### Facultatea de Automatică și Calculatoare

### Programul de licență: Calculatoare și Tehnologia Informației

**SISTEM DE IRIGARE AUTOMAT**

**CONTROLAT CU AJUTORUL UNEI APLICAȚII MOBILE**

Proiect de Diplomă

Absolvent: Nicolae-Cosmin POPA

Coordonator științific:

Prof. Dr. Habil. Ing. Marius MARCU

## Timișoara 2020

Cuprins

1. [Introducere 4](#_bookmark0)
   1. [Internet of Things (IoT) 4](#_bookmark1)
   2. [Descrierea lucrării și prezentarea problemei 6](#_bookmark2)
2. [Aplicații și sisteme similare existente 7](#_bookmark3)
   1. [Controllere Rain Bird 7](#_bookmark4)
      1. [Controllerul Rain Bird ESP-RZX4i 7](#_bookmark5)
      2. [Controllerul Rain Bird ESP-RZX8 8](#_bookmark6)
      3. [Rain Bird 8](#_bookmark7)
   2. [Controllere Hunter 9](#_bookmark8)
      1. [Controller tip X-Core 9](#_bookmark9)
      2. [Controller Hunter X2 9](#_bookmark10)
      3. [Hydrawise Irrigation 10](#_bookmark11)
   3. [ii.ri Smart Irrigation Control 11](#_bookmark12)
   4. [MySOLEM 11](#_bookmark13)
3. [Concepte teoretice 12](#_bookmark14)
   1. [Arduino IDE 12](#_bookmark15)
   2. [Android Studio 12](#_bookmark16)
   3. [Limbaje de programare 12](#_bookmark17)
   4. [Sistemul și componentele sistemului 13](#_bookmark18)
   5. [Programarea/dezvoltarea aplicațiilor mobile pentru Android 15](#_bookmark19)
4. [Implementarea soluției sistemului 17](#_bookmark20)
   1. [Schema logică a sistemului 17](#_bookmark21)
   2. [Programarea modulului NodeMCU ESP8266 18](#_bookmark22)
   3. [Aplicația mobilă 20](#_bookmark23)
5. [Concluzii 42](#_bookmark24)

[Bibliografie 43](#_bookmark25)

# Introducere

* 1. **Internet of Things (IoT)**

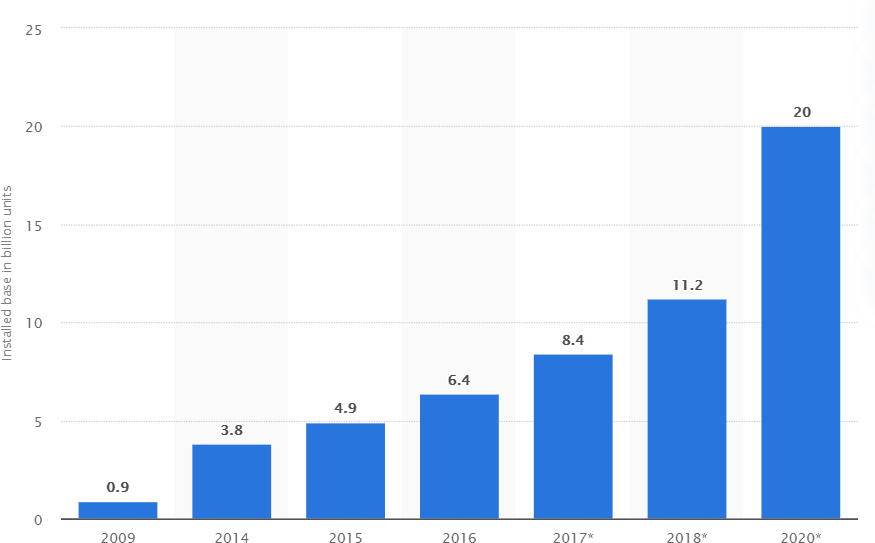
„Internet of Things (IoT) este un concept și o paradigmă care consideră prezența omniprezentă în mediul înconjurător a unei varietăți de lucruri / obiecte care prin conexiuni wireless și cu fir și scheme de adresare unice sunt capabile să interacționeze între ele și să coopereze cu alte lucruri / obiecte pentru a crea noi aplicații / servicii și a atinge obiective comune.”1

Provocările pentru dezvoltarea și cercetarea acestui domeniu în vederea creării unei lumi inteligente sunt foarte mari. O lume în care toate domeniile de activitate a vieții se îndreaptă spre dezvoltarea de medii inteligente.

Scopul Internet of Things este de a interconecta lucrurile / obiectele în orice moment și în orice loc folosindu-se de diverse căi sau rețele.

Obiectele, odată interconectate, pot lua decizii de unele singure pe baza informațiilor primate de la celelalte obiecte și pot da informații celorlalte obiecte cu care sunt interconectate.

Acest domeniu cunoaște o dezvoltare foarte rapidă în zilele noastre. Statisticile arată faptul că în ultimii ani, acest domeniu are o creștere exponențială. Odată cu dezvoltarea domeniului, crește și numărul dispozitivelor interconectate după cum se poate vedea și în următorul grafic. Nick Jones, vicepreședinte pe partea de cercetare la Gartner (companie americană de consultanță și cercetare IT, fondată în 1979) spunea că până în anul 2021 vor fi aproxitativ 25 de miliarde de dispozitive interconectate.



*Fig. 1.1 Numărul de dispozitive IoT utilizate în întreaga lume între 2009 și 2020*

*[*[*https://www.statista.com/statistics/764026/number-of-iot-devices-in-use-worldwide/*](https://www.statista.com/statistics/764026/number-of-iot-devices-in-use-worldwide/)*]*

Numărul dispozitivelor conectate și dezvoltarea rapidă și în creștere se datorează faptului că acest domeniu își desfășoară activitatea în fiecare capitol al vieții noastre. Lucrul acesta se poate observa prin faptul că oamenii își schimbă modul de viață adaptându-se după tehnologie.

Pentru a vedea modul în care se schimbă rutina noastră zilnică, iată câteva exemple despre influența pe care IoT o are asupra diferitelor industrii:

* + - Case inteligente: casele inteligente sunt tot mai prezente. Se pot ajusta setările de iluminare, aer condiționat, încuietori de uși și multe altele cu câteva clicuri de pe un dispozitiv inteligent.
    - Lucruri în mișcare: Senzorii și software-ul colectează date în mod constant, iar acestea se prelucrează pentru a obține informații despre cerințele de fitness, sănătate și divertisment.
    - Mașini inteligente: În zilele noastre, vehiculele sunt optimizate pentru a oferi funcționalități importante care facilitează viața șoferului. De la citirea e-mailurilor prin sistemul de sunet la statistici și metrici inteligente despre performanța mașinii, IoT oferă numeroase aplicații pentru industria auto.
    - Agricultură: Furnizarea de produse alimentare este o problemă critică care este monitorizată îndeaproape și IoT oferă mijloacele pentru o agricultură inteligentă, unde idei semnificative despre umiditatea solului, nutrienți, consumul de apă, ciclurile de creștere a plantelor, îngrășăminte și multe altele, joacă un rol cheie în cercetarea și avansarea tehnologiei, pentru a crește producția de alimente.
    - Comerț: IoT oferă comercianților posibilitatea de a se conecta cu clienții într-un mod organic care este orientat pentru a îmbunătăți experiența generală din magazin și online. De exemplu, comercianții pot urmări calea unui client prin magazin pentru a maximiza aspectul și a îmbunătăți plasarea produsului.
    - Energie: rețelele electrice inteligente sunt concepute pentru a colecta date și analiza comportamentul consumatorilor și furnizorilor pentru a îmbunătăți utilizarea eficientă a resurselor. În plus, rețelele electrice inteligente sunt capabile să detecteze surse de întrerupere a energiei pentru a permite sisteme de energie distribuite.
    - Asistență medicală: dispozitivele medicale inteligente alimentate de IoT ajută la colectarea datelor pentru a oferi o analiză mai cuprinzătoare a pacienților și chiar pentru a detecta bolile. Prin dispozitivele conectate, furnizorii de servicii medicale pot personaliza consultarea și diagnosticul pacientului.
  1. **Descrierea lucrării și prezentarea problemei**

Scopurile principale ale acestei lucrări sunt de a prezenta nevoia de automatizare a locuinței sau a împrejurimilor locuinței și anume acela de a automatiza irigarea fie a gazonului din curtea casei, fie a grădinii/solarului și beneficiile aduse de acest sistem de irigare. Din dorința de a avea în totdeauna curtea verde, apare de nevoia de a avea un sistem care să rezolve această problemă.

Ceea ce vreau să arăt prin prezenta lucrare este necesitatea deținerii unui astfel de sistem și beneficiile aduse de acesta în urma utilizării în comparație cu irigarea manuală.

Acest subiect mi-a venit în gând în timp ce călătoream cu trenul spre Timișoara în vara anului trecut, dar și în urma unor discuții cu alte persoane care se plângeau de lipsa de timp pentru a efectua lucrurile dorite și lipsa de sisteme automate care să rezolve o parte din aceste lucruri. Văzând starea vremii cu temperaturile ridicate și cu seceta care a fost, și neavând posibilitatea de a iriga gazonul de acasă, m-am gândit la realizarea unui sistem care să facă lucrul acesta în locul meu și la momentul potrivit, astfel economisind timp și având gazonul dorit.

Sistemul pe care vreau să îl prezint pe scurt în acest capitol și dezvoltat în capitolele următoare, este format din două părți: hardware și software.

Partea hardware a sistemului este alcătuită din: o plăcuță Arduino, un modul NodeMCU ESP8266, un releu, o electrovalvă și un senzor de umiditate a solului.

Partea software a sistemului este alcătuită dintr-o aplicație mobilă prin care se controlează și comandă sistemul de irigare.

Aplicația mobilă prezintă două moduri de a utiliza acest sistem: modul de irigare manuală și modul de irigare autoată. Acest al doilea mod, dă posibilitatea irigării în funcție de doi factori: unul din factori este timpul, și anume se irigă în intervalul de timp selectat; iar al doilea factor este senzorul de umiditate, și anume se poate iriga doar atunci când nivelul de umiditate al solului este sub o anumită limită și durează până în momentul în care nivelul de umiditate atinge un anumit prag.

