fundamental al dezvoltării aplicațiilor Android în Android Studio, permițând dezvoltatorilor să definească aspectul și comportamentul interfeței utilizator grafice într-un mod flexibil și ușor de gestionat. În proiectul nostru, am realizat interfețe care includ: meniul pentru introducerea adresei IP, câte o interfață pentru fiecare tip de conexiune (reușită sau eșuată), interfața de monitorizare (putem observa datele oferite de server) și interfața de control (conține diferite butoane pentru fiecare funcționalitate) alături de fiecare interfață pentru modul de control selectat. Fiecare interfață activitate vine însoțită de o clasă în limbajul Java ce servește pentru funcționalitatea acesteia, practic cele 2 sunt interconectate și se numesc activități. De asemenea, în ceea ce privește procesul de depanare, am simulat aplicația folosind simulator Android integrat în IDE, am pornit de la modelul unui telefon Pixel 3a API 29 pentru că ne permite să integrăm aplicația pentru modele Android asemănătoare acestuia. Design-ul este minimalist, implementat pentru a simplifica folosirea aplicației, accentul fiind pus pe funcționalitatea software.

5.2 Realizarea interconectivității dintre aplicația Android și Dispozitivul IoT

Realizarea conexiunii dintre dispozitivul IoT (Microcontrolerul ESP32) și aplicația Android, se va realiza prin intermediul unui protocol TCP. Pe scurt, se va realiza o conexiune de tipul server client, unde serverul este microcontrolerul iar clientul este userul ce folosește aplicația. Conexiunea este una simplă, pentru fiecare cerere de conexiune există un răspuns, dacă răspunsul nu a fost primit atunci conexiunea nu se va putea stabili (Pentru fiecare pachet de date transmis există un răspuns primit). Am ales acest tip de conexiune întrucât dorim să asigurăm user-ului siguranța de transmitere a datelor, acesta să știe dacă cererea lui de monitorizare sau de control a fost trimisă cu succes. Protocolul descris a fost emulat manual de către noi, apelând doar la bibliotecile simple ce realizează crearea unui server, aceasta realizându-se la nivel local, ambele dispozitive trebuind să se afle în aceeași rețea locala de Wi-Fi.

5.2.1 Aplicația Android

În cadrul aplicației Android, am declarat 3 variabile pentru realizarea conexiunii în clasa principală MainActivity (Home)-Figura4.

```
9 usages
public static String ip;
2 usages
String Mesaj_conexiune;
9 usages
public static int port;
```

Figura4

Variabilele sunt declarate de tip static și public deoarece vor trebui refolosite ulterior în cadrul altor activități(clase), pentru a realiza conexiunea cu serverul. Portul îl vom declara ca fiind 80 implicit, deoarece portul 80 este cel standard pentru conexiunile la nivel local, Mesaj_conexiune este variabila cu care vom trimite mesajul către microcontroler. De asemenea mesajul este declarat standard, în cazul acesta nu de către utilizator, utilizatorul doar dând ca input adresa ip, mesajul implicit trimis către server fiind un „1”. De asemenea în variabila ip se va salva ip-ul introdus de către utilizator.