# Aplicații și sisteme similare existente

În acest capitol voi prezenta câteva sisteme similare existente deja pe piață împreună cu aplicațiile lor mobile, dar și câteva aplicații mobile separate care se pot configura pe sistemul personal de irigare.

* 1. **Controllere Rain Bird**4
     1. **Controllerul Rain Bird ESP-RZX4i**

Controllerul Rain Bird ESP-RZX4i este utilizat pentru a controla un număr de maxim 4 electrovane ale sistemului de irigații.

Programatorul este pregătit pentru conexiune WiFi, cu ajutorul căreia se poate realiza programarea și operarea programatorului de la orice dispozitiv cu conexiune internet. Pentru aceasta este necesară adăugarea [modulului Rainbird LNK,](http://gardenium.ro/irigatii/wifi/modul-wifi-rainbird-lnk.html) și o conexiune la internet WiFi stabilă în zona programatorului.

Caracteristici:

* Montaj programator: la interior
* ecran: da, LCD
* funcționare: Automat/Manual
* alimentare: 220V AC- 230V AC, 50Hz
* ieșire: 24V AC, 650mA (transformator inclus)
* 6 timpi de pornire individuali pe fiecare zonă
* programe presetate pe zile ale săptămânii, cu posibilitate de customizare
* timp de functionare pe circuit : 0 - 199 min
* setarea udării sezoniere între : -90% până la 100%
* programe independente pe fiecare zonă
* pornirea manuală a unei zone sau a unui program
* baterii pentru memorie: da, 2 x AAA 1,5 V
  + 1. **Controllerul Rain Bird ESP-RZX8**

Controllerul Rain Bird ESP-RZX8 este utilizat pentru a controla un număr de maxim 8 electrovane ale sistemului de irigații.

Programatorul este pregătit pentru conexiune WiFi, cu ajutorul căreia se poate realiza programarea și operarea programatorului de la orice dispozitiv cu conexiune internet. Pentru aceasta este necesară adăugarea [modulului Rainbird LNK,](http://gardenium.ro/irigatii/wifi/modul-wifi-rainbird-lnk.html) și o conexiune la internet WiFi stabilă în zona programatorului.

Caracteristici:

* Montaj programator: la exterior
* ecran: da, LCD
* funcționare: Automat/Manual
* alimentare: 220V AC- 230V AC, 50Hz
* ieșire: 24V AC, 650mA (transformator inclus, intern)
* 6 timpi de pornire individuali pe fiecare zonă
* programe presetate pe zile ale săptămânii, cu posibilitate de customizare
* timp de funcționare pe circuit : 0 - 199 min
* setarea udării sezoniere între : -90% până la 100%
* programe independente pe fiecare zonă
* pornirea manuală a unei zone sau a unui program
* baterii pentru memorie: da, 2 x AAA 1,5 V
  + 1. **Rain Bird**5

Aceasta este aplicația mobilă aferentă controllerelor Rain Bird.

Udare inteligentă simplă. Conectați controllerul Rain Bird la dispozitivul dvs. Mobil pentru a porni aspersoarele, configurați programe de udare personalizate și automatizați ajustările meteorologice economisind apă și bani.

* 1. **Controllere Hunter**4
     1. **Controller tip X-Core**

Caracteristici :

* Număr de programe independente : 3
* Număr de porniri per program : 4 (max 12 porniri pe zi)
* Durata maximă de funcționare a unei "zone" de udare : 4h
* Compatibil cu orice senzor Hunter Clik
* Pornire manuală printr-o singură apăsare de buton
* Program TEST pentru o evaluare rapidă a sistemului
* Memorie non-volatilă
* Protecție automată la scurt-circuit
* Ajustare procentuală sezonieră : de la 10% la 150% Specificații tehnice :
* Intrare transformator : 230/240V
* Ieșire transformator : 24V, 1A
* Ieșire solenoid : 24V, 0.56A
* Capacitate de comandare simultană a 3 solenoizi
* Temperatura de funcționare : -180 C – +660 C
  + 1. **Controller Hunter X2**

Programator irigații cu capacitate de conectare online (cu ajutorul modulului [WAND *comercializat separat*](https://gardenium.ro/irigatii/wifi/modul-pentru-reteaua-wi-fi-hunter-wand-x2.html)), cu funcții rapide de programare și de optimizare a consumului de apă.

Evolutțe a modelului consacrat Hunter X-Core, cu o serie de caracteristici și capacități similare și foarte eficiente, X2 reprezintă noua generație de management al irigării, cu ajutorul opțiunii de adăugare a conexiunii Wi-Fi la internet pentru control de la distanță, utilizând software-ul Hydrawise, lider de piață în domeniu.

X2 folosește același tip de stil de programare pe bază de meniu cu rotiță ca X-Core, însă poate fi îmbunătățit cu modulul WAND, pentru o programare rapidă, familiară și ușoară prin intermediul unui smartphone sau dispozitiv desktop. Aplicația Hydrawise maximizează economia de apă ajustând automat programul de irigare în baza datelor meteo primite de la diverse surse online.

Caracteristici :

* Număr de zone: 4, 6, 8 sau 14 zone (nemodular)
* Opțiune de control Wi-Fi : [Da, cu modulul opțional WAND](https://gardenium.ro/irigatii/wifi/modul-pentru-reteaua-wi-fi-hunter-wand-x2.html)
* Display iluminat
* Număr programe : 3 programe a câte 4 timpi de pornire și până la 6 ore timp de udare per timp de pornire
* Funcție QuickCheck pentru diagnosticarea avariilor electrice din teren
* Protecție la scurtcircuit : programul sare peste zona cu avarie și continuă udarea, fără eroare
* Memorie Easy Retrieve™ pentru backup al programului de irigare
* Compatibil senzori Click (ploaie, vânt, îngheț, debit, umiditate)
* Ajustare sezonieră, întârziere între stații, oprirea pentru un anumit număr de zile (Rain Delay) și multe alte funcții preluate de la X-Core

Caracteristici modul WiFi :

* Programare rapidă și gestionare online a sistemului de irigare prin aplicația Hydrawise
* Alerte și notificări despre funcționarea sistemului
* Programare standard cu 6 programe a câte 6 timpi de pornire
* Programare avansată cu 36 timpi de pornire și timpi de udare de până la 24 ore
* Predictive Watering™ : Udare predictivă pentru ajustări precise ale programului în funcție de prognoza meteo
* Compatibilitate cu sistemele de automatizare casnică Amazon Alexa™ și Control4®

Date tehnice :

* Curent intrare: 230 V AC
* Curent ieșire transformator: 24 V AC, 1 A
* Curent ieșire electrovane : 24 V AC, 0,56 A
* Curent ieșire releu pompă/electrovană master: 24 V AC, 0,28 A
* Intrări senzor: 1
* Dimensiuni: 23 x 19 x 10 cm
* Standarde controller: IP44, UL, cUL, FCC, CE, RCM
* Standarde modul *(*[*achizitionat separat*](https://gardenium.ro/irigatii/wifi/modul-pentru-reteaua-wi-fi-hunter-wand-x2.html)*)*: Wi-Fi b/g/n, Bluetooth 5.0, CE, UL, RCM, FCC
  + 1. **Hydrawise Irrigation**5

Sistemul de control al Hunter de irigare de la distanță, utilizând această aplicație sau folosind un browser web de oriunde din lume.

* 1. **ii.ri Smart Irrigation Control**5

ii.ri este primul controler de irigare care permite oricui să controleze propriul lor sistem computerizat de irigat gradina printr-o aplicație smartphone ușor de folosit.

ii.ri este un controler inteligent care face totul ușor, cu un dispozitiv simplu, dar genial

ii.ri este compatibil cu orice tip de robinet disponibil pe piață

* 1. **MySOLEM**5

Aplicația MySOLEM vă permite să construiți programe de irigare precum și monitorizarea contactelor electrice.

Această aplicație vă permite, de asemenea, să gestionați manual modulele instalate în grădină.

Programele sau comenzile manuale sunt transmise prin intermediul unor module Wi-Fi sau Bluetooth.

# Concepte teoretice

În acest capitol vor fi prezentate atât aspectele teoretice folosite, cât și programele utilizate în realizarea acestui proiect de diplomă.

* 1. **Arduino IDE**

„Proiectul Arduino oferă un mediu integrat de dezvoltare (IDE), care este o aplicație [cross-](https://ro.wikipedia.org/wiki/Cross-platform) [platform,](https://ro.wikipedia.org/wiki/Cross-platform) scrisă în [Java](https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)). Acesta își are originile în mediul de dezvoltare pentru limbajul de programare Processing și în proiectul Wiring. Este proiectat pentru a introduce programarea în lumea artiștilor și a celor nefamiliarizați cu dezvoltarea software. Include un editor de cod cu funcții ca evidențierea sintaxelor, potrivirea acoladelor și spațierea automată și oferă mecanisme simple cu un singur click, pentru a compila și a încărca programele în plăcuța Arduino.

Arduino IDE suportă limbajele de programare C și C++ folosind reguli special de organizare a codului. Arduino IDE oferă o librărie software numită Wiring, din proiectul Wiring, care oferă multe procedure comune de intrare și ieșire.”6

* 1. **Android Studio**

„Android Studio este [un mediu de dezvoltare](https://ro.wikipedia.org/wiki/Mediu_de_dezvoltare) pentru colaborarea cu platforma Android, anunțată pe data de 16 mai 2013 în cadrul conferinței I / O Google.

Android Studio este bazat pe software-ul IntelliJ IDEA de la JetBrains, este instrumentul oficial de dezvoltare a aplicațiilor Android. Acest mediu de dezvoltare este disponibil pentru Windows, OS X și Linux. Pe 17 mai 2017, la conferința anuală Google I / O, Google a anunțat asistență pentru limbajul Kotlin utilizat de Android Studio ca limbaj de programare oficial pentru platforma Android, pe lângă Java și C ++.”6

* 1. **Limbaje de programare**

### Limbajul de programare C

**„C** este [un limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) [standardizat](https://ro.wikipedia.org/wiki/Standard). Este implementat pe majoritatea platformelor de calcul existente azi, și este cel mai popular limbaj de programare pentru scrierea de [software de sistem.](https://ro.wikipedia.org/w/index.php?title=Software_de_sistem&amp;action=edit&amp;redlink=1) Este apreciat pentru eficiența [codului obiect](https://ro.wikipedia.org/w/index.php?title=Cod_obiect&amp;action=edit&amp;redlink=1) generat de compilatoarele **C**, și pentru [portabilitatea](https://ro.wikipedia.org/w/index.php?title=Portabilitate&amp;action=edit&amp;redlink=1) sa.

**C** este [un limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) relativ minimalist ce operează în strânsă legătură cu [hardware](https://ro.wikipedia.org/wiki/Hardware)-ul, fiind cel mai apropiat de [limbajul de asamblare](https://ro.wikipedia.org/wiki/Limbaj_de_asamblare) față de majoritatea celorlalte limbaje de programare.”6

În această lucrare, limbajul de programare C a fost utilizat în programul Arduino pentru a programa modulul Wi-Fi, NodeMCU ESP8266, care asigură legătura între partea hardware a sistemului de irigare și partea software a acestuia.

### Limbajul de programare Java

„Java este [un limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) orientat-[obiect](https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte), puternic tipizat, conceput de către James Gosling la [Sun Microsystems](https://ro.wikipedia.org/wiki/Sun_Microsystems) (acum filială [Oracle](https://ro.wikipedia.org/wiki/Oracle)) la începutul [anilor ʼ90,](https://ro.wikipedia.org/wiki/Anii_1990) fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la [C](https://ro.wikipedia.org/wiki/C) și [C++](https://ro.wikipedia.org/wiki/C%2B%2B), dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java ([engleză](https://ro.wikipedia.org/wiki/Limba_englez%C4%83) *Java Virtual Machine*, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi [C](https://ro.wikipedia.org/wiki/C)) este posibil deoarece sursele Java sunt compilate într-un format *standard* numit cod de octeți ([engleză](https://ro.wikipedia.org/wiki/Limba_englez%C4%83) *byte-code*) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.”6

Limbajul de programare Java a fost folosit în programul Android Studio pentru realizarea părții software a sistemului de irigare, adică a aplicației mobile cu ajutorul căreia se exectă procesul de irigare.

* 1. **Sistemul și componentele sistemului**

### Ce este un sistem?

„Un sistem este o mulțime de obiecte interconectate care prin interconectare alcătuiesc un tot unitar, bine încadrat în timp, bine delimitat spațial față de mediul exterior de care este influențat și care poate fi privit ca un ansamblu structurat, având un anumit scop și îndeplinind o anumită funcție.

Obiectele unui sistem fizic interacţionează, interacţiunile realizându-se prin fluxuri de energie / masă /informaţie. Prelucrările fluxurilor de către sisteme sunt denumite procese. Vorbim despre procese de transfer de energie / masă / informaţie. Modelele matematice, imagini ale sistemelor fizice, redau transferurile numai din punct de vedere informațional.”3

### Plăcuța Arduino Uno

O plăcuță Arduino este compusă dintr-un microcontrole[r Atmel](https://ro.wikipedia.org/w/index.php?title=Atmel&amp;action=edit&amp;redlink=1) [AVR](https://ro.wikipedia.org/w/index.php?title=AVR&amp;action=edit&amp;redlink=1) de 8, 16 sau 32-biți cu componente complementare care facilitează programarea și încorporarea în alte circuite. Un aspect important la Arduino este că acesta dispune de conectori standard, care permit utilizatorului să conecteze plăcuța cu procesorul la diferite module interschimbabile.

La nivel conceptual, când se folosește mediul de dezvoltare integrat Arduino, programarea tuturor plăcuțelor se face prin conexiune serială. Implementarea acesteia diferă în funcție de versiunea hardware. Plăcuțele Arduino din prezent sunt programate prin USB, având integrate cipuri de conversie USB-serial.

### Plăcuța NodeMCU ESP8266

Placa de dezvoltate WiFi bazată pe ESP8266, un circuit integrat care conține module GPIO, PWM, IIC, 1-Wire și ADC, toate pe o singură placă și facil de utilizat cu sintaxa gen Arduino pentru acces la hardware și API în stilul NodeJS pentru utilizarea rețelei.

### Releu

Releul este o componentă electronică, un dispozitiv, care produce anumite modificări (cum ar fi închiderea și deschiderea unui circuit) pe baza unui parametru care variază (precum tensiunea electrică aplicată), permițând controlarea unui curent de intensitate mare cu ajutorul unui curent de intensitate mică.

### Electovalvă

Electrovalva este o componentă mecanică acționată electric. Prin acționarea curentului electric asupra solenoidului (parte componentă a electrovalvei), acesta glisează pentru a permite trecerea lichidului prin ea. Este asemenea unui robinet, dar acționat electric.

* 1. **Programarea/dezvoltarea aplicațiilor mobile pentru Android**

Procesul de dezvoltare al unei aplicații mobile constă în urmarea câtorva etape esențiale în economia proiectului, și anume:

* **Proiectarea aplicației** → Identificarea nevoilor clientului: este esențial să se identifice nevoile clientului pentru că această etapă este baza pentru dezvoltarea și proiectarea unei aplicații
* **Aspectul grafic** → Alocarea resurselor: aspectul grafic sau design-ul aplicației se va realiza tot cu ajutorul clientului, respectând viziunea și nevoile acestuia
* **Implementarea funcțiilor →** În funcție de cerințe: aplicația mobilă poate oferii diferite funcționalități, de la cele mai simple, până la cele mai complexe, în conformitate cu nevoile și cerințele clientului
* **Testarea aplicației** → O perioadă de timp: testarea apliacției este un aspect important și necesar pentru a asigura funcționalitatea corespunzătoare cerințelor clientului
* **Lansarea aplicației** → Faza finală: lansarea aplicației reprezintă ultima fază în procesul de dezvoltare

În strânsă legătură cu etapele prezentate, și o continuare a acestor etape sunt și mentenanța aplicației, cât și îmbunătățirea acesteia. Este nevoie de îmbunătățirea aplicației mobile pentru a fi în pas cu dezvoltarea și evoluția tehnologiilor, dar și pentru a-și păstra utilitatea odată cu schimbările apărute în jurul ei.

Orice aplicație mobilă trebuie să conțină obligatoriu anumite fișiere pentru a funcționa.

Unul dintre aceste fișiere este *AndroidManifest.xml.*

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<manifest ... >

<!-- ... -->

<application ... >

<activity android:name=**".MainActivity"** android:label=**"@string/app\_name"** >

<intent-filter>

<action android:name=**"android.intent.action.MAIN"** />

<category android:name=**"android.intent.category.LAUNCHER"** />

</intent-filter>

</activity>

</application>

</manifest>

În acest fișier trebuie să apară toate activitățile prezente în aplicație, iar acestea trebuie incluse în intrarea de tip *<activity> </activity>.* În interiorul acestui tip de intrare este obligatoriu să se specifice doar denumirea activității *„android:name”.*

Ulterior, se va implementa clasa identificată prin denumirea activității declarate în fișierul de mai sus. Această clasă trebuie să extindă clasa *android.app.Activity* și trebuie să implementeze cel puțin o metodă din această clasă, și anume metoda *onCreate()* în care sunt inițializate componentele sale. Fișierul *MainActivity.java* este următorul:

public class MainActivity **extends** Activity **{**

@Override

protected void onCreate**(**Bundle savedInstanceState**) { super.**onCreate**(**savedInstanceState**);** setContentView**(**R**.**layout**.**activity\_main**);**

// ...

**}**

// ...

**}**

Fișierul *MainActivity.java* este implementarea funcționalităților componentelor grafice prezente într-o aplicație mobilă. Elementele grafice ale aplicației mobile se găsesc în fișiere cu extensia *.xml*. Pentru fișiereul prezentat mai sus, fișiereul *.xml* este *activity\_main.xml*.

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)

xmlns:app=[**"http://schemas.android.com/apk/res**](http://schemas.android.com/apk/res-auto)**-**[**auto"**](http://schemas.android.com/apk/res-auto)xmlns:tools=[**"http://schemas.android.com/tools"**](http://schemas.android.com/tools)android:layout\_width=**"match\_parent"** android:layout\_height=**"match\_parent"** tools:context=**".MainActivity"**>

<TextView

android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Hello World!"** app:layout\_constraintBottom\_toBottomOf=**"parent"** app:layout\_constraintLeft\_toLeftOf=**"parent"** app:layout\_constraintRight\_toRightOf=**"parent"** app:layout\_constraintTop\_toTopOf=**"parent"**

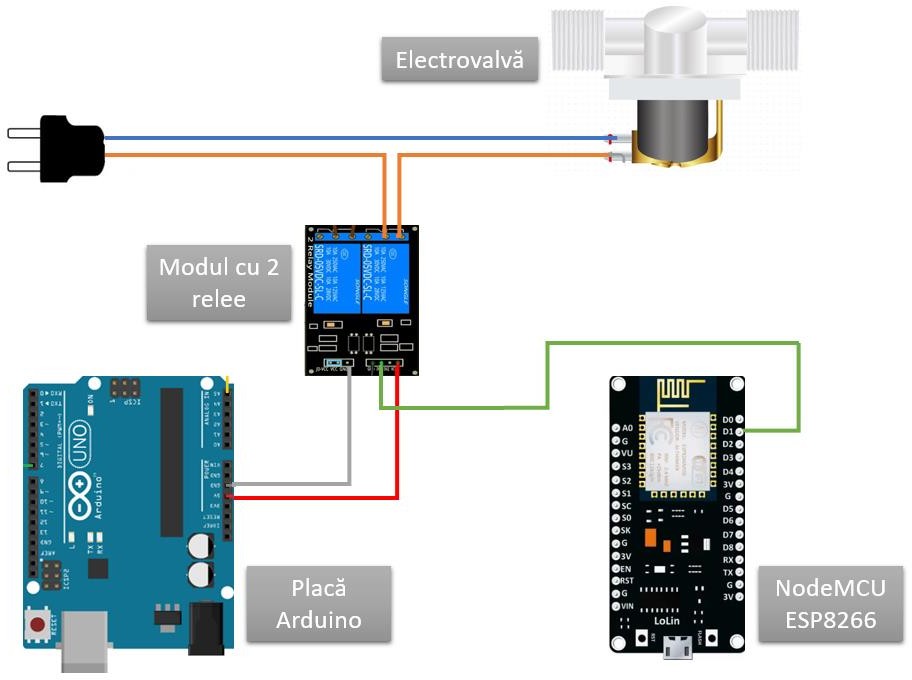
android:id=**"@+id/main\_text"** android:textSize=**"42sp"** android:layout\_margin=**"5sp"** android:textAlignment=**"center"** android:gravity=**"center\_horizontal"** />

</androidx.constraintlayout.widget.ConstraintLayout>

# Implementarea soluției sistemului

În acest capitol urmează să discutăm despre schema logică a sistemului de irigare, arhitectura acestuia, cât și despre implementarea soluției pentru partea harware și software.