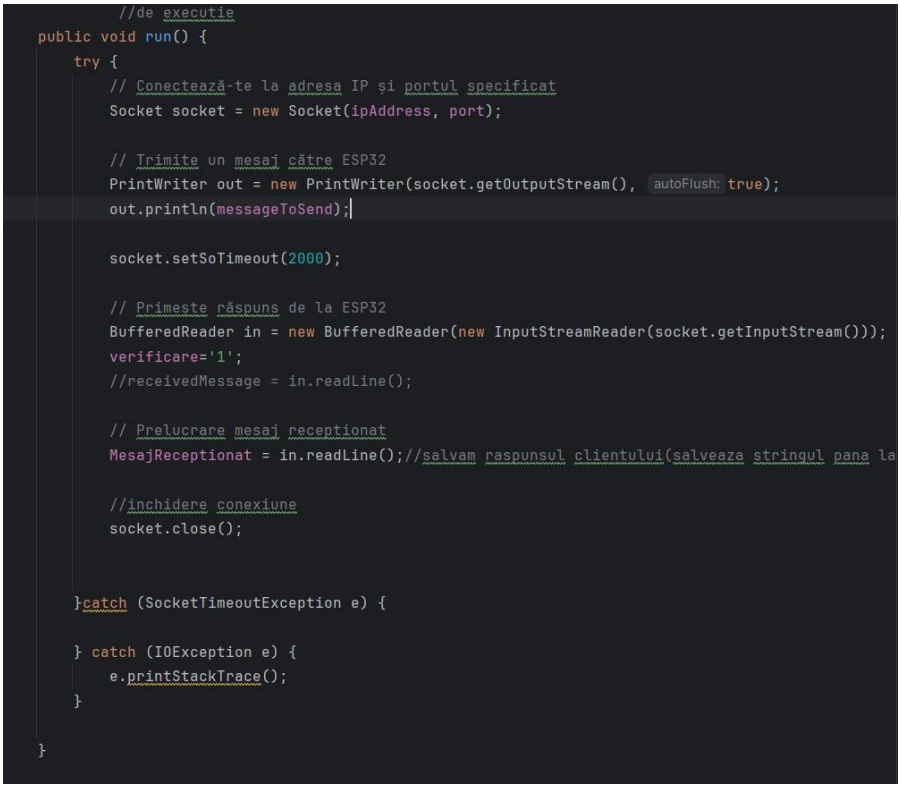
figura5

Pentru crearea serverului și a sesiunii TCP, am folosit un Thread (fir de execuție) pentru a paraleliza procesele. Acest lucru este esențial în eficientizarea aplicației și în evitarea crash-urilor, firul de execuție nu afectează procesul principal în care se alfă userul, prin urmare nu vor exista întreruperi în proces, sau încetiniri. Pentru realizarea firului de execuție am creat o nouă clasă denumită ConnectionThread. În cadrul clasei ConnectionThread, am folosit librăriile: .io.BufferedReader, .io.IOException, .io.InputStreamReader, .io.PrinWriter, .net.Socket și .net.SocketTimeoutException.Initial am declarat un constructor (figura 6), care ne va ajuta să pasăm parametrii firului de execuție creat pentru a realiza conexiune cu datele date de către user.

```
8 usages
public ConnectionThread(String ipAddress, int port, String messageToSend) {
    this.ipAddress = ipAddress;
    this.port = port;
    this.messageToSend = messageToSend;

}
```