* 1. **Schema logică a sistemului**



*Fig. 4.1 Schema logică a sistemului de irigare*

După cum se poate observa în figura de mai sus, componentele care alcătuiesc sistemul de irigare sunt:

* Placă Arduino Uno
* Modulul NodeMCU ESP8266
* Modul cu 2 relee
* Electrovalvă
* Fire de legătură

Placa Arduino Uno este folosită pentru a asigura tensiunea de 5V necesară funcționării releului. În acest scop, releul este conectat la placa Arduino Uno la pinul de 5V (Output) și la pinul GND, pin care asigură pământarea circuitului electric.

Releul este controlat cu ajutorul modulului NodeMCU ESP8266. Legătura dintre aceste două componente este realizată la pinul IN1 a releului și la pinul D1 al modulului NodeMCU ESP8266. Pinul D1 al modulului NodeMCU ESP8266 este pinul GPIO5 (General Purpose Input/Output).

Releul are trei tipuri de ieșiri: NC (Normally Closed), COM (Common) și NO (Normally Open). Ieșirea COM este partea comună între celelalte două ieșiri.

Modul de funcționare al ieșirii NC este periculos în anumite situații pentru că circuitul conectat la această ieșire va funcționa chiar dacă se întrerupe alimentarea cu curent electric.

Modul de funcționare al ieșirii NO este unul sigur pentru că în momentul în care se întrerupe alimentarea cu curent electric, funcționarea circuitului se va opri.

Având în vedere aceste două moduri de utilizare a ieșirilor releului, pentru acest sistem de irigare, electrovalva este conectată la ieșirea COM și la ieșirea NO a relelului.

Modulul NodeMCU ESP8266 este componenta sistemului prin care se asigură conexiunea dintre sistemul fizic și aplicația mobilă.

Dintre toate componentele sistemului de irigare, modulul NodeMCU ESP8266 este singurul modul programat. Codul aferent programării modulului este prezentat în subcapitolul următor.

* 1. **Programarea modulului NodeMCU ESP8266**

În acest subcapitol este prezentat codul aferent programării modulului NodeMCU ESP8266 și analiza acestui cod.

Codul se găsesște în fișierul *HidroAppArdu.ino.*

#include <ESP8266WiFi.h> #include <ESP8266WebServer.h>

#ifdef ESP32

#pragma message(THIS EXAMPLE IS FOR ESP8266 ONLY!)

#error Select ESP8266 board. #endif

ESP8266WebServer server**(**80**);** // (80) is the port number const char**\*** ssid **=** "ssidWifi"**;**

const char**\*** password **=** "passwordWifi"**;**

String waterOn ="", waterOff = "";

void WaterOn**()**

**{**

digitalWrite**(**D1**,** LOW**);**

server**.**send**(**200**,** "text/html"**,** waterOn**);**

**}**

void WaterOff**()**

**{**

digitalWrite**(**D1**,** HIGH**);**

server**.**send**(**200**,** "text/html"**,** waterOff**);**

**}**

void setup**() {** pinMode**(**D1**,** OUTPUT**);** digitalWrite**(**D1**,** HIGH**);**

WiFi**.**begin**(**ssid**,** password**);** Serial**.**begin**(**115200**);** WiFi**.**mode**(**WIFI\_STA**);**

**while (**WiFi**.**status**() !=** WL\_CONNECTED**)** delay**(**500**);**

Serial**.**print**(**"IP Address: "**);** Serial**.**print**(**WiFi**.**localIP**());**

server**.**on**(**"/waterOn"**,** WaterOn**);** server**.**on**(**"/waterOff"**,** WaterOff**);**

server**.**begin**();**

**}**

void loop**()**

**{**

server**.**handleClient**();** delay**(**1**);**

**}**

Pentru ca modulul NodeMCU ESP8266 să se poată conecta la internet prin intermediul rețelei locale, în fișierul de mai sus s-au inclus cele două biblioteci *ESP8266WiFi.h* și *ESP8266WebServer.h.*

Pentru a se putea conecta la rețeaua locală de internet, modulul are nevoie de *ssid-ul* rețelei și de *password-ul* acesteia. În acest scop s-au declarat cele două șiruri de caractere cu nume sugestive conținând ssid-ul, respectiv parola relțelei.

Funția *void setup()* este una din cele două funcții obligatorii a unui fișier *.ino.* În interiorul acestei funcții, se declară pinul D1 al modulului NodeMCU ESP8266 ca fiind pin de ieșire cu ajutorul funcției *pinMode()*.

Prin apearea funcției *WiFi.begin(ssid, password)* se inițializează conexiunea efectivă la WiFi, iar prin funcția *WiFI.mode(WIFI\_STA)* se specifică faptul că realizarea conexiunii este la un punct static.

În bucla *while()* se verifică dacă s-a finalizat procesul de conectare la internet. Durata acestei bucle este până în momentul în care funcția *WiFi.status()* primește răspunsul *WL\_CONNECTED.*

Funția *server.on()* așteaptă ca să fie solicitată în browser *„/waterOn”* sau *„/waterOff”,* după caz. În momentul în care apare această solicitare, această funcție apelează funcția *WaterOn()* sau *WaterOff()*. Dacă se solicită *„/waterOn”* sau *„/waterOff”*, serverul, prin funcția *server.send()* trebuie să răspundă cu un cod de stare HTTP de 200 și cu șirul de caractere *waterOn* sau *waterOff.*

După apelarea funcției corespunzătoare cererii, funcția *server.begin()* „ascultă” cererile HTTP. În timpul buclei, se verifică în permanență dacă se primește o nouă cerere cu ajutorul funcției *server.handleClient(),* funcție aflată în cea de a doua funcție obligatorie dintr-un fișier

*.ino*. Această funcție este *void loop()*.

* 1. **Aplicația mobilă**

### AndroidManifest.xml

Așa cum am prezentat în capitolul anterior, orice aplicație mobilă conține un fișier cu numele *AndroidManifest.xml,* fișier în care este obligatoriu să apară toate activitățile existente în aplicație. Pe lângă activitățile prezente în aplicație, care sunt specificate în interiorul intrărilor de tip *<activity> </activity>*, tot în acest fișier trebuie să apară permisiunile aplicației. Aceste permisiuni vor fi adăugate în interiorul intrărilor de tip *<uses-permission> </uses-permission>*.

Conținutul fișierului *AndroidManifest.xml* corespunzător aplicației mobile a acestui sitem de irigare este următorul:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<manifest xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)package=**"com.jetbrains.handson.mpp.hidroapp"**>

<uses-permission android:name=**"android.permission.INTERNET"**/>

<uses-permission android:name=**"android.permission.ACCESS\_NETWORK\_STATE"**

/>

<application

android:name=**".Notification"** android:allowBackup=**"true"** android:icon=**"@mipmap/ic\_launcher"** android:label=**"@string/app\_name"** android:roundIcon=**"@mipmap/ic\_launcher\_round"** android:supportsRtl=**"true"** android:theme=**"@style/AppTheme"**>

<activity android:name=**".IrigareAutomata"** android:parentActivityName=**".MainActivity"**/>

<activity

android:name=**".IrigareManuala"** android:parentActivityName=**".MainActivity"** />

<activity android:name=**".MainActivity"**>

<intent-filter>

<action android:name=**"android.intent.action.MAIN"** />

<category android:name=**"android.intent.category.LAUNCHER"** />

</intent-filter>

</activity>

<receiver android:name=**".Alarm"**/>

</application>

</manifest>

### Activity\_main.xml

Activitatea principală a unei aplicații mobile este activitatea cu care pornește aplicația respectivă. Această activitate poartă numele de *activity\_main*.

Fișierul *activity\_main.xml* este variat de la o aplicație mobilă la alta. Acest fișier conține o serie de intrări specifice anumitor elemente, iar fiecare dintre aceste intrări conțin o serie de proprietăți specifice acelui element. Câteva dintre proprietățile comune tuturor tipurilor de intrări sunt *id-ul* și proprietățile care țin de poziționarea elementului.

În această aplicație, fișierul *activity\_main.xml* conține două intrări de tip *TextView*, o intrare de tip *EditText* și două intrări de tip *Button.*

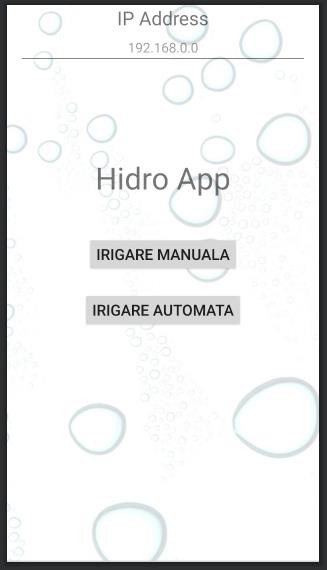
Prima intrare de tip *TextView* afișează în această activitate textul *IP Address*, specificat în proprietatea *android:text*.