Figura6

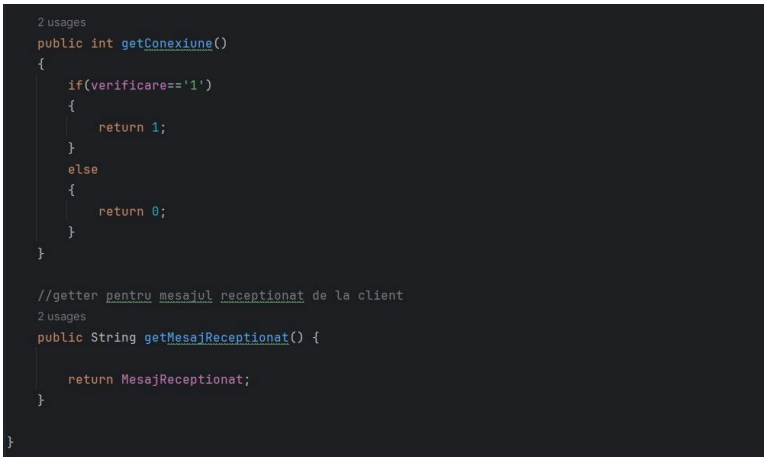
Un detaliu foarte important este că clasa aceasta moștenește clasa de bază denumită Thread , prin urmare vom putea folosi metodele specifice clasei Thread în crearea și funcționarea firelor de execuție.



```
        //de executie
public void run() {
    try {
        // Conectează-te la adresa IP și portul specificat
        Socket socket = new Socket(ipAddress, port);

        // Trimite un mesaj către ESP32
        PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
        out.println(messageToSend);

        socket.setSoTimeout(2000);

        // Primește răspuns de la ESP32
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        verificare='1';
        //receivedMessage = in.readLine();

        // Prelucrare mesaj receptionat
        MesajReceptionat = in.readLine();//salvam raspunsul clientului(salveaza stringul pana la

        //inchidere conexiune
        socket.close();


    }catch (SocketTimeoutException e) {

    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Figura7

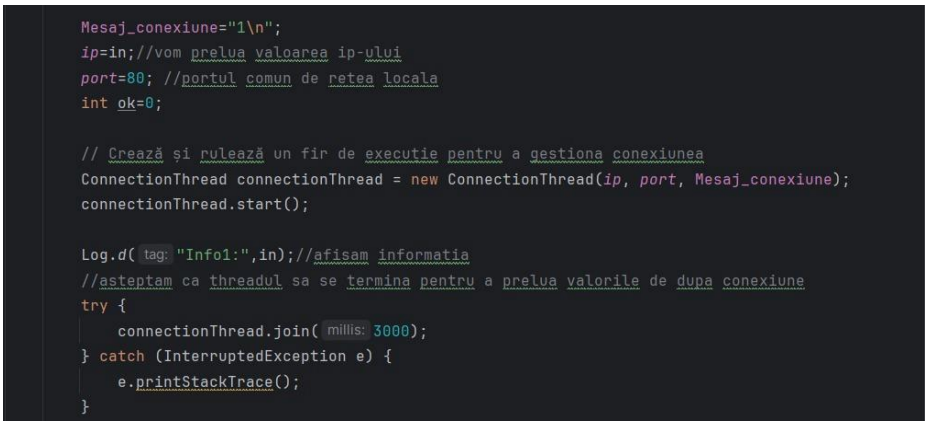
Odată declarat constructorul, am creat funcția run() (figura7) din cadrul funcției de bază, prin urmare apelând la noțiunea de polimorfism am dat override la aceasta pentru a o utiliza sub același nume. Try și Catch este o cale ușoară de a ne asigura că în cazul în care există vreo eroare de conexiune aceasta să fie capturată și afișată. Inițial declarăm un nou socket de conexiune, specificând portul și ip-ul destinatarului, ca apoi să trimită mesajul folosind variabila out de tipul de dată PrintWriter prin metoda out.println. În cazul în care nu se realizează conexiunea, ne vom folosi de metoda socket.setSoTimeout() pentru a afișa excepția catchTimeoutException. În cazul în care conexiunea s-a realizat, vom citi folosind variabila in de tipul BuffReader mesajul recepționat (aceasta citește până întâlnește caracterul \n deci este important ca fiecare mesaj transmis de server înapoi spre client să se termine în new line). În cazul în care avem o conexiune stabilită, vom folosi o variabilă denumită Verificare de tip char care se va face 1 dacă conexiunea a avut loc și în caz contrar rămâne 0 ca la declarare. De asemenea în variabila MesajReceptionat vom salva răspunsul primit.



```
2 usages
public int getConexiune()
{
    if(verificare=='1')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

//getter pentru mesajul receptionat de la client
2 usages
public String getMesajReceptionat() {

    return MesajReceptionat;
}

}
```

Figura8

În cele din urmă am realizat în cadrul clasei ConnectionThread 2 Getters(figura 8), unul pentru verificarea conexiunii iar celălalt pentru obținerea răspunsului transmis de către server. Revenind la funcția principală, vom crea o nouă variabilă denumită connectionThread de tipul de dată ConnectionThread(figura 9), folosind constructorul definit anterior pasând parametrii de conexiune (ip-ul, portul și mesajul de transmis).



```
Mesaj_conexiune="1\n";
ip=in;//vom prelua valoarea ip-ului
port=80; //portul comun de retea locala
int ok=0;

// Crează și rulează un fir de execuție pentru a gestiona conexiunea
ConnectionThread connectionThread = new ConnectionThread(ip, port, Mesaj_conexiune);
connectionThread.start();

Log.d( tag: "Info1:",in);//afisam informatia
//asteptam ca threadul sa se termina pentru a prelua valorile de dupa conexiune
try {
    connectionThread.join( millis: 3000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Figura9

Metoda .start() în mod implicit apelează metoda run() din cadrul Thread-ului, practic dă startul paralelizării proceselor (am folosit logcat-ul anume Log.d pentru afișarea unor mesaje de test pentru a verifica corectitudinea executării unor secvențe de cod). Foarte important este try-catch-ul folosit în cele ce urmează deoarece în cazul în care nu a avut loc conexiunea, după un timp de 3000ms să se încheie firul de execuție și să se treacă mai departe (în cadrul clasei Thread când citim inputul, aceasta va citi la nesfârșit până nu întâlnește un \n, prin urmare va trebui să dăm un timp maxim de conexiune și așteptare primire mesaj, în cazul în care conexiunea a avut loc toată procedura durează câteva milisecunde deci un timp de 3000ms este suficient pentru a aștepta terminarea conexiunii și implicit primirea răspunsului, lucru testat ulterior și confirmat). În funcție de mesajul primit la recepție de la server, utilizatorul va fi redirecționat către acitivtatea/interfața de conexiune reușită sau interfața de conexiune eșuată.

5.2.2 Dispozitivul IoT

În ceea ce privește setarea conexiunii pentru ESP32, am utilizat biblioteca standard wifi.h, în funcția de set-up realizând astfel conexiunea efectivă dintre microcontroler și rețeaua local de Wi-Fi. Există două funcții principale, cea de set-up și cea de loop care se apelează la un interval regulat cu o frecvență foarte mare.

```
//------------------------------------------------------------SET-up Wifi-----
Serial.begin(115200); // Inițializează comunicarea serială

// Conectează Arduino la rețeaua WiFi
Serial.print("Conectare la rețeaua WiFi");
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
}
Serial.println("");
Serial.println("Conectat la rețeaua WiFi cu succes");
ok=1;

server.begin(); // Pornirea serverului
Serial.println("Server pornit cu adresa ip:");
Serial.println(WiFi.localIP());
```

Figura10

În funcția principală loop() (figura 10) se așteaptă conexiunea cu clientul, în cazul în care clientul este disponibil (adică s-a conectat la adresa ip și portul dispozitivului), se va afișa în seriala mesajul „Client conectat” iar cât timp acesta este conectat se va citi mesajul trimis până la întâlnirea caracterului \n cu variabila String command. Pentru prima conexiune de ipcheck, mesajul transmis este 1 (figura 11), caz în care se va afișa un mesaj în consola de recepționare a acestuia iar mai departe către client se va trimite un 1 ca validare a mesajului către client. (Partea de închidere era folosită anterior ca prototip în încercare de conexiune, momentan nu are vreo utilitate). La finalizarea codului vom deconecta clientul afișând un mesaj de verificare.

```
WiFiClient client = server.available(); // asteapta conexiuni de

if (client) { // Dacă s-a conectat un client
  Serial.println("Client conectat");

  while (client.connected()) {
    if (client.available()) {
      String command = client.readStringUntil('\n'); // citeste c

      //Check-Connection din aplicatie
      if (command == "1") {

        Serial.println("Primire pachet date receptionate");
        client.println("1\n");//folosim \n deoarece in Java socke
      } else if (command == "0") {

        Serial.println("Inchidere receptionat");
        client.println("0\n");
      }
```