Următorul element din activitatea principală este *EditText*. Acest tip de element permite introducerea de date. În această aplicație, acest element este folosit pentru a obține adresa IP a modulului NodeMCU ESP8266.

A doua intrare de tip *TextView* este poziționată în partea centrală a activității și afișează în activitatea principală numele aplicației *HidroApp.*

Ultimele două elemente prezente în activitatea principală sunt două elemente de tip *Button*. Aceste butoane sunt pentru cele două tipuri de irigare prezentate în capitolele anterioare. Unul dintre butoane, cu numele *Irigare Manuală* este pentru a naviga în activitatea specifică acestei operațiuni, iar al doilea buton, cu numele *Irigare Automată* este și acesta specific pentru activitatea respectivă.

Această activitate, din punct de vedere grafic se prezintă astfel:



*Fig. 4.2 Activitatea principală a aplicației mobile*

Conținutul fișierului *activity\_main.xml* este următorul:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<RelativeLayout xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)xmlns:app=[**"http://schemas.android.com/apk/res**](http://schemas.android.com/apk/res-auto)**-**[**auto"**](http://schemas.android.com/apk/res-auto)xmlns:tools=[**"http://schemas.android.com/tools"**](http://schemas.android.com/tools)android:layout\_width=**"match\_parent"** android:layout\_height=**"match\_parent"**

tools:context=**".MainActivity"** android:background=**"@drawable/rain\_backg\_2"**

>

<TextView

android:id=**"@+id/textviewip"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"IP Address"** android:textAlignment=**"center"** android:gravity=**"center\_horizontal"** android:textSize=**"25dp"** android:layout\_centerHorizontal=**"true"**/>

<EditText

android:id=**"@+id/edittext1"** android:layout\_width=**"match\_parent"** android:layout\_height=**"wrap\_content"** android:layout\_marginStart=**"16dp"** android:layout\_marginEnd=**"16dp"** android:inputType=**"text"** android:hint=**"192.168.0.0"** android:textAlignment=**"center"** android:gravity=**"center\_horizontal"** android:layout\_below=**"@+id/textviewip"**/>

<TextView

android:id=**"@+id/textview1"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Hidro App"** android:textSize=**"40sp"** android:layout\_marginTop=**"200dp"** android:layout\_centerHorizontal=**"true"**/>

<Button

android:id=**"@+id/button\_irig\_man"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Irigare Manuala"** android:textSize=**"20sp"** android:layout\_below=**"@+id/textview1"** android:layout\_centerHorizontal=**"true"** android:layout\_marginTop=**"50dp"** />

<Button

android:id=**"@+id/button\_irig\_auto"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Irigare Automata"** android:textSize=**"20sp"** android:layout\_below=**"@+id/button\_irig\_man"** android:layout\_centerHorizontal=**"true"** android:layout\_marginTop=**"25dp"**/>

</RelativeLayout>

### MainActivity.java

În strânsă legătură cu fișierul prezentat mai sus, *activity\_main.xml*, este acest fișier, *MainActivity.java*, pentru că acest fișier conține toate implementările funcționalităților elementelor prezentate.

Pentru a putea implementa funcționalitatea fiecărui element, respectivul element trebuie identificat. Acest lucru se realizează cu ajutorul metodei *findViewById()* și cu ajutorul *id-ului* specificat ca proprietate a fiecărui element.

Button button\_man **= (**Button**)** findViewById**(**R**.**id**.**button\_irig\_man**);**

Am spus mai sus că fiecare buton, la acționare, te transferă în activitatea specifică butonului. Acest lucru este posibil datorită metodei *setOnClickListener()*.

Metoda *setOnClickListener()* se apelează pe obiectul asupra căruia se vrea să se acționeze. În interiorul acestei metode se crează o nouă instanță *new View.OnClickListener()*, unde se supra scrie metoda *onClick(View view)*. În corpul acestei metode se inserează codul pentru funcționalitatea elementului respectiv.

|  |
| --- |
| button\_man**.**setOnClickListener**(new** View**.**OnClickListener**() {**  @Override  public void onClick**(**View view**) {** |
| openIrigareManuala**();**  **}**  **});** |

Legătura dintre activitatea curentă și noua activitate se realizează cu ajutorul clasei *Intent.* Se crează o nouă instanță a clasei *Intent* cu parametrii *this* (adică activitatea curentă) și clasa activității respective. De exemplu *IrigareManuala.class.* Pentru a putea trimite conținutul unor variabile în noua activitate, se folosește metoda *putExtra(),* apelată pe instanța clasei *Intent.*

Intent intent **= new** Intent**(this,** IrigareManuala**.**class**);** intent**.**putExtra**(**EXTRA\_TEXT**,** text**);**

Pentru a porni noua activitate, se apeleaza metoda *startActivity(),* primind ca și parametru instanța clasei *Intent*.

Conținutul fișierului *MainActivity.java* este prezentat în cele ce urmează:

package com**.**jetbrains**.**handson**.**mpp**.**hidroapp**;**

**import** androidx**.**appcompat**.**app**.**AppCompatActivity**; import** android**.**content**.**Intent**;**

**import** android**.**os**.**Bundle**; import** android**.**view**.**View**; import** android**.**widget**.**Button**; import** android**.**widget**.**EditText**;**

public class MainActivity **extends** AppCompatActivity **{**

public static final String EXTRA\_TEXT **=**

"com.example.hidroapp.example.EXTRA\_TEXT"**;**

@Override

protected void onCreate**(**Bundle savedInstanceState**) { super.**onCreate**(**savedInstanceState**);** setContentView**(**R**.**layout**.**activity\_main**);**

Button button\_man **= (**Button**)** findViewById**(**R**.**id**.**button\_irig\_man**);** button\_man**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

openIrigareManuala**();**

**}**

**});**

Button button\_auto **= (**Button**)** findViewById**(**R**.**id**.**button\_irig\_auto**);**

button\_auto**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

openIrigareAutomata**();**

**}**

**});**

**}**

public void openIrigareManuala**() {**

EditText editText **= (**EditText**)** findViewById**(**R**.**id**.**edittext1**);** String text **=** editText**.**getText**().**toString**();**

Intent intent **= new** Intent**(this,** IrigareManuala**.**class**);** intent**.**putExtra**(**EXTRA\_TEXT**,** text**);**

startActivity**(**intent**);**

**}**

public void openIrigareAutomata**() {**

EditText editText **= (**EditText**)** findViewById**(**R**.**id**.**edittext1**);** String text **=** editText**.**getText**().**toString**();**

Intent intent **= new** Intent**(this,** IrigareAutomata**.**class**);** intent**.**putExtra**(**EXTRA\_TEXT**,** text**);**

startActivity**(**intent**);**

**}**

**}**

### Activity\_irigare\_manuala.xml

Activitatea corespunzătoare procesului de irigare manuală prezintă în fișierul *activity\_irigare\_manuala.xml* trei elemente distincte: un element de tip *TextView*, un element de tip *SwitchCompat* și un element de tip *Chronometer*.

Elementul de tip *TextView* afișează pe ecran denumirea activității curente, și anume *Irigare Manuală.* Acest text este memorat cu ajutorul proprietății *android:text.*

Elementul de tip *SwitchCompat* conține și afișează pe ecran două componente principale. Prima dintre aceste componente este numele sau semnificația butonului, adică *Irigă.* Cea de a două componentă principală a acestui element este butonul de switch în sine. Acest buton a fost creat cu ajutorul a două proprietăți: *android:thumb* și *app:track.* Pentru fiecare dintre aceste două proprietăți a fost creat câte un fișier.

Pentur proprietatea *android:thumb*, fișierul este *sw\_thumb.xml*:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<selector xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)>

<item android:state\_checked=**"true"**>

<shape android:shape=**"rectangle"**>

<solid android:color=**"#FFF"**/>

<size android:height=**"50dp"** android:width=**"70dp"**/>

<corners android:radius=**"40dp"**/>

<stroke android:width=**"2dp"** android:color=**"#3AF610"**/>

</shape>

</item>

<item android:state\_checked=**"false"**>

<shape android:shape=**"rectangle"**>

<solid android:color=**"#FFF"**/>

<size android:height=**"50dp"** android:width=**"70dp"**/>

<corners android:radius=**"40dp"**/>

<stroke android:width=**"2dp"** android:color=**"#cdcdcd"**/>

</shape>

</item>

</selector>

În acest fișier sunt prezentate două intrări de tip *item.* Prima intrare de tip *item* este pentru poziția *on* sau *verificată* a butonului. În interiorul acestei intrări sunt prezente alte patru intrări pentru a contura și colora acest buton cu proprietăți specifice pentru momentul în care acesta este activat.

A doua intrare de tip *item* este pentru poziția *off* sau *neverificată* a butonului. Asemenea poziției de *on*, sunt prezente proprietăți specifice și pentru această poziție a butonului. În interiorul intrării *item* sunt prezente aceleași patru proprietăți ca mai sus, dar cu valori diferite.

Acest fișier este pentru partea orizontală a butonului sau suportul pe care va glisa partea verticală a butonului, așa cum se poate observa în figura următoare.



*Fig 4.3 Partea grafică orizontală a butonului de switch*

Pentru proprietatea *app:track*, fișierul este *sw\_track.xml*:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<selector xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)>

<item android:state\_checked=**"true"**>

<shape android:shape=**"rectangle"**>

<solid android:color=**"#3AF610"**/>

<corners android:radius=**"200dp"**/>

</shape>

</item>

<item android:state\_checked=**"false"**>

<shape android:shape=**"rectangle"**>

<solid android:color=**"#cdcdcd"**/>

<corners android:radius=**"200dp"**/>

</shape>

</item>

</selector>

În acest fișier sunt prezentate două intrări de tip *item.* Prima intrare de tip *item* este pentru poziția *on* sau *verificată* a butonului. În interiorul acestei intrări sunt prezente alte două intrări pentru a contura și colora acest buton cu proprietăți specifice pentru momentul în care acesta este activat.

A doua intrare de tip *item* este pentru poziția *off* sau *neverificată* a butonului. Asemenea poziției de *on*, sunt prezente proprietăți specifice și pentru această poziție a butonului. În interiorul intrării *item* sunt prezente aceleași două proprietăți ca mai sus, dar cu valori diferite.