Figura11

Important de mentionat că s-a ținut cont de cazul în care rețeaua la care este conectat dispozitivul IoT a căzut, astfel se verifică întotdeauna la fiecare loop starea conexiunii, în cazul în care rețeaua nu mai este vizibilă, programul va intra într-un loop infinit în care se așteaptă reconectarea și prin urmare reinițializarea conexiunii la rețea (figura 12).

```
while(WiFi.status() != WL_CONNECTED) //daca reteaua wifi locala a cazut, asteapta pana cand revine
{
  delay(1000);
  Serial.print(".");

  if(ok1==0)
  {
    tft.fillScreen(ST77XX_BLACK);
    tft.setCursor(0,0);
    tft.print("Neconectat...");
    ok1=1;
    ok=0;
  }

  ReconnectToWiFi();

}
ok1=0;

if(ok==0) //In cazul in care ne reconecram vom rescrie pe ecran datele de baza
{
  initializare_ecran(temp,humi,IpEcran);
  ok=1;
}
```

Figura12

5.3 Funcționalități. Implementare Aplicație/Microcontroler.

5.3.1 Control

În cadrul aplicației există diferite opțiuni de control, printre care se numără control led RGB, control Buzzer și control ecran. Va urma o prezentare a legăturilor hardware efectuate pentru a interconecta senzorii.
Pentru LED-ul RGB am folosit pinii 15, 2 si 5, alături de rezistențe de 220 Ohm pentru a limita curentul prin acesta. De asemenea, pentru Buzzer am folosit pinul 13.

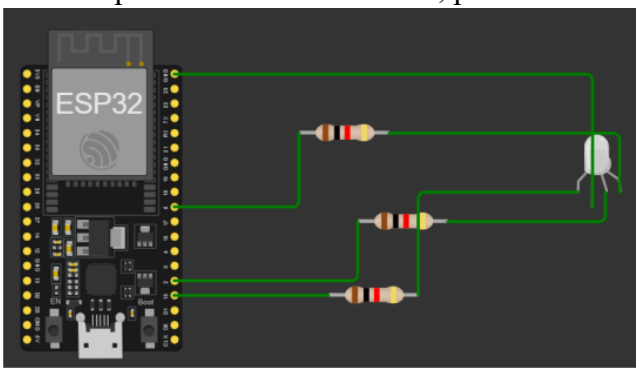


Figura12

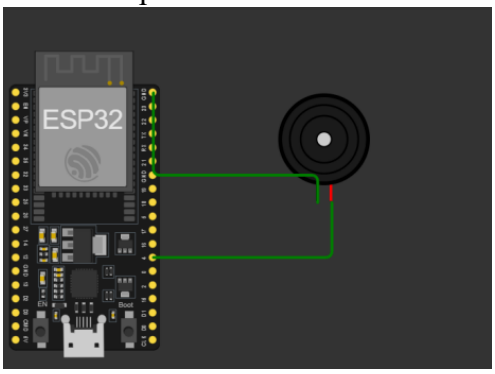


Figura13

Am folosit un senzor de temperatură DHT și un ecran LCD TFT cu driverul ST7735, cu o conexiune SPI pentru afișajul grafic. De asemenea am mai adăugat un led roșu pentru alarmă (modalitatea alarmă).

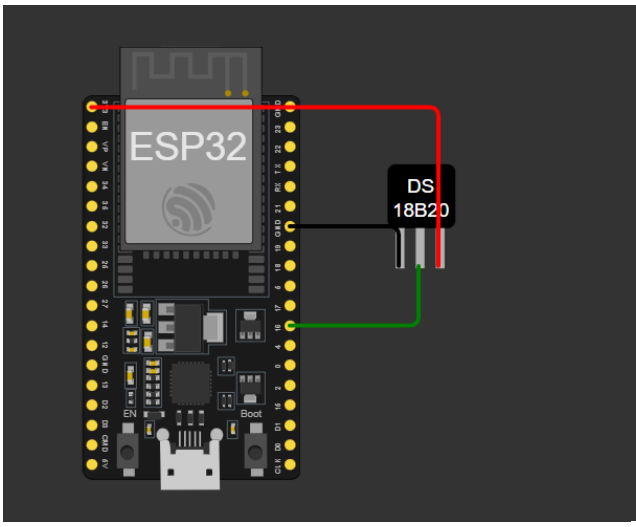


Figura14

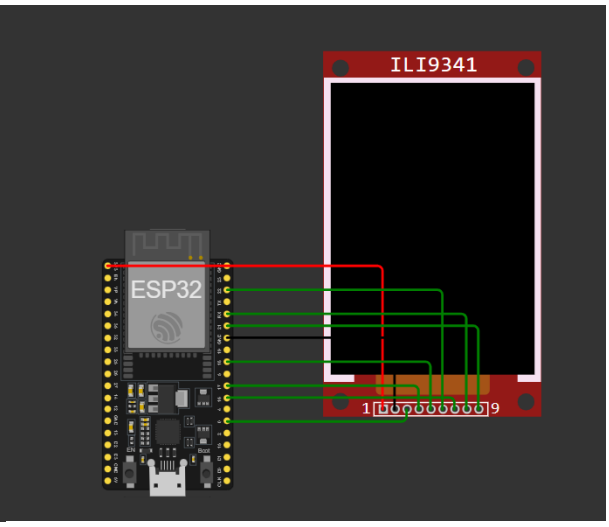


Figura15

Principiul este simplu. Odată ce userul a apăsat pe un buton (de exemplu pentru a aprinde LED-ul RGB sau a-l stinge, pe butonul “Trimite” din interfața control ecran), se transmite un mesaj către server (ESP32) care în funcția de comandă recepționată va realiza funcționalitatea respectivă. În codul Arduino există o variabilă numită “command” care preia mesajul transmis de către client.

Aprinderea LED-ului RGB a fost realizată prin principiul automatei (proces secvențial), în care fiecărei posibile culori i s-a atribuit o stare (avem maxim 4 stări, pentru culorile roșu, verde, albastru și alb). La fiecare apăsare de buton pentru a aprinde LED-ul, din aplicație, se va apela o funcție în cadrul microcontroller-ului unde un contor ce reprezintă starea curentă va crește modulo 4 (vom avea practic stările 0,1,2 si 3) fiecare stare fiind descrisă de o anumită culoare, adică se setează pinii corespunzători HIGH sau LOW.

Ecranul a fost programat și inițializat folosind librăria grafică specifică driverelor ST7735, folosind funcții specifice de manipulare a pixelilor am reușit să afișez diferite informații pe acesta, cel mai important fiind IP-ul dispozitivului IoT pentru ca userul să se poată conecta cât și un mesaj de neconectare în cazul în care rețeaua Wi-fi a căzut. Am folosit pinii 25 pentru CS (Chip Selected), 14 pentru RST(Reset), 12 pentru RS (Register Select), 23 pentru SDA (Serial Data) și 18 pentru SCL (Serial Clock).

Este important de menționat că odată ce clientul s-a conectat la server și a primit răspunsul de la acesta, conexiunea va fi încheiată și viceversa din partea serverului, așteptând prin urmare conexiunea cu un nou client. Deoarece microcontrolerul are resurse limitate și nu permite mulți threading-ul, astfel încât să paralelizăm procesele și să existe mai mulți utilizatori conectați în același timp, am proiectat din punct de vedere Software o metodă simplă prin care mai mulți utilizatori să poată accesa în același timp dispozitivul prin a încheia sesiunea de conectare după fiecare primire și recepționare a pachetului.