Acest fișier este pentru partea verticală a butonului sau partea care va glisa partea orizontală a butonului, așa cum se poate observa în figura următoare.



*Fig. 4.4 Partea grafică verticală a butonului de switch*

Cel de al treilea element conținut în activitatea de irigare manuală este *Chronometer*. Acesta este activat în momentul în care butonul prezentat mai sus este comutat pe poziția *on.* Acest element este folosit doar pentru contorizarea timpului de irigare.

Conținutul fișierului *activity\_irigare\_manuala.xml* este următorul:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<RelativeLayout xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)xmlns:app=[**"http://schemas.android.com/apk/res**](http://schemas.android.com/apk/res-auto)**-**[**auto"**](http://schemas.android.com/apk/res-auto)xmlns:tools=[**"http://schemas.android.com/tools"**](http://schemas.android.com/tools)android:layout\_width=**"match\_parent"** android:layout\_height=**"match\_parent"**

tools:context=**".IrigareManuala"** android:background=**"@drawable/rain\_backg\_2"**>

<TextView

android:id=**"@+id/textview2"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Irigare Manuala"** android:textSize=**"30sp"** android:layout\_centerHorizontal=**"true"** android:layout\_marginTop=**"20dp"**/>

<androidx.appcompat.widget.SwitchCompat android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:id=**"@+id/switchButtonMan"** android:text=**"Irigă "** android:textSize=**"45sp"** android:textStyle=**"bold"** android:textOff=**"OFF"** android:textOn=**"ON"** android:gravity=**"center"** android:layout\_marginTop=**"40dp"** android:thumb=**"@drawable/sw\_thumb"** app:track=**"@drawable/sw\_track"**

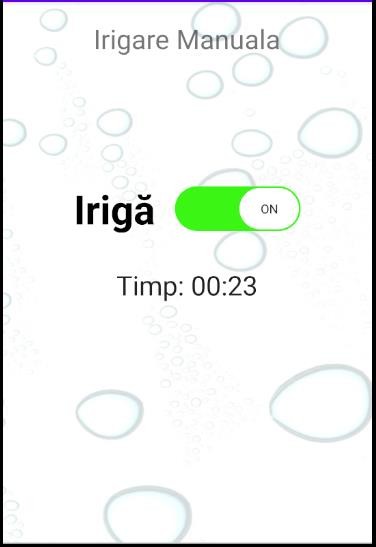
app:showText=**"true"** android:layout\_centerHorizontal=**"true"** android:layout\_marginVertical=**"200dp"**/>

<Chronometer

android:id=**"@+id/chronometerMan"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:layout\_below=**"@+id/switchButtonMan"** android:layout\_marginTop=**"-165dp"** android:layout\_centerHorizontal=**"true"** android:textSize=**"30sp"** />

</RelativeLayout>

Această activitate din punct de vedere grafic se prezintă astfel:



*Fig. 4.5 și 4.6 Activitatea de irigare manuală inainte de pornirea sistemului de irigare și în timpul funcționării acestuia*

### IrigareManuala.java

Acest fișier conține implementarea funcționalităților elementelor prezente în activitatea de irigare manuală.

În momentul în care am prezentat conținutul fișierului *MainActivity.java* am precizat faptul că legătura dintre activitatea curentă și noua activitate se realizează cu ajutorul clasei *Intent.* Se crează o nouă instanță a clasei *Intent* cu parametrii *this* (adică activitatea curentă) și clasa activității respective, iar pentru a putea trimite conținutul unor variabile în noua activitate, se folosește metoda *putExtra(),* apelată pe instanța clasei *Intent.*

Intent intent **= new** Intent**(this,** IrigareManuala**.**class**);** intent**.**putExtra**(**EXTRA\_TEXT**,** text**);**

Conținutul variabilei trimise din activitatea principală în această activitate este conținutul intrării *EditText* din *activity\_main.xml,* adică adresa IP introdusă.

Pentru a putea prelua conținutul variabilei transmise, în acest fișier, *IrigareManuala.java*, se crează o instanță a clasei *Intent* și se apelează metoda *getIntent()*.

Intent intent **=** getIntent**();**

text **=** intent**.**getStringExtra**(**MainActivity**.**EXTRA\_TEXT**);**

Transmiterea și preluarea conținutului acestei variabile este necesar pentru a putea transmite mai departe din clasa curentă spre clasele care se ocupă de pornirea și oprirea sistemului de irigare, *StartWater.java* respectiv *StopWater.java*.

Pentru a identifica butonul de switch din *activity\_irigare\_manual.xml* se apelează metoda *findViewById()*. Iar pentru a oferi funcțioanlitatea dorită acestui element, pe instanța creată se apelează metoda *setOnClickListener()* dând ca și parametru acestei metode o nouă instanță a clasei *View.OnClickListener()*.

final SwitchCompat switchCompat **=** findViewById**(**R**.**id**.**switchButtonMan**);**

switchCompat**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {}**

**});**

În interiorul acestei metode se verifică cu ajutorul instrucțiunii de decizie *if* dacă butonul de switch este activ sau nu.

În cazul în care butonul este activ, se pornește cronometrul prin apelarea metodei *start()*, se crează un obiect instanță a clasei *StartWater* și se apelează metoda *doInBackground()* pe instanța creată.

Pe lângă faptul că se pornește irigarea, în momentul în care se apelează metoda *doInBackground()*, se mai apelează o metodă *sendNotificațion()* cu ajutorul căreia se primește pe telefon o notificare prin care se anunță pornirea sistemului de irigare.

În cazul în care butonul devine inactiv, se oprește cronometrul prin apelarea metodei *stop()*, se crează un obiect instanță a clasei *StopWater* și se apelează metoda *doInBackground()* pe instanța creată. Cu toate că se apelează metoda *stop()* pentru oprirea cronometrului, în momentul în care se reactivează butonul de switch, acesta va continua numărătoarea. Cronometrul se resetează doar după ce a funcționat timp de o oră. Acest aspect se realizează în interiorul metodei *setOnChronometerTickListener()*.

Pentru a putea trimite o notificare este nevoie de creea unui canal de notificare. Acest aspect este obligatoriu începând cu versiunea Android 8.0 (API level 26).8 Canalul de notificări pentru aplicația aceasta a fost creat în fișierul *Notification.java*.

Conținutul fișierului *Notification.java* este următorul:

package com**.**jetbrains**.**handson**.**mpp**.**hidroapp**;**

**import** android**.**app**.**Application**;**

**import** android**.**app**.**NotificationChannel**; import** android**.**app**.**NotificationManager**; import** android**.**os**.**Build**;**

public class Notification **extends** Application **{**

public static final String CHANNEL\_1\_ID **=** "channel1"**;** @Override

public void onCreate**() {**

**super.**onCreate**();**

createNotificationChannel**();**

**}**

private void createNotificationChannel**() {**

**if (**Build**.**VERSION**.**SDK\_INT **>=** Build**.**VERSION\_CODES**.**O**) {**

NotificationChannel channel **= new** NotificationChannel**(**CHANNEL\_1\_ID**,** "Channel 1"**,** NotificationManager**.**IMPORTANCE\_HIGH**);**

channel**.**setDescription**(**"This is channel 1"**);**

NotificationManager manager **=**

getSystemService**(**NotificationManager**.**class**);** manager**.**createNotificationChannel**(**channel**);**

**}**

**}**

**}**

Pentru a putea folosi canalul de notificări, această clasă trebuie inclusă în fișierul

*AndroidManifest.xml* în interiorul intrării *aplication* specificat prin proprietatea *android:name*.

Conținutul fișierului *IrigareManuala.java* este următorul:

package com**.**jetbrains**.**handson**.**mpp**.**hidroapp**;**

**import** androidx**.**appcompat**.**app**.**AppCompatActivity**; import** androidx**.**appcompat**.**widget**.**SwitchCompat**; import** androidx**.**core**.**app**.**NotificationCompat**;**

**import** androidx**.**core**.**app**.**NotificationManagerCompat**;**

**import** android**.**content**.**Intent**; import** android**.**os**.**Bundle**; import** android**.**os**.**SystemClock**; import** android**.**view**.**View**;**

**import** android**.**widget**.**Chronometer**; import** android**.**widget**.**Toast**;**

**import** static com**.**jetbrains**.**handson**.**mpp**.**hidroapp**.**Notification**.**CHANNEL\_1\_ID**;** public class IrigareManuala **extends** AppCompatActivity **{**

private Chronometer chronometer**;** private long pauseOffset**;** private boolean running**;**

private String text**;**

private NotificationManagerCompat notificationManager**;** @Override

protected void onCreate**(**Bundle savedInstanceState**) {**

**super.**onCreate**(**savedInstanceState**);**

setContentView**(**R**.**layout**.**activity\_irigare\_manuala**);**

notificationManager **=** NotificationManagerCompat**.**from**(this);** Intent intent **=** getIntent**();**

text **=** intent**.**getStringExtra**(**MainActivity**.**EXTRA\_TEXT**);**

chronometer **=** findViewById**(**R**.**id**.**chronometerMan**);** chronometer**.**setFormat**(**"Timp: %s"**);** chronometer**.**setBase**(**SystemClock**.**elapsedRealtime**());**

chronometer**.**setOnChronometerTickListener**(new** Chronometer**.**OnChronometerTickListener**() {**

@Override

public void onChronometerTick**(**Chronometer chronometer**) { if((**SystemClock**.**elapsedRealtime**() -** chronometer**.**getBase**())**

**>=** 60000 **\*** 60**) {**//60 sec \*60

chronometer**.**setBase**(**SystemClock**.**elapsedRealtime**());** Toast**.**makeText**(**IrigareManuala**.this,** "Depasit!"**,**