5.3.2 Monitorizare

Pentru modalitatea de monitorizare, la nivel de aplicație, am creat o interfață în care odată ce apăsăm pe butonul de actualizare se va transmite către server o cerere. Serverul odată ce a primit cererea specifică monitorizării, anume CerereMonitorizare, microcontrolerul va transmite clientului un string care conține toate datele necesare în formatul următor: StareLed ProcentUmiditate Temperatura Alarma ca un exemplu: "Aprins/Stins 34.00% 25.30 AlarmaOn/AlarmaOff". La nivel de aplicație, vom folosi getter-ul specific pentru a prelua mesajul transmis de către server, urmând ca într-o funcție specifică să separăm în variabile separate șirul de date transmis de către server iar în funcție de parametrii primiți îi vom afișa, de altfel pornind în cazul în care temperatura depășește un anumit prag (am ales în cazul aplicației 24 de grade), o alarmă sonoră și vizuală (LED roșu și Buzzer).

```
//-----------------------------------------------------------------

if(command == "Monitorizare") //ne vom crea un String in care ne
{                             //StareLed ProcentUmiditate Tempera

  MesajMonitorizare="";//resetam Stringul de salvare a mesajului

  if(stins==0)
  {
    MesajMonitorizare += "Stins ";
  }

  if(stins==1)
  {
    MesajMonitorizare += "Aprins ";
  }

  MesajMonitorizare += String(humi);
  MesajMonitorizare += " " + String(temp);

  if(temp>24) //pentru modul de alarma ledul rosu
  {
    digitalWrite(13,HIGH);
    CheckAlarma=1;
    tone(4,1000);
  }
```

Figura16

```
1 usage
private void PrelucreazaMesajReceptionat()
{
    vector=DataRec.split( regex: " ");
    DataLed=vector[0];
    DataHumi=vector[1];
    DataTemp=vector[2];
    DataAlarma=vector[3];
    Log.d( tag: "MesajReceptionat:",DataLed);
    Log.d( tag: "MesajReceptionat:",DataHumi);
    Log.d( tag: "MesajReceptionat:",DataTemp);
    Log.d( tag: "MesajReceptionat:",DataAlarma);

}
}
```

Figura17

De asemenea, temperatura și umiditatea alături de IP vor fi afișate pe ecranul LCD. În mod implicit în cadrul aplicației, butonul pentru a dezactiva alarma este oprit, doar în cazul în care aceasta se activează, butonul va deveni disponibil pentru a fi stins, în rest va rămâne inactiv.

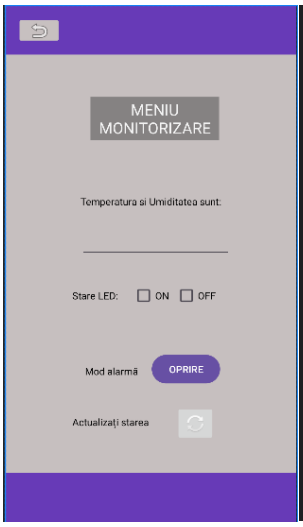


Figura18

```
1 usage
public void DezactivareAlarma(View v) //odata ce apasam pe butonul de alarma se va dezactiva
{
    //vom trimite un mesaj catre server ca sa inchidem alarma (ledul si buzzerul)
    Button ButonAlarma = findViewById(R.id.buttonOpririei4);
    ok=0;
    String ip=MainActivity.ip;
    int port=MainActivity.port;
    DataSent="StingeAlarma";
    ButonAlarma.setEnabled(false);
    ConnectionThread connectionThread = new ConnectionThread(ip, port, DataSent);
    connectionThread.start();

    try {
        connectionThread.join( millis: 3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}
```

Figura19

6.Rezultate obținute și Concluzii

Aplicația este funcțională din punct de vedere software, firmware cât și hardware. Montajul a fost efectuat pe o plăcuță de test de tip Breadboard pentru a ușura depanările.