Toast**.**LENGTH\_SHORT**).**show**();**

**}**

**}**

**});**

final SwitchCompat switchCompat **=** findViewById**(**R**.**id**.**switchButtonMan**);**

switchCompat**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

**if (**switchCompat**.**isChecked**()) { if(!**running**) {**

chronometer**.**setBase**(**SystemClock**.**elapsedRealtime**() -**

pauseOffset**);**

chronometer**.**start**();** running **= true;**

**}**

Toast**.**makeText**(**getBaseContext**(),** "Irigare PORNITA"**,** Toast**.**LENGTH\_SHORT**).**show**();**

StartWater startWater **= new** StartWater**(**text**);** startWater**.**doInBackground**();**

sendNotification**(**0**);**

**} else {**

**if(**running**) {**

chronometer**.**stop**();**

pauseOffset **=** SystemClock**.**elapsedRealtime**() -**

chronometer**.**getBase**();**

**}**

running **= false;**

Toast**.**makeText**(**getBaseContext**(),** "Irigare OPRITA"**,** Toast**.**LENGTH\_SHORT**).**show**();**

StopWater stopWater **= new** StopWater**(**text**);** stopWater**.**doInBackground**();**

**}**

**}});**

**}**

private void sendNotification**(**long minute**) {**

String message**;**

**if(** minute **==** 0**) {**

message **=** "Irigarea a pornit!"**;**

**} else {**

message **=** "Irigarea a pornit de " **+** minute **+** " minute!"**;**

**}**

android**.**app**.**Notification notification **= new**

NotificationCompat**.**Builder**(this,** CHANNEL\_1\_ID**)**

**.**setSmallIcon**(**R**.**drawable**.**ic\_notif**)**

**.**setContentTitle**(**"Hidro App"**)**

**.**setContentText**(**message**)**

**.**setPriority**(**NotificationCompat**.**PRIORITY\_HIGH**)**

**.**setCategory**(**NotificationCompat**.**CATEGORY\_MESSAGE**)**

**.**build**();**

notificationManager**.**notify**(**1**,** notification**);**

**}**

**}**

### Activity\_irigare\_automată.xml

Activitatea corespunzătoare procesului de irigare automată prezintă în fișierul

*activity\_irigare\_automată.xml* trei tipuri de elemente grafice: *TextView, Timepicker* și *Button*.

În această activitate se găsesc trei elemente de tip *TextView* și anume: primul element este folosit pentru a memora numele activității curente, *Irigare Automată*; al doilea element este folosit pentru a afișa dacă există programare pentru pornirea sistemului de irigare sau nu. În cazul în care nu există nicio programare, acest element va afișa pe ecran mesajul *Nu există programări*; în cazul în care există programare, acest element va afișa pe ecran mesajul: *Irigare start* urmat de ora de începere a irigării. Cel de al treilea element de acest tip, în cazul în care nu există programări, nu afișează nimic; în caz contrar, mesajul afișat este: *Irigare stop* urmat de ora la care procesul de irigare va fi oprit.

Al doilea tip de element este *Timepicker*. Acesta este utilizat pentru a selecta ora la care se dorește începerea procesului de irigare și ora la care se dorește încetarea acestuia. Una dintre proprietățile importante ale acestui element este *android:timePickerMode* cu ajutorul căreia se poate seta modul de afișare al elementului.



*Fig. 4.7 Modurile de afișare ale elementului TimePicker [*[*https://yourpcfriend.com/wp-content/uploads/2019/06/image-719.png*](https://yourpcfriend.com/wp-content/uploads/2019/06/image-719.png)*]*

Al treilea tip de element prezent în această activitate este *Button*. Cu toate că sunt șase elemente de acest fel, ele sunt grupate în trei categorii. În momentul în care se face o programare, ea se realizează cu ajutorul primelor două butoane: se apasă butonul *PROGRAMARE START*, moment în care se activează *TimePicker-ul*. Pentru a memora ora de începere a programării, se apasă butonul *OK* din dreapta butonului *PROGRAMARE START*.

Pentru a selecta ora la care se dorește oprirea sistemului, se apasă butonul *PROGRAMARE STOP*. Pentru a memora ora de oprire a sistemului, se apasă butonul *OK* din dreapta.

Un alt buton prezent în această activitate este butonul *VALIDEAZĂ PROGRAMAREA*. Ultimul buton prezent în această activitate este butonul *ANULARE*.

Conținutul fișierului *activity\_irigare\_automata.xml* este următorul:

<?xml version=**"1.0"** encoding=**"utf-8"**?>

<RelativeLayout xmlns:android=[**"http://schemas.android.com/apk/res/android"**](http://schemas.android.com/apk/res/android)xmlns:app=[**"http://schemas.android.com/apk/res**](http://schemas.android.com/apk/res-auto)**-**[**auto"**](http://schemas.android.com/apk/res-auto)xmlns:tools=[**"http://schemas.android.com/tools"**](http://schemas.android.com/tools)android:layout\_width=**"match\_parent"** android:layout\_height=**"match\_parent"**

tools:context=**".IrigareAutomata"** android:background=**"@drawable/rain\_backg\_2"**>

<TextView

android:id=**"@+id/textview1"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Irigare Automata"** android:textSize=**"30sp"** android:layout\_centerHorizontal=**"true"** android:layout\_marginTop=**"20dp"**/>

<TextView

android:id=**"@+id/textview2"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Nu exista programare"** android:textSize=**"30sp"** android:layout\_marginTop=**"90dp"** android:layout\_marginLeft=**"60dp"**/>

<TextView

android:id=**"@+id/textview3"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:layout\_marginLeft=**"63dp"** android:textSize=**"30sp"** android:layout\_below=**"@+id/textview2"** />

<TimePicker

android:id=**"@+id/time"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:layout\_centerHorizontal=**"true"** android:layout\_marginTop=**"170dp"** android:timePickerMode=**"spinner"** />

<Button

android:id=**"@+id/button\_set\_time\_start"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Programare Start"** android:textSize=**"20sp"** android:layout\_marginLeft=**"20dp"** android:layout\_marginTop=**"400dp"**/>

<Button

android:id=**"@+id/button\_ok\_start"** android:layout\_width=**"wrap\_content"**

android:layout\_height=**"wrap\_content"** android:text=**"OK"** android:textSize=**"20sp"** android:layout\_marginTop=**"400dp"** android:layout\_marginLeft=**"280dp"**/>

<Button

android:id=**"@+id/button\_set\_time\_stop"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Programare Stop"** android:textSize=**"20sp"** android:layout\_marginLeft=**"20dp"** android:layout\_below=**"@+id/button\_set\_time\_start"**/>

<Button

android:id=**"@+id/button\_ok\_stop"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"OK"** android:textSize=**"20sp"**

android:layout\_below=**"@+id/button\_ok\_start"** android:layout\_marginLeft=**"280dp"**/>

<Button

android:id=**"@+id/button\_validation"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Valideaza Programarea"** android:textSize=**"20sp"** android:layout\_centerHorizontal=**"true"** android:layout\_below=**"@id/button\_set\_time\_stop"**/>

<Button

android:id=**"@+id/button\_cancel"** android:layout\_width=**"wrap\_content"** android:layout\_height=**"wrap\_content"** android:text=**"Anulare"** android:textSize=**"20sp"** android:layout\_centerHorizontal=**"true"**

android:layout\_below=**"@+id/button\_validation"**/>

</RelativeLayout>

Activitatea din punct de vedere grafic arată în felul următor:



*Fig. 4.8 și 4.9 Activitatea de irigare automată înainte de a fi programat și după ce s-a făcut o programare*

### IrigareAutomata.java

Acest fișier conține implementarea funcționalităților elementelor prezentate în activitatea de irigare automată.

În momentul în care am prezentat conținutul fișierului *MainActivity.java* am precizat faptul că legătura dintre activitatea curentă și noua activitate se realizează cu ajutorul clasei *Intent.* Se crează o nouă instanță a clasei *Intent* cu parametrii *this* (adică activitatea curentă) și clasa activității respective, iar pentru a putea trimite conținutul unor variabile în noua activitate, se folosește metoda *putExtra(),* apelată pe instanța clasei *Intent.*

Intent intent **= new** Intent**(this,** IrigareManuala**.**class**);** intent**.**putExtra**(**EXTRA\_TEXT**,** text**);**

Conținutul variabilei trimise din activitatea principală în această activitate este conținutul intrării *EditText* din *activity\_main.xml,* adică adresa IP introdusă.

Pentru a putea prelua conținutul variabilei transmise, în acest fișier, *IrigareAutomata.java*, se crează o instanță a clasei *Intent* și se apelează metoda *getIntent()*.

Intent intent **=** getIntent**();**

textIP **=** intent**.**getStringExtra**(**MainActivity**.**EXTRA\_TEXT**);**

Transmiterea și preluarea conținutului acestei variabile este necesar pentru a putea transmite mai departe din clasa curentă spre clasele care se ocupă de pornirea și oprirea sistemului de irigare, *StartWater.java* respectiv *StopWater.java*.

Cea mai mare parte a explicațiilor referitoare la funcționalitatea elementelor prezente în activitatea de irigare automată a fost făcută în momentul prezentării elementelor prezente în fișierul *activity\_irigare\_automată.xml*.

Câteva precizări legate de funcționalitatea butonului *VALIDEAZĂ PROGRAMAREA* vor fi făcute în rândurile următoare. Acest buton este esențial pentru activitatea de irigare automată pentru că pornirea automată și oprirea automată a sistemului de irigare sunt prezente în interiorul acestui buton. Odată cu pornirea și oprirea sistemului de irigare, aplicația mobilă va trimite o notificare telefonului prin care anunță momentul pornirii și momentul opririi sistemului.

Butonul *ANULARE* este butonul care va anula programarea făcută și va reseta activitatea de irigare automată.