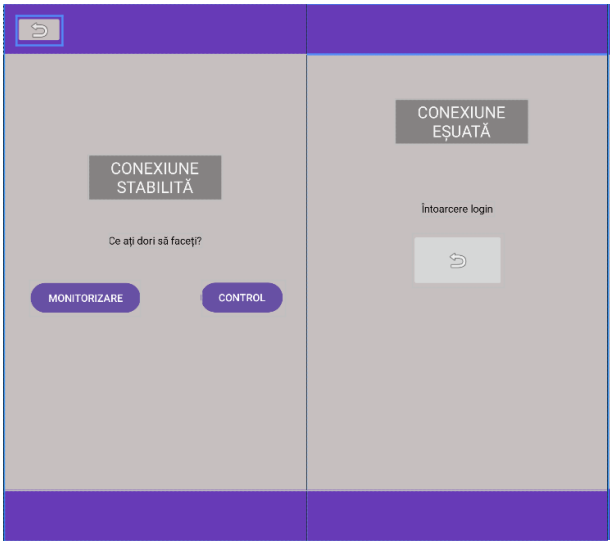


Figura20

Figura21

Figura22

Figura23

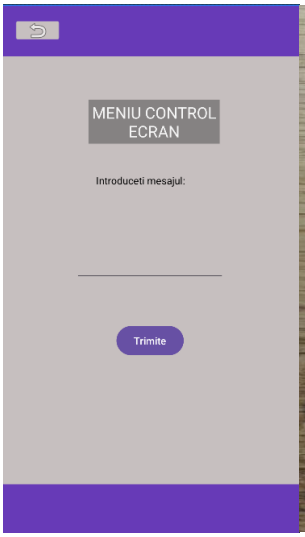


Figura24

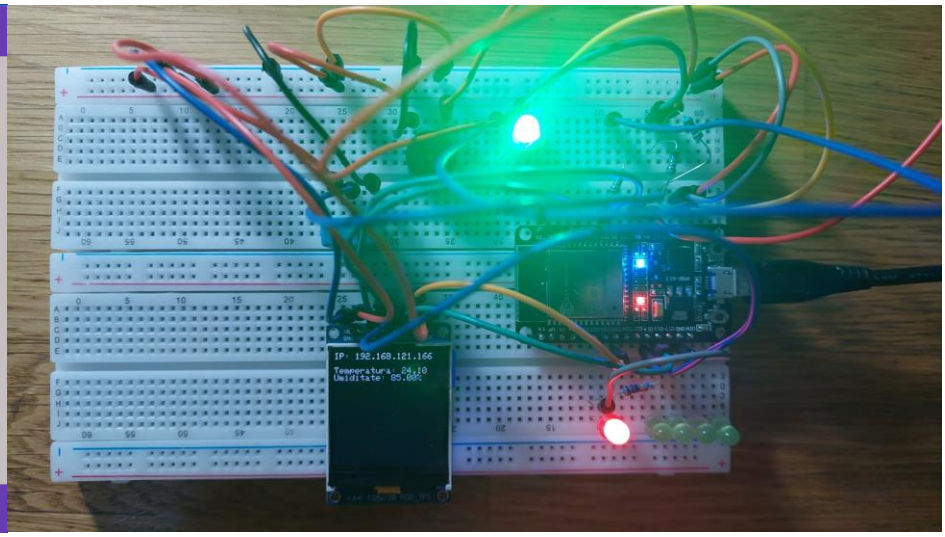


Figura25

Link prezentare video aplicație: https://youtu.be/8axI6xfS8J0

7.Bibliografie

Curs/Laborator TPI-Tehnologii de programare in internet UNSTPB ETTI,
Laborator APC-Arhitecturi si protocoale de comunicatii UNSTPB ETTI