Conținutul fișierului *IrigareAutomată.java* este următorul:

package com**.**jetbrains**.**handson**.**mpp**.**hidroapp**;**

**import** androidx**.**appcompat**.**app**.**AppCompatActivity**; import** androidx**.**core**.**app**.**NotificationCompat**;**

**import** androidx**.**core**.**app**.**NotificationManagerCompat**;**

**import** android**.**content**.**Intent**; import** android**.**os**.**Bundle**; import** android**.**view**.**View**; import** android**.**widget**.**Button**; import** android**.**widget**.**TextView**;**

**import** android**.**widget**.**TimePicker**; import** android**.**widget**.**Toast**;**

**import** java**.**util**.**Calendar**;**

**import** static com**.**jetbrains**.**handson**.**mpp**.**hidroapp**.**Notification**.**CHANNEL\_1\_ID**;** public class IrigareAutomata **extends** AppCompatActivity **{**

private TextView textViewStart**,** textViewStop**;** private TimePicker tp**;**

private NotificationManagerCompat notificationManager**;** private Button buttonTimeStart**,** buttonTimeStop**,** buttonCancel**,**

buttonOKStart**,** buttonOKStop**,** buttonValidation**;**

private int hourStart **= -**1**,** minStart **= -**1**,** hourStop **= -**1**,** minStop **= -**1**;** private int hourRealStart**,** minRealStart**,** hourRealStop**,** minRealStop**;** private long timeStart**,** timeStop**;**

private String textIP**;** private Calendar calendar**;**

@Override

protected void onCreate**(**Bundle savedInstanceState**) { super.**onCreate**(**savedInstanceState**);** setContentView**(**R**.**layout**.**activity\_irigare\_automata**);**

Intent intent **=** getIntent**();**

textIP **=** intent**.**getStringExtra**(**MainActivity**.**EXTRA\_TEXT**);** notificationManager **=** NotificationManagerCompat**.**from**(this);**

textViewStart **= (**TextView**)** findViewById**(**R**.**id**.**textview2**);** textViewStop **= (**TextView**)** findViewById**(**R**.**id**.**textview3**);**

buttonOKStart **= (**Button**)** findViewById**(**R**.**id**.**button\_ok\_start**);** buttonOKStop **= (**Button**)** findViewById**(**R**.**id**.**button\_ok\_stop**);**

buttonOKStart**.**setEnabled**(false);** buttonOKStop**.**setEnabled**(false);** buttonValidation**.**setEnabled**(false);**

tp **= (**TimePicker**)** findViewById**(**R**.**id**.**time**);** tp**.**setEnabled**(false);**

buttonTimeStart **= (**Button**)** findViewById**(**R**.**id**.**button\_set\_time\_start**);** buttonTimeStart**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

tp**.**setEnabled**(true);** tp**.**setOnTimeChangedListener**(new**

TimePicker**.**OnTimeChangedListener**() {**

@Override

public void onTimeChanged**(**TimePicker timePicker**,** int hourOfDay**,** int minute**) {**

hourStart **=** hourOfDay**;** minStart **=** minute**;**

**}**

**});**

buttonOKStart**.**setEnabled**(true);**

**}**

**});**

buttonOKStart**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

textViewStart**.**setText**(**"Irigare start: " **+** hourStart **+** ":" **+**

minStart**);**

buttonTimeStart**.**setEnabled**(false);**

tp**.**setEnabled**(false);**

calendar **=** Calendar**.**getInstance**();**

hourRealStart **=** calendar**.**get**(**Calendar**.**HOUR\_OF\_DAY**);** minRealStart **=** calendar**.**get**(**Calendar**.**MINUTE**);**

timeStart **= ((**hourStart **-** hourRealStart**) \*** 3600 **+ (**minStart

**-** minRealStart**) \*** 60**) \*** 1000**;**

timeStart **=** System**.**currentTimeMillis**() +** timeStart**;**

**}**

**});**

buttonTimeStop **= (**Button**)** findViewById**(**R**.**id**.**button\_set\_time\_stop**);** buttonTimeStop**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

buttonOKStart**.**setEnabled**(false);** tp**.**setEnabled**(true);**

tp**.**setOnTimeChangedListener**(new** TimePicker**.**OnTimeChangedListener**() {**

@Override

public void onTimeChanged**(**TimePicker timePicker**,** int hourOfDay**,** int minute**) {**

hourStop **=** hourOfDay**;** minStop **=** minute**;**

**}**

**});**

buttonOKStop**.**setEnabled**(true);**

**}**

**});**

buttonOKStop**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

textViewStop**.**setText**(**"Irigare stop: " **+** hourStop **+** ":" **+**

minStop**);**

buttonTimeStop**.**setEnabled**(false);** tp**.**setEnabled**(false);**

calendar **=** Calendar**.**getInstance**();**

hourRealStop **=** calendar**.**get**(**Calendar**.**HOUR\_OF\_DAY**);** minRealStop **=** calendar**.**get**(**Calendar**.**MINUTE**);**

timeStop **= ((**hourStop **-** hourRealStop**) \*** 3600 **+ (**minStop **-**

minRealStop**) \*** 60**) \*** 1000**;**

timeStop **=** System**.**currentTimeMillis**() +** timeStop**;** buttonValidation**.**setEnabled**(true);**

**}**

**});**

buttonValidation **= (**Button**)** findViewById**(**R**.**id**.**button\_validation**);** buttonValidation**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

buttonOKStop**.**setEnabled**(false);**

**while (!(**System**.**currentTimeMillis**() ==** timeStart**));** sendNotification**(**0**);** Toast**.**makeText**(**getBaseContext**(),** "Irigare PORNITA"**,**

Toast**.**LENGTH\_SHORT**).**show**();**

StartWater startWater **= new** StartWater**(**textIP**);** startWater**.**doInBackground**();**

**while (!(**System**.**currentTimeMillis**() ==** timeStop**));** sendNotification**(-**1**);** Toast**.**makeText**(**getBaseContext**(),** "Irigare OPRITA"**,**

Toast**.**LENGTH\_SHORT**).**show**();**

StopWater stopWater **= new** StopWater**(**textIP**);** stopWater**.**doInBackground**();**

**}**

**});**

buttonCancel **= (**Button**)** findViewById**(**R**.**id**.**button\_cancel**);** buttonCancel**.**setOnClickListener**(new** View**.**OnClickListener**() {**

@Override

public void onClick**(**View view**) {**

textViewStart**.**setText**(**"Nu exista programare"**);** textViewStop**.**setText**(**""**);**

**}**

**});**

**}**

buttonTimeStart**.**setEnabled**(true);** buttonOKStart**.**setEnabled**(false);**

buttonTimeStop**.**setEnabled**(true);** buttonOKStop**.**setEnabled**(false);**

private void sendNotification**(**long minute**) {**

String message**;**

**if(** minute **==** 0**) {**

message **=** "Irigarea a pornit!"**;**

**} else if (**minute **== -**1**) {**

message **=** "Irigarea s-a oprit!"**;**

**} else {**

message **=** "Irigarea a pornit de " **+** minute **+** " minute!"**;**

**}**

android**.**app**.**Notification notification **= new**

NotificationCompat**.**Builder**(this,** CHANNEL\_1\_ID**)**

**.**setSmallIcon**(**R**.**drawable**.**ic\_notif**)**

**.**setContentTitle**(**"Hidro App"**)**

**.**setContentText**(**message**)**

**.**setPriority**(**NotificationCompat**.**PRIORITY\_HIGH**)**

**.**setCategory**(**NotificationCompat**.**CATEGORY\_MESSAGE**)**

**.**build**();**

notificationManager**.**notify**(**1**,** notification**);**

**}**

**}**

# Concluzii

Tema acestei lucrări este realizarea unui sistem de irigare automată, controlând funcționalitatea sistemului cu ajutorul unei aplicații mobile.

Sistemul propus consta pe partea hardware din: o plăcuță Arduino, un modul NodeMCU ESP8266, un releu, o electrovalvă și un senzor de umiditate a solului.

Pe partea hardware a acestui sistem, în lucrarea de față nu este prezent senzorul de umiditate a solului.

Partea software a sistemului constă într-o aplicație mobilă cu ajutorul căreia se controlează și comandă sistemul. Aplicația prezintă două moduri de a utiliza acest sistem: modul de irigare manuală și modul de irigare autoată. Acest al doilea mod, dă posibilitatea irigării în funcție de doi factori: unul din factori este timpul, și anume se irigă în intervalul de timp selectat; iar al doilea factor este senzorul de umiditate, și anume se poate iriga doar atunci când nivelul de umiditate al solului este sub o anumită limită și durează până în momentul în care nivelul de umiditate atinge un anumit prag.

Pe partea software a acestui sistem, în lucrarea de față nu este realizată opțiunea de a iriga automat în funcție de datele primite de la senzorul de umiditate al solului.

Partea hardware a sistemului prezentat urmează să fie mărit cu încă două componente: un senzor de umiditate al solului și un debitmetru de apă. Prin adăugarea celor două componente, performanța sistemul va fi îmbunătățită considerabil.

Partea software va fi actualizată astfel încât să fie integrate cele două componente hardawe, iar posibilitățile de control și comandă a sistemului vor fi mai multe. Una din posibilitățile viitoare de control este pe baza datelor primite de la senzorul de umiditate al solului. A doua posibilitate viitoare de control este pe bază de cantitatea de apă dorită cu care să se facă irigarea.

Bibliografie

1. I.E.R.C., IERC-European Research Cluster on the Internet of Things, [http://www.internet-of- things-research.eu/pdf/IERC\_Cluster\_Book\_2014\_Ch.3\_SRIA\_WEB.pdf](http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2014_Ch.3_SRIA_WEB.pdf), accesare 28.08.2020
2. Svitla, 10 Best Examples of IoT applications, [https://svitla.com/blog/best-10-examples-of-iot- applications,](https://svitla.com/blog/best-10-examples-of-iot-applications) accesare 28.08.2020
3. T.L. Dragomir, Teoria sistemelor, Curs anul II CTI, Cap. 1, p. 1, an universitar 2017/2018
4. Gardenium, Gardenium.ro-Irigatii si gradina, <https://gardenium.ro/irigatii/controllere>, accesare 28.08.2020
5. Google Play, Aplicatii Android pe Google Play, <https://play.google.com/store/apps/>, accesare 28.08.2020
6. Wikipedia Enciclopedia libera, Wikipedia, https://ro.wikipedia.org/wiki, accesare 29.08.2020
7. Optimus Digital, Optimus Digital, <https://www.optimusdigital.ro/ro>, accesare 29.08.2020
8. Android, Android Developers, https://developer.android.com, accesare 30.08.2